

Introduction	1
Technology/Licensing Overview	2
Hardware Resources	3
Network Communications	4
Input/Output Interfaces	5
Electrical and Mechanical Specifications	6
Programming Model	7
LonTalk Protocol	8
Appendices	9
Engineering Bulletins	EB
Applications Literature	AL
Glossary	GL
Index	IND

DATA CLASSIFICATION

Product Preview

This heading on a data sheet indicates that the device is in the formative stages or under development at the time of printing of this data book. Please check with Motorola for current status. The disclaimer at the bottom of the first page reads: "This document contains information on a product under development. Motorola reserves the right to change or discontinue this product without notice."

Advance Information

This heading on a data sheet indicates that the device is in sampling, preproduction, or first production stages at the time of printing of this data book. Please check with Motorola for updated information. The disclaimer at the bottom of the first page reads: "This document contains information on a new product. Specifications and information herein are subject to change without notice."

Fully Released

A fully released data sheet contains neither a classification heading nor a disclaimer at the bottom of the first page. This document contains information on a product in full production. Guaranteed limits will not be changed without written notice to your local Motorola Semiconductor Sales Office.

Technical Summary

The Technical Summary is an abridged version of the complete device data sheet that contains the key technical information required to determine the correct device for a specific application. Complete device data sheets for these more complex devices are available from your Motorola Semiconductor Sales Office or authorized distributor.



LONWORKS

Technology Device Data, Rev. 5 Volume I — Device Data

Through the LONWORKS program, Motorola offers the MC143120 and MC143150 Neuron integrated circuits. These integrated circuits are sophisticated VLSI devices for network applications. The *LONWORKS Technology Device Data Book* combines specifications on these parts with a large selection of applications literature. The applications literature is included in this book in order to provide you with suggestions and hints for implementing network solutions in your own applications.


This book is divided into three volumes:

- Volume I — Device Data
- Volume II — Engineering Bulletins from Echelon
- Volume III — Applications Literature

For the most current copy of this data book, please visit our web site at:
<http://motorola.com/lonworks>

Increasingly sophisticated LONWORKS devices are constantly under development. For the latest releases, additional technical information, and pricing, please contact your nearest Motorola Semiconductor Sales Office or authorized distributor. A complete listing of sales offices and distributors is included at the back of this book.

Motorola LONWORKS Application Support
Austin, Texas: (512) 934-7610 FAX: (512) 934-7991
<http://motorola.com/lonworks>
Toulouse, France: 33-561-199175
Hong Kong: 852-2-666-8470
Tokyo, Japan: 81-33-280-8414

Motorola reserves the right to make changes without further notice to any products herein. Motorola makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Motorola assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters can and do vary in different applications. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. Motorola does not convey any license under its patent rights nor the rights of others. Motorola products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Motorola product could create a situation where personal injury or death may occur. Should Buyer purchase or use Motorola products for any such unintended or unauthorized application, Buyer shall indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the design or manufacture of the part. Motorola and  are registered trademarks of Motorola, Inc. Motorola, Inc. is an Equal Opportunity/Affirmative Action Employer.

This IC contains firmware which has license restrictions. Sample Neurons can be obtained from Motorola after signing a developers license agreement with Echelon Corporation. Production procurement of the Neuron Chips can be acquired from Motorola only after signing an OEM license agreement with Echelon Corporation.

Echelon, LON, LonBuilder, LonManager, LonTalk, LonUsers, LONWORKS, Neuron, 3120, 3150, and NodeBuilder are registered trademarks of Echelon Corporation.

LonLink, LonMaker, LONMARK, LONews, and LonSupport are trademarks of Echelon Corporation.

SIDACtor is a trademark of Teccor Electronics, Inc.

CONTENTS

SECTION 1 INTRODUCTION

1–3 (I)

SECTION 2

LONWORKS TECHNOLOGY AND LICENSING OVERVIEW

2–3 (I)

2.1	LONWORKS TECHNOLOGY OVERVIEW AND ARCHITECTURE	2–3 (I)
	MC143120B1DW Neuron Chip Distributed Communications and Control Processor	2–5 (I)
	MC143120E2 Neuron Chip Distributed Communications and Control Processor	2–6 (I)
	MC143120FE2 Neuron Chip Distributed Communications and Control Processor	2–8 (I)
	MC143120LE2 Neuron Chip Distributed Communications and Control Processor	2–10 (I)
	MC143150B1FU1 Neuron Chip Distributed Communications and Control Processor	2–12 (I)
	MC143150B2 Neuron Chip Distributed Communications and Control Processor	2–13 (I)

SECTION 3

Neuron CHIP PROCESSOR FAMILY — HARDWARE RESOURCES

3–3 (I)

3.1	PROCESSING UNITS	3–3 (I)
3.2	MEMORY	3–9 (I)
	3.2.1 Memory Allocation Overview	3–9 (I)
	3.2.2 EEPROM	3–9 (I)
	3.2.3 Static RAM	3–12 (I)
	3.2.4 Preprogrammed ROM	3–12 (I)
	3.2.5 External Memory (MC143150 Only).	3–12 (I)
	3.2.6 Memory Integrity Using Checksums.	3–13 (I)
3.3	INPUT/OUTPUT.	3–16 (I)
	3.3.1 Eleven Bidirectional I/O Pins	3–16 (I)
	3.3.2 Two 16-Bit Timer/Counters	3–16 (I)

SECTION 4

COMMUNICATIONS, CLOCKING, RESET, AND SERVICE

4–3 (I)

4.1	COMMUNICATIONS	4–3 (I)
4.2	COMMUNICATIONS PORT.	4–4 (I)
	4.2.1 Single-Ended Mode	4–5 (I)
	4.2.2 Differential Mode	4–8 (I)
	4.2.3 Special-Purpose Mode.	4–9 (I)
4.3	TWISTED-PAIR TRANSCEIVERS	4–12 (I)
	4.3.1 Direct-Drive	4–12 (I)
	4.3.2 EIA-485	4–13 (I)
	4.3.3 Transformer-Coupled	4–14 (I)
4.4	OTHER TRANSCEIVER EXAMPLES	4–17 (I)
	4.4.1 Powerline Transceivers	4–17 (I)
	4.4.2 Radio Frequency Transceivers	4–17 (I)
4.5	CLOCKING SYSTEM.	4–17 (I)
	4.5.1 Clock Generation	4–17 (I)
4.6	ADDITIONAL FUNCTIONS	4–19 (I)
	4.6.1 Sleep/Wake-Up Circuitry	4–19 (I)
	4.6.2 Reset Function	4–20 (I)
	4.6.3 $\overline{\text{RESET}}$ Pin.	4–21 (I)
	4.6.4 Reset Processes and Timing	4–23 (I)
4.7	$\overline{\text{SERVICE}}$ PIN	4–30 (I)

CONTENTS (CONTINUED)

SECTION 5 INPUT/OUTPUT INTERFACES

5-3 (I)

5.1	HARDWARE CONSIDERATIONS	5-4 (I)
5.2	I/O TIMING ISSUES	5-8 (I)
5.2.1	Scheduler-Related I/O Timing Information	5-8 (I)
5.2.2	Firmware and Hardware Related I/O Timing Information	5-10 (I)
5.3	DIRECT OBJECTS (BIT I/O, BYTE I/O, LEVELDETECT, AND NIBBLE)	5-10 (I)
5.3.1	Bit I/O	5-10 (I)
5.3.2	Byte I/O	5-12 (I)
5.3.3	Leveldetect (Logic Low Level for Input > 200 ns)	5-13 (I)
5.3.4	Nibble I/O	5-13 (I)
5.4	PARALLEL I/O INTERFACE OBJECT	5-15 (I)
5.4.1	Introduction	5-15 (I)
5.4.2	Master/Slave A Mode	5-15 (I)
5.4.3	Slave B Mode	5-19 (I)
5.4.4	Token Passing	5-20 (I)
5.4.5	Handshaking	5-20 (I)
5.4.6	Data Transferring	5-22 (I)
5.5	SERIAL OBJECTS	5-24 (I)
5.5.1	Bitshift I/O	5-24 (I)
5.5.2	I ² C I/O	5-26 (I)
5.5.3	Magcard Input	5-28 (I)
5.5.4	Magtrack1 Input	5-29 (I)
5.5.5	Neurowire (SPI Interface) I/O Object	5-30 (I)
5.5.6	Serial I/O	5-33 (I)
5.5.7	Touch I/O	5-34 (I)
5.5.8	Wiegand Input	5-36 (I)
5.6	TIMER/COUNTER OBJECTS	5-37 (I)
5.6.1	Timer/Counter Input Objects (Dualslope, Edgelog, Infrared, Onetime, Period, Pulsecount Input, Quadrature, Totalcount)	5-38 (I)
5.6.2	Timer/Counter Output Objects (Edgedivide, Frequency, Oneshot, Pulsecount Output, Pulsewidth, Triac, Triggered Count)	5-47 (I)
5.7	MUXBUS I/O	5-55 (I)
5.8	NOTES	5-56 (I)

SECTION 6

Neuron CHIP ELECTRICAL AND MECHANICAL SPECIFICATIONS

6-3 (I)

6.1	INTRODUCTION	6-3 (I)
6.2	ELECTRICAL SPECIFICATIONS	6-4 (I)
6.2.1	Absolute Maximum Ratings	6-4 (I)
6.2.2	Recommended Operating Conditions	6-4 (I)
6.2.3	Electrical Characteristics	6-5 (I)
6.2.4	External Memory Interface Timing — MC143150B1/B2, V _{DD} ± 10%	6-11 (I)
6.2.5	Communications Port Programmable Hysteresis Values	6-14 (I)
6.2.6	Communications Port Programmable Glitch Filter Values	6-14 (I)
6.2.7	Receiver* (End-to-End) Absolute Asymmetry	6-14 (I)
6.2.8	Differential Receiver (End-to-End) Absolute Symmetry	6-15 (I)
6.2.9	Differential Transceiver Electrical Characteristics	6-15 (I)

CONTENTS (CONTINUED)

6.3	MECHANICAL SPECIFICATIONS	6-16 (I)
6.3.1	Pin Descriptions	6-16 (I)
6.3.2	MC143150 Pin Assignment	6-17 (I)
6.3.3	MC143150 Package Dimensions	6-18 (I)
6.3.4	MC143150 Pad Layout	6-19 (I)
6.3.5	MC143120 Pin Assignments	6-20 (I)
6.3.6	MC143120 Package Dimensions	6-21 (I)
6.3.7	MC143120 Pad Layout	6-23 (I)
6.3.8	Sockets for Neuron Chips	6-23 (I)
6.3.9	Test Clips	6-23 (I)

SECTION 7

LonWorks PROGRAMMING MODEL

7-3 (I)

7.1	TIMERS	7-3 (I)
7.2	NETWORK VARIABLES	7-3 (I)
7.3	NETWORK VARIABLE ALIASES	7-6 (I)
7.4	EXPLICIT MESSAGES	7-6 (I)
7.5	SCHEDULER	7-9 (I)
7.6	ADDITIONAL LIBRARY FUNCTIONS	7-10 (I)
7.7	BUILT-IN VARIABLES	7-15 (I)
7.8	MC143120 AND MC143120E2 FIRMWARE EXTENSIONS	7-16 (I)

SECTION 8

LonTalk PROTOCOL

8-3 (I)

8.1	MULTIPLE MEDIA SUPPORT	8-3 (I)
8.2	SUPPORT FOR MULTIPLE COMMUNICATION CHANNELS	8-3 (I)
8.3	COMMUNICATIONS RATES	8-4 (I)
8.4	LonTalk ADDRESSING LIMITS	8-4 (I)
8.5	MESSAGE SERVICES	8-5 (I)
8.6	AUTHENTICATION	8-5 (I)
8.7	PRIORITY	8-6 (I)
8.8	COLLISION AVOIDANCE	8-6 (I)
8.9	COLLISION DETECTION	8-6 (I)
8.10	DATA INTERPRETATION	8-6 (I)
8.11	NETWORK MANAGEMENT AND DIAGNOSTIC SERVICES	8-6 (I)

APPENDIX A

Neuron CHIP DATA STRUCTURES

9-3 (I)

A.1	FIXED READ-ONLY DATA STRUCTURE	9-6 (I)
A.2	THE DOMAIN TABLE	9-11 (I)
A.3	THE ADDRESS TABLE	9-12 (I)
A.3.1	Declaration of Group Address Format	9-13 (I)
A.3.2	Group Address Field Descriptions	9-14 (I)
A.3.3	Declaration of Subnet/Node Address Format	9-14 (I)
A.3.4	Subnet/Node Address Field Descriptions	9-14 (I)
A.3.5	Declaration of Broadcast Address Format	9-15 (I)

A.3.6	Broadcast Address Field Descriptions	9-15 (I)
A.3.7	Declaration of Turnaround Address Format	9-15 (I)
A.3.8	Turnaround Address Field Descriptions	9-15 (I)
A.3.9	Declaration of Neuron ID Address Format	9-16 (I)
A.3.10	Neuron ID Address Field Descriptions	9-16 (I)
A.3.11	Timer Field Descriptions	9-16 (I)
A.4	NETWORK VARIABLE AND ALIAS TABLES	9-17 (I)
A.4.1	Network Variable Configuration Table Field Descriptions	9-19 (I)
A.4.2	Network Variable Alias Table Field Descriptions	9-20 (I)
A.4.3	Network Variable Fixed Table Field Descriptions	9-20 (I)
A.5	THE STANDARD NETWORK VARIABLE TYPE STRUCTURES	9-20 (I)
A.5.1	SNVT Structure Field Descriptions	9-21 (I)
A.5.2	SNVT Descriptor Table Field Descriptions	9-22 (I)
A.5.3	SNVT Table Extension Records	9-23 (I)
A.5.4	SNVT Alias Field Descriptions	9-24 (I)
A.6	THE CONFIGURATION STRUCTURE	9-24 (I)
A.6.1	Configuration Structure Field Descriptions	9-25 (I)
A.6.2	Direct-Mode Transceiver Parameters Field Descriptions	9-28 (I)

APPENDIX B

NETWORK MANAGEMENT AND DIAGNOSTIC SERVICES

9-30 (I)

B.1	NETWORK MANAGEMENT MESSAGES	9-35 (I)
B.1.1	Node Identification Messages	9-35 (I)
B.1.2	Domain Table Messages	9-36 (I)
B.1.3	Address Table Messages	9-38 (I)
B.1.4	Network Variable-Related Messages	9-39 (I)
B.1.5	Memory-Related Messages	9-42 (I)
B.1.6	Special-Purpose Messages	9-47 (I)
B.2	NETWORK DIAGNOSTIC MESSAGES	9-50 (I)
B.3	NETWORK VARIABLE MESSAGES	9-53 (I)

APPENDIX C

EXTERNAL MEMORY INTERFACING

9-57 (I)

APPENDIX D

DESIGN AND HANDLING GUIDELINES

9-63 (I)

D.1	APPLICATION CONSIDERATIONS	9-63 (I)
D.1.1	Termination of Unused Pins	9-63 (I)
D.1.2	Avoidance of Damaging Conditions	9-64 (I)
D.1.3	Power Supply, Ground, and Noise Considerations	9-65 (I)
D.1.4	Transmission Line Termination	9-66 (I)
D.1.5	Decoupling Capacitors	9-67 (I)
D.2	BOARD SOLDERING CONSIDERATIONS	9-68 (I)
D.3	HANDLING PRECAUTIONS AND ELECTROSTATIC DISCHARGE	9-68 (I)
D.4	REDUCTION OF ELECTROMAGNETIC INTERFERENCE	9-72 (I)
D.5	HARDWARE DESIGN	9-73 (I)
D.5.1	Power Distribution	9-74 (I)
D.5.2	EMI	9-76 (I)
D.5.3	Power Interruptions	9-77 (I)
D.5.4	Testing	9-78 (I)
D.5.5	Product Testing and Design Validation	9-78 (I)

D.5.6	Board Layout Issues and Guidelines	9-79 (I)
D.6	CMOS LATCH-UP	9-80 (I)
D.7	RECOMMENDED BYPASS CAPACITOR PLACEMENT	9-82 (I)

APPENDIX E Neuron IC MEMORY MAPS 9-87 (I)

APPENDIX F SUPPORT TOOLS 9-95 (I)

BR1139	LONWORKS® Support Tools	9-96 (I)
M143120DWEVK	Neuron Chip Custom Node Development Board with 32-Pin Socket.	9-100 (I)
M143120FBEVK	Neuron Chip Custom Node Development Board with 44-Pin Socket	9-103 (I)
M143150EVK	Neuron Chip Custom Node Development Board	9-106 (I)
M143204EVK	LonBuilder Direct-Connect Transceiver Board.	9-110 (I)
M143206EVK	Neuron Chip Gizmo 3 I/O Interface Board	9-114 (I)
M143208EVK	I/O Interface Test Board	9-117 (I)
M143232EVK	Voice Compression Board for LONWORKS Networks	9-121 (I)
M143235EVK	Neuron Chip Custom Node Development Board	9-124 (I)
MC143238EVK	Neuron® Chip LiteNode Development Boards	9-126 (I)

APPENDIX G CUSTOMER ALERTS 9-132 (I)

G.1	Neuron EEPROM PROTECTION	9-132 (I)
G.2	Neuron CHIP HANDLING PRECAUTIONS	9-133 (I)
G.3	USING ECHELON'S EMULATOR BOARDS WITH MOTOROLA'S DIRECT-CONNECT BOARD (M143204EVK).	9-133 (I)
G.4	LonBuilder 3.0 SOFTWARE REVISION	9-134 (I)

APPENDIX H Neurowire (SPI INTERFACE) COMPATIBLE DEVICES 9-136 (I)

APPENDIX I USAGE GUIDELINES FOR ECHELON TRADEMARKS 9-139 (I)

SECTION EB ENGINEERING BULLETINS FROM ECHELON EB-1 (II)

EB146	Neuron Chip Quadrature Input Function Interface	EB-3 (II)
EB147	LONWORKS Installation Overview	EB-10 (II)
EB148	Enhanced Media Access Control with Echelon's LonTalk Protocol	EB-27 (II)
EB149	Optimizing LonTalk Response Time.	EB-33 (II)
EB151	Scanning a Keypad with the Neuron Chip	EB-38 (II)
EB153	Driving a Seven-Segment Display with the Neuron Chip.	EB-43 (II)
EB155	Analog-to-Digital Conversion with the Neuron Chip.	EB-55 (II)
EB157	Creating Neuron C Applications with the Gizmo 3.	EB-82 (II)
EB161	LonTalk Protocol.	EB-117 (II)
EB167	A Hybrid System for Fast Synchronized Response	EB-144 (II)
EB168	EIA-232C Serial Interfacing with the Neuron Chip.	EB-163 (II)

EB169	LONWORKS 78 kbps Self-Healing Ring Architecture.	EB-173 (II)
EB172	LONWORKS Custom Node Development	EB-179 (II)
EB173	The SNVT Master List	EB-195 (II)
EB174	Junction Box and Wiring Guidelines for Twisted Pair LONWORKS Networks	EB-223 (II)
EB176	File Transfer	EB-240 (II)
EB177	LONWORKS Power Line SCADA Systems.	EB-253 (II)
EB178	Developing a Network Driver for the PC LonTalk Adapter	EB-265 (II)
EB179	Determinism in Industrial Computer Control Network Applications	EB-277 (II)

SECTION AL APPLICATIONS LITERATURE

AL-1 (III)

AN781A	Revised Data Interface Standards	AL-3
AN1208	Parallel I/O Interface to the Neuron® Chip	AL-9 (III)
AN1211	Interfacing DACs and ADCs to the Neuron® IC	AL-34 (III)
AN1216	Setback Thermostat Design Using the Neuron® IC	AL-44 (III)
AN1225	Fuzzy Logic and the Neuron® Chip.	AL-55 (III)
AN1247	MC683xx to Neuron® Chip Parallel I/O Interface	AL-74 (III)
AN1248	Programmable Peripheral	AL-85 (III)
AN1250	Low Cost PC Interface to LONWORKS®-Based Nodes.	AL-101 (III)
AN1251	Programming the MC143120 Neuron® IC.	AL-112 (III)
AN1252	MIP Guidelines and Design Issues	AL-145 (III)
AN1266	LONWORKS® Distributed Node Crane Demonstration	AL-160 (III)
AN1276	Installation of Neuron® Chip-Based Products	AL-175 (III)
AN1278	LONWORKS® Software Review.	AL-191 (III)
AN1715	Fiber Optic LONWORKS® Network Control Products from Raytheon Electronics	AL-197 (III)

LIST OF TABLES

Table 1-1.	Neuron IC Family	1-5 (I)
Table 1-2.	Neuron IC Specifications	1-5 (I)
Table 1-3.	LonBuilder Firmware Supported	1-5 (I)
Table 1-4.	NodeBuilder Firmware Supported	1-5 (I)
Table 3-1.	Comparison of Neuron Chip Processors	3-3 (I)
Table 3-2.	Register Set	3-6 (I)
Table 3-3.	Program Control Instruction Timings	3-7 (I)
Table 3-4.	Memory/Stack Instruction Timings	3-8 (I)
Table 3-5.	ALU Instruction Timings	3-8 (I)
Table 3-6.	External Memory Interface Pins	3-13 (I)
Table 3-7.	Recovery Action Bit Masks.	3-14 (I)
Table 4-1.	Transceiver Types	4-3 (I)
Table 4-2.	Communications Port Pin Characteristics	4-4 (I)
Table 4-3.	Single-Ended and Differential Network Data Rates	4-5 (I)
Table 4-4.	Receiver Jitter Tolerance Windows	4-8 (I)
Table 4-5.	Special-Purpose Mode Transmit and Receive Status Bits.	4-11 (I)
Table 4-6.	Echelon Transceiver Products	4-15 (I)
Table 4-7.	Twisted-Pair Transformer Manufacturers for 78 kbps and 1.25 Mbps	4-15 (I)
Table 4-8.	Echelon's Powerline Transceivers	4-17 (I)
Table 4-9.	Typical Clock Generator Component Values	4-18 (I)
Table 4-10.	Typical Start-Up Times.	4-19 (I)
Table 4-11.	Comparison of Motorola Low-Voltage Detector ICs.	4-23 (I)
Table 4-12.	Time Required for MC143120 to Perform Reset Sequence	4-27 (I)
Table 4-13.	Time Required for MC143150 to Perform Reset Sequence	4-28 (I)
Table 5-1.	Summary of Direct I/O Objects.	5-4 (I)
Table 5-2.	Summary of Parallel I/O Objects	5-4 (I)
Table 5-3.	Summary of Serial I/O Objects.	5-5 (I)
Table 5-4.	Summary of Timer/Counter Input Objects	5-5 (I)
Table 5-5.	Summary of Timer/Counter Output Objects.	5-6 (I)
Table 5-6.	Timer/Counter Resolution and Maximum Range	5-56 (I)
Table 5-7.	Timer/Counter Square Wave Output	5-57 (I)
Table 5-8.	Timer/Counter Pulsetrain Output	5-57 (I)
Table 8-1.	LonTalk Protocol Layering	8-3 (I)
Table A-1.	Data Structure and Memory Location	9-5 (I)
Table A-2.	Encoding of Timer Field Values (ms)	9-17 (I)
Table A-3.	Transceiver Bit Rate (kbps) as a Function of comm_clock and input_clock	9-26 (I)
Table D-1.	Recommended Capacitor Placement	9-83 (I)

LIST OF FIGURES

Figure 1-1.	MC143150 Neuron Chip Simplified Block Diagram	1-3 (I)
Figure 1-2.	MC143120 Neuron Chip Simplified Block Diagram	1-4 (I)
Figure 2-1.	Typical Node Block Diagram	2-3 (I)
Figure 2-2.	The MC143150 or MC143120 in a LONWORKS Network	2-4 (I)
Figure 3-1.	Neuron Chip Block Diagram.	3-4 (I)
Figure 3-2.	Processor Organization Memory Allocation	3-5 (I)
Figure 3-3.	Processor/Memory Activity During One of the Three System Clock Cycles of a Minor Cycle	3-6 (I)
Figure 3-4.	Base Page Memory Layout	3-7 (I)
Figure 3-5.	MC143150 Processor Memory Map	3-11 (I)
Figure 3-6.	MC143120B1 Processor Memory Map	3-11 (I)
Figure 3-7.	MC143120E2/FE2/LE2 Processor Memory Map	3-11 (I)
Figure 3-8.	Timer/Counter Circuits	3-16 (I)
Figure 4-1.	Internal Transceiver Block Diagram	4-4 (I)
Figure 4-2.	Single-Ended Mode Configuration	4-5 (I)
Figure 4-3.	Single-Ended Mode Data Format.	4-6 (I)
Figure 4-4.	Packet Timing.	4-7 (I)
Figure 4-5.	Differential Mode	4-9 (I)
Figure 4-6.	Differential Mode Data Format.	4-9 (I)
Figure 4-7.	Special-Purpose Mode Data Format	4-10 (I)
Figure 4-8a.	Simple Direct-Connect Network Interface (Used with Direct Mode Differential)	4-13 (I)
Figure 4-8b.	Low Cost/Small Size Network	4-13 (I)
Figure 4-9.	EIA-485 Twisted-Pair Interface (Used with Single-Ended Mode)	4-14 (I)
Figure 4-10.	Twisted-Pair Topologies.	4-15 (I)
Figure 4-11.	Basic Transformer with Signal Conditioning	4-16 (I)
Figure 4-12.	Neuron Chip Clock Generator Circuit.	4-18 (I)
Figure 4-13.	Test Point Levels for CLK1 Duty Cycle Measurements	4-18 (I)
Figure 4-14.	Example of Reset Circuit	4-22 (I)
Figure 4-15.	Reset Timing Diagram for the Neuron Chip.	4-22 (I)
Figure 4-16.	Reset Timeline for MC143120 and MC143150	4-24 (I)
Figure 4-17a.	Power Up	4-29 (I)
Figure 4-17b.	Reset After Power Up.	4-29 (I)
Figure 4-18.	SERVICE Pin Circuit	4-31 (I)
Figure 4-19.	Buffers SERVICE Pin Message Written.	4-32 (I)
Figure 5-1.	Neuron Chip Timer/Counter External Connections	5-3 (I)
Figure 5-2.	Summary of I/O Objects.	5-7 (I)
Figure 5-3.	Synchronization of External Signals.	5-8 (I)
Figure 5-4.	when-Clause to when-Clause Latency, t_{ww} and Scheduler Overhead Latency, t_{sol}	5-9 (I)
Figure 5-5.	Bit I/O	5-10 (I)
Figure 5-6.	Bit Input Latency Values	5-11 (I)
Figure 5-7.	Bit Output Latency Values	5-11 (I)
Figure 5-8.	Byte I/O	5-12 (I)
Figure 5-9.	Byte Input Latency Values	5-12 (I)
Figure 5-10.	Byte Output Latency Values.	5-12 (I)
Figure 5-11.	Leveldetect Input Latency Values	5-13 (I)
Figure 5-12.	Nibble I/O	5-14 (I)
Figure 5-13.	Nibble Input Latency Values	5-14 (I)
Figure 5-14.	Nibble Output Latency Values	5-14 (I)
Figure 5-15.	Parallel I/O — Master and Slave A	5-15 (I)
Figure 5-16.	Master Mode Timing.	5-16 (I)
Figure 5-17.	Slave A Mode Timing.	5-17 (I)

LIST OF FIGURES (CONTINUED)

Figure 5-18.	Parallel I/O Master/Slave B (Neuron Chip as Memory-Mapped I/O Device)	5-20 (I)
Figure 5-19.	Slave B Mode Timing	5-21 (I)
Figure 5-20.	Handshake Protocol Sequence Between the Master and the Slave	5-22 (I)
Figure 5-21.	Bitshift I/O	5-24 (I)
Figure 5-22.	Bitshift Input Latency Values	5-25 (I)
Figure 5-23.	Bitshift Output Latency Values	5-26 (I)
Figure 5-24.	I ² C I/O Object	5-27 (I)
Figure 5-25.	Magcard Input Object	5-28 (I)
Figure 5-26.	Magtrack1 Input Object	5-29 (I)
Figure 5-27.	Neurowire I/O	5-30 (I)
Figure 5-28.	Neurowire (SPI) Master Timing	5-31 (I)
Figure 5-29.	Neurowire (SPI) Slave Timing	5-32 (I)
Figure 5-30.	Serial Input Object	5-33 (I)
Figure 5-31.	Serial Output	5-34 (I)
Figure 5-32.	Touch I/O Object	5-35 (I)
Figure 5-33.	Wiegand Input Object	5-37 (I)
Figure 5-34.	<i>when</i> Statement Processing Using the Ontime Input Function	5-38 (I)
Figure 5-35.	Dualslope Input Object	5-39 (I)
Figure 5-36.	Edgelog Input Object	5-40 (I)
Figure 5-37.	Infrared Input Object	5-41 (I)
Figure 5-38.	Ontime Latency Values	5-42 (I)
Figure 5-39.	Period Input Latency Values	5-43 (I)
Figure 5-40.	Pulse Count Input Latency Values	5-44 (I)
Figure 5-41.	Quadrature Input Latency Values	5-45 (I)
Figure 5-42.	Totalcount Input Latency Values	5-46 (I)
Figure 5-43.	Edgedivide Output Object	5-47 (I)
Figure 5-44.	Frequency Output Latency Values	5-48 (I)
Figure 5-45.	Oneshot Output Latency Values	5-49 (I)
Figure 5-46.	Pulsecount Output	5-50 (I)
Figure 5-47.	Pulsewidth Output Latency Values	5-51 (I)
Figure 5-48.	Triac Output Latency Values (Sheet 1 of 2)	5-52 (I)
Figure 5-48.	Triac Output Latency Values (Sheet 2 of 2)	5-53 (I)
Figure 5-49.	Triggered Count Output Latency Values	5-54 (I)
Figure 5-50.	Muxbus I/O Object	5-55 (I)
Figure 6-1.	External Memory Interface Timing Diagram	6-12 (I)
Figure 6-2.	Signal Loading for Timing Specifications Unless Otherwise Specified	6-13 (I)
Figure 6-3.	Test Point Levels for \bar{E} Pulse Width Measurements	6-13 (I)
Figure 6-4.	Drive Levels and Test Point Levels for Timing Specifications Unless Otherwise Specified	6-13 (I)
Figure 6-5.	Test Point Levels for Three-State-to-Driven Time Measurements	6-13 (I)
Figure 6-6.	Signal Loading for Driven-to-Three-State Time Measurements	6-13 (I)
Figure 6-7.	Test Point Levels for Driven-to-Three-State Time Measurements	6-13 (I)
Figure 6-8.	Receiver Input Waveform	6-14 (I)
Figure 6-9.	Communications Port Signal Input for Table 6.2.8	6-15 (I)
Figure B-1.	Network Variable Message Structure	9-54 (I)
Figure C-1.	EPROM Memory Interface	9-57 (I)
Figure C-2.	MC143150B1FU1 External Memory Interface with 32 Kbyte EPROM and 24 Kbyte RAM	9-58 (I)
Figure C-3.	32K Flash	9-60 (I)
Figure C-4.	32K Flash/24K SRAM	9-61 (I)
Figure D-1.	CMOS Inverter	9-63 (I)

LIST OF FIGURES (CONTINUED)

Figure D-2.	Digital Input	9-65 (I)
Figure D-3.	Digital I/O	9-65 (I)
Figure D-4.	Termination Resistors at the Receiver	9-67 (I)
Figure D-5.	Termination Resistors at Both the Line Driver and Receiver	9-67 (I)
Figure D-6.	Switching Currents for $C_L < 5$ pF	9-68 (I)
Figure D-7.	Switching Currents for $C_L = 50$ pF	9-68 (I)
Figure D-8.	Networks for Minimizing ESD and Reducing CMOS Latch-Up Susceptibility	9-70 (I)
Figure D-9.	Typical Manufacturing Work Station	9-70 (I)
Figure D-10.	Inductance Creates Noise	9-74 (I)
Figure D-11.	Noise With and Without Bypass Capacitors	9-74 (I)
Figure D-12.	Reducing Noise by Adding Additional Capacitors	9-75 (I)
Figure D-13.	Choosing a Bulk Capacitor	9-75 (I)
Figure D-14.	Choosing a High-Frequency Bypass Capacitor	9-76 (I)
Figure D-15.	Radiated EMI of a Large Loop Area versus a Smaller Loop Area	9-77 (I)
Figure D-16.	Stress Hardware to Emulate Thermo-Mechanical Problems and Aging	9-78 (I)
Figure D-17.	Using Ferrite Beads at the Power Connector Input	9-79 (I)
Figure D-18.	CMOS Wafer Cross Section	9-81 (I)
Figure D-19.	Latch-Up Circuit Schematic	9-81 (I)
Figure D-20.	Latch-Up Due to a Negative Voltage Spike	9-81 (I)
Figure D-21.	MC143150 and MC143120 Recommended Bypass Capacitor Configuration	9-84 (I)
Figure D-22.	Minimum Recommended Capacitor Placement (Sheet 1 of 2)	9-85 (I)
Figure D-22.	Minimum Recommended Capacitor Placement (Sheet 2 of 2)	9-86 (I)
Figure E-1.	Six Major Memory Structures	9-87 (I)
Figure E-2.	Neuron Fixed Structure	9-88 (I)
Figure E-3.	Neuron Chip Domain Table	9-89 (I)
Figure E-4.	Neuron Chip Address Table	9-90 (I)
Figure E-5.	Network Variable Tables	9-91 (I)
Figure E-6.	SNVT Structures	9-92 (I)
Figure E-7.	Neuron Configuration Structure	9-93 (I)
Figure F-1.	Evaluation and I/O Interface Boards	9-99 (I)
Figure F-2.	M143120DWEVK Schematic Diagram	9-101 (I)
Figure F-3.	M143120FBEVK Schematic Diagram	9-104 (I)
Figure F-4a.	M143150EVK Schematic Diagram	9-107 (I)
Figure F-4b.	9-108 (I)
Figure F-5.	Direct-Connect Transceiver Board	9-111 (I)
Figure F-6.	LonBuilder Developer's Workbench	9-111 (I)
Figure F-7.	Pinout	9-111 (I)
Figure F-8.	Connecting to a LonBuilder Developer's Workbench	9-112 (I)
Figure F-9.	Connecting to a Custom Node	9-113 (I)
Figure F-10.	LonWORKS Development Tools	9-115 (I)
Figure F-11.	M143206EVK Functional Diagram (Gizmo 3)	9-115 (I)
Figure F-12.	Schematic Diagram of M143206EVK (Gizmo 3) – Rev. 2.5	9-116 (I)
Figure F-13.	LonWORKS Development Tools	9-118 (I)
Figure F-14.	Voice on a Distributed-Control Network in Homes, Offices, and Factories	9-122 (I)
Figure F-15.	9-123 (I)
Figure F-16.	9-127 (I)
Figure F-17.	MC143238EVK 2-Channel — I/O Node	9-128 (I)
Figure F-18.	MC143239EVK H-Bridge Motor Node	9-129 (I)
Figure F-19.	MC143240EVK 10-Bit A/D Node	9-130 (I)
Figure F-20.	LiteNode Kit Connectors	9-131 (I)

SECTION 1 INTRODUCTION

The Motorola MC143150 and MC143120 Neuron Chips are sophisticated very large-scale integration (VLSI) devices that make it possible to implement low-cost local operating network applications. Through a unique combination of hardware and firmware, they provide all the key functions necessary to process inputs from sensors and control devices intelligently, and propagate control information across a variety of network media. The MC143150 and MC143120, with the LonBuilder[®] Developer's Workbench or the NodeBuilder[®] Development Tool, offer the system designer:

- Easy implementation of distributed sense and control networks.
- Flexible reconfiguration capability after network installation.
- Management of LonTalk[®] protocol messages on the network.
- An object-oriented high level environment for system development.

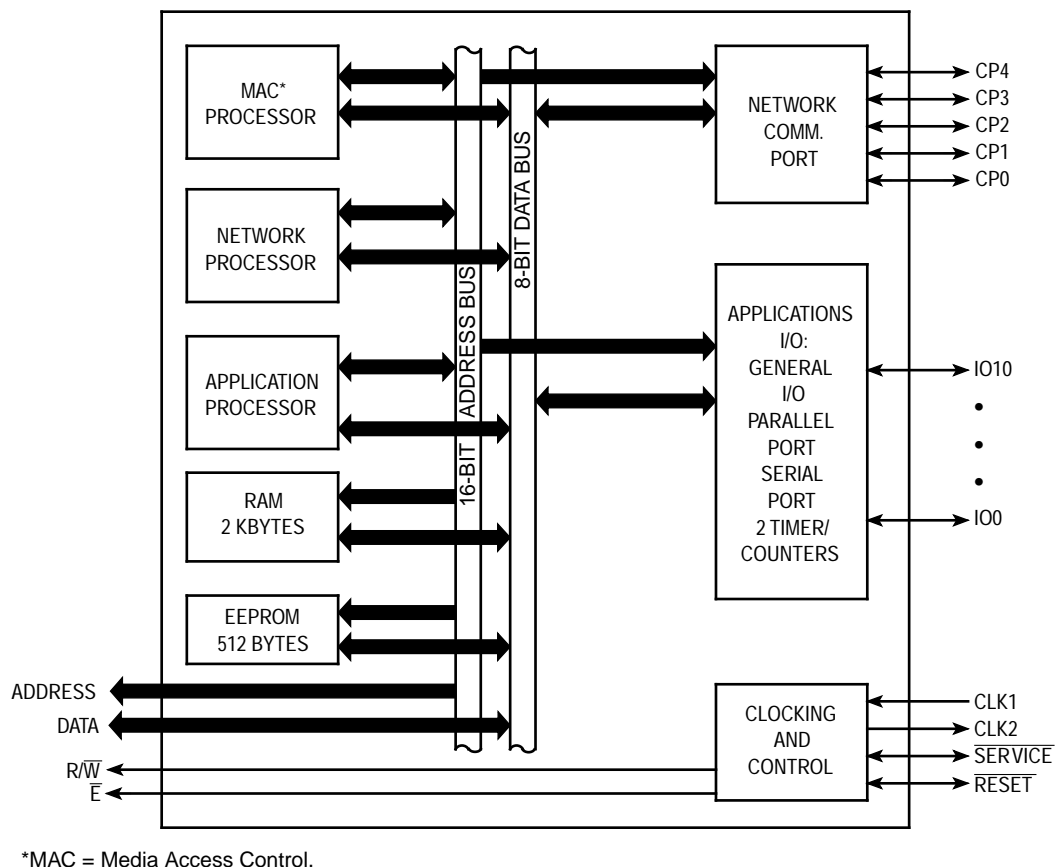
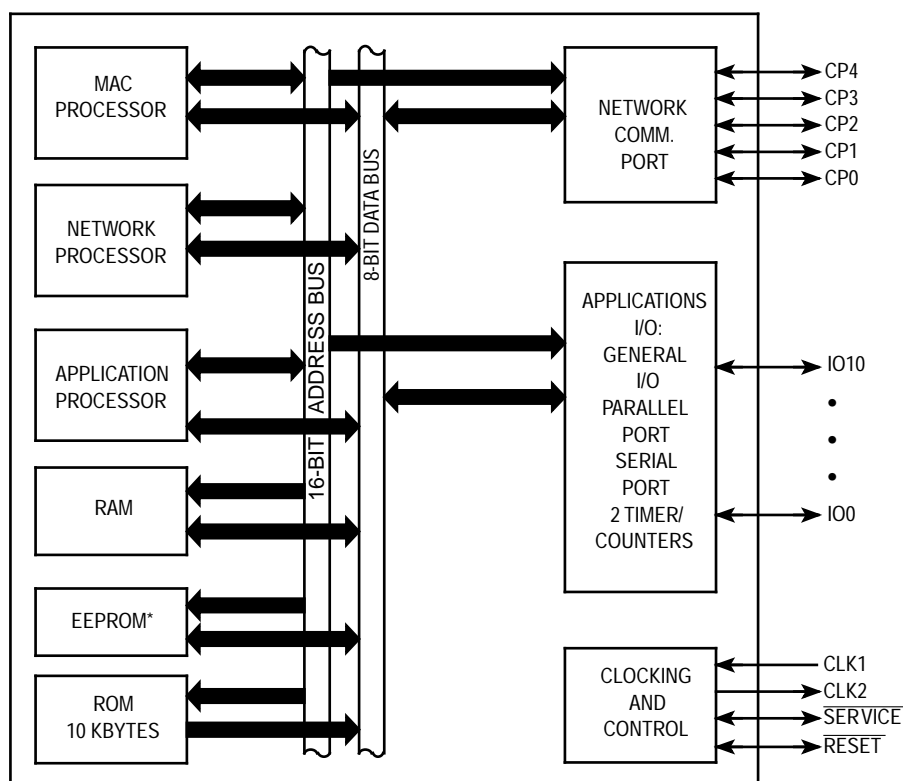


Figure 1-1. MC143150 Neuron Chip Simplified Block Diagram



*See Table 1-1.

Figure 1-2. MC143120 Neuron Chip Simplified Block Diagram

The MC143150 is designed for sense and control systems that require large application programs. An external memory interface allows the system designer to use 42K of the available 64K of address space for application program storage. The MC143150 has no ROM on the device. The communications protocol, developer's operating system, and I/O device driver object code are provided with Echelon's LonBuilder Developer's Workbench and NodeBuilder Development Tool. The protocol and application object code can be stored in external ROM, flash, or other nonvolatile memory.

The Neuron Chip is available in multiple versions. See Table 1-1 for the 3150[®] and 3120[®] Neuron Chip family.

Throughout this data book, the device numbers "MC143120" and "MC143150" refer to the Motorola 3120 and 3150 families of Neuron Chips. Only if Toshiba is being referenced, will the data be applied to the particular Toshiba Neuron Chip.

Table 1-1. Neuron IC Family

Device*	RAM (Bytes)	EEPROM (Bytes)	ROM (Bytes)	Firmware Version	Model No.
3150 Neuron Chip					
MC143150B1	2K	512	—	N/A	0
MC143150B2	2K	512	—	N/A	0
3120 Neuron Chip (Internal ROM = 10,240 bytes)					
MC143120B1	1K	512	10K	4	8
MC143120E2	2K	2K	10K	6	A
	1K	1K	10K	6	9
MC143120FE2	2K	2K	10K	7	A
	2K	2K	10K	9	A
MC143120LE2	2K	2K	10K	6	A

*Refer to Toshiba as an alternate source.

Table 1-2. Neuron IC Specifications

Device*	Max Clock Rate (MHz)	Neuron Chip Access Time (ns)	Supply Voltage Range (V)	EEPROM Programming Temperature Range (°C)	Typical I _{DD} (mA)	Typical Sleep Mode I _{DD} (μA)
3150 Neuron Chip						
MC143150B1	10	130	4.5 – 5.5	– 40 to + 85	15	15
MC143150B2	10	130	4.5 – 5.5	– 40 to + 85	15	15
3120 Neuron Chip						
MC143120B1	10	—	4.5 – 5.5	– 40 to + 85	14	9
MC143120E2	10	—	4.5 – 5.5	– 40 to + 85	16	9
MC143120FE2	20	—	4.5 – 5.5	– 40 to + 85	32	9
MC143120LE2	5	—	2.8 – 3.3	– 40 to + 85	4	4

*Refer to Toshiba as an alternate source.

Table 1-3. LonBuilder Firmware Supported

LonBuilder Software Version	3150 Firmware Supported	3120 Firmware Supported	3120E1/E2/LE2 Firmware Supported	3120FE2 Firmware Supported
3.01	3–7	3, 4	6	7
3.0	3–6	3, 4	6	Not Supported
2.2	2–4	2, 3, (4) (See Note)	Not Supported	Not Supported
2.1	2, 3	2, 3	Not Supported	Not Supported

NOTE: Version 4 support available through a patch on Echelon's web site.

Table 1-4. NodeBuilder Firmware Supported

Node Builder Software Version	3150 Firmware Supported	3120 Firmware Supported	3120E1/E2 Firmware Supported	3120FE2 Firmware Supported
1.0	3–6	3, 4	6	Not Supported
1.5	3–6	3, 4	6	7

Both parts have an 11-pin I/O interface with integrated hardware and firmware for connecting to motors, valves, display drivers, A/D converters, pressure sensors, thermistors, switches, relays, triacs, tachometers, other microprocessors, modems, etc. They each have three processors, of which two interact with a communication subsystem to make the transfer of information from node to node in a distributed control system an automatic process. These devices make it possible to rapidly implement many applications. These include: distributed sense and control systems, instrumentation, machine automation, process control, diagnostic equipment, environmental monitoring and control, power distribution and control, production control, lighting control, building automation and control, security systems, data collection/acquisition, robotics, home automation, consumer electronics, and automotive electronics.

The Neuron Chips can send and receive information on either the 5-pin communications port or the 11-pin applications I/O port.

Features

- Three 8-bit pipelined processors
 - Selectable input clock rates: 625 kHz, 1.25 MHz, 2.5 MHz, 5 MHz, 10 MHz
 - (MC143120FE2 operates at clock rates of up to 20 MHz)
- On-chip memory
 - 2 Kbytes static RAM (MC143150, MC143120E2, MC143120FE2, and MC143120LE2)
 - 1 Kbyte static RAM (MC143120B1)
 - 512 bytes EEPROM (MC143150 and MC143120B1)
 - 2 Kbytes EEPROM (MC143120E2, MC143120FE2, and MC143120LE2)
 - 10 Kbytes ROM (all MC143120s)
- 11 programmable I/O pins
 - 34 selectable modes of operation
 - Programmable pull-ups (IO4 – IO7)
 - 20 mA current sink (IO0 – IO3)
- Two 16-bit timer/counters for frequency and timer I/O
- Up to 15 software timers
- Sleep mode for reduced current consumption while retaining operating state
- Network communications port
 - Single-ended mode
 - Differential mode
 - Selectable transmission rates: 0.6 kbps to 1.25 Mbps
 - 40 mA current output for differentially driving twisted-pair networks
 - Optional collision detect input
 - Relay no longer needed for differential communications
- Firmware
 - LonTalk protocol conforming to 7-layer OSI reference model
 - I/O drivers
 - Event-driven task scheduler
- Service pin for remote identification and diagnostics
- Unique 48-bit internal Neuron ID stored redundantly in every device
- Channel capacity: 560 packets/second throughput sustained; 700 packets/second peak at 10 MHz
- Built-in low-voltage inhibit (LVI) for added EEPROM protection (MC3150B2, MC3120FE2, MC3120LE2, and MC3120E2)

Technology/Licensing Overview **2**

SECTION 2

LONWORKS TECHNOLOGY AND LICENSING OVERVIEW

2.1 LONWORKS TECHNOLOGY OVERVIEW AND ARCHITECTURE

LONWORKS technology is a complete platform for implementing control network systems. These networks consist of intelligent devices or *nodes* that interact with their environment, and communicate with one another over a variety of communications *media* using a common, message-based control *protocol*.

LONWORKS technology includes all of the elements required to design, deploy, and support control networks, specifically the following components:

- MC143150 and MC143120 Neuron Chips
- LonTalk Protocol
- LONWORKS Transceivers
- LonBuilder and NodeBuilder Development Tools

The Motorola Neuron Chip is a VLSI component that performs the network and application-specific processing within a node. A node typically consists of a Neuron Chip, a power source, a transceiver for communicating over the network medium, and circuitry for interfacing to the device being controlled or monitored. The specific circuitry will depend on the networking medium and application. See Figures 2-1 and 2-2.

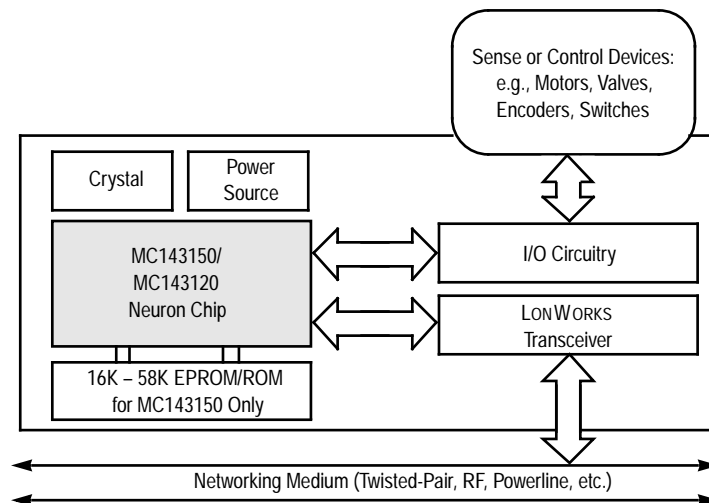


Figure 2-1. Typical Node Block Diagram

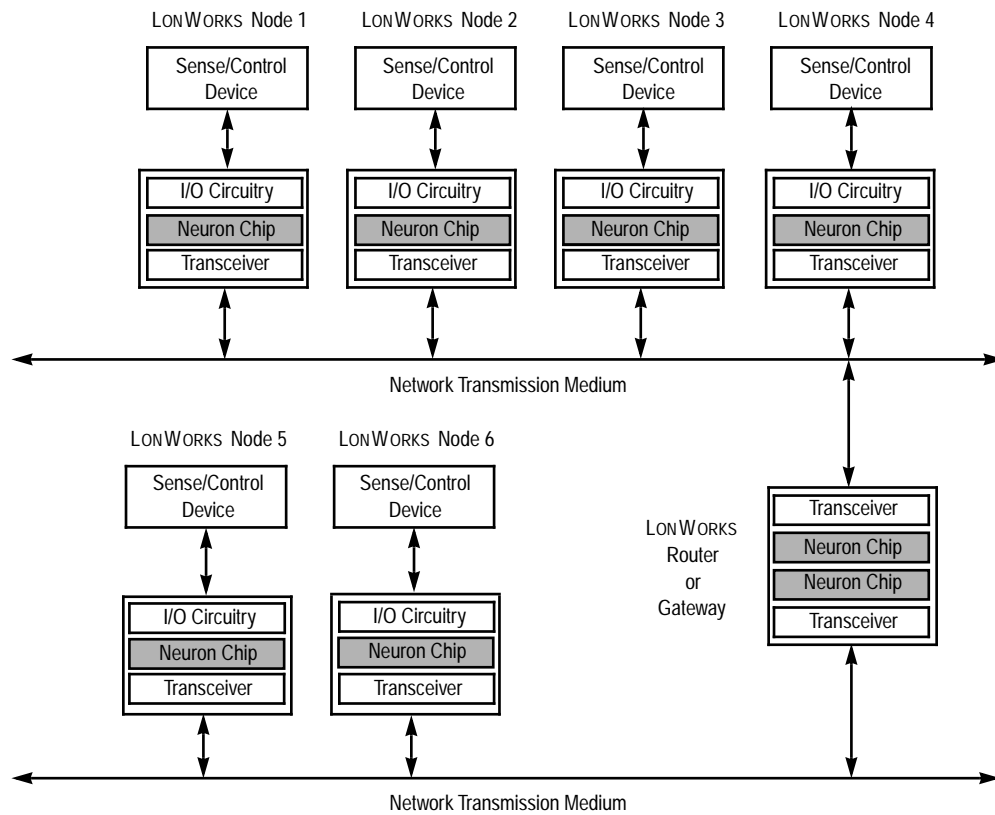


Figure 2-2. The MC143150 or MC143120 in a LonWorks Network

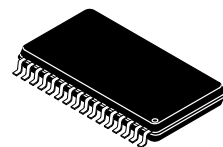
Neuron[®] Chip Distributed Communications and Control Processor

The MC143120B1 is a communications and control processor which enables the development of interoperable products. It provides systems designers with many features to accelerate product development for distributed sense and control applications. Services at every layer of the OSI networking model are implemented in the included LonTalk[®] firmware-based protocol and are easily and optionally invoked. In addition, 24 I/O models are integrated with ROM to provide simplified sense and control device interfacing.

The MC143120B1 is designed for maximum clock operation of 10 MHz over a temperature range of – 40 to + 85°C including EEPROM writes.

- Three 8-Bit Pipelined Processors for Concurrent Processing of Application Code and Network Packets
- 11-Pin I/O Port Programmable in 24 Models for Fast Application Program Development
- Two 16-Bit Timer/Counters for Measuring and Generating I/O Device Waveforms
- 5-Pin Communications Port That Supports Direct Connect and Network Transceiver Interface
- 1024 Bytes of Static RAM for Buffering Network and Application Data
- 512 Bytes of EEPROM with On-Chip Charge Pump for Address, Binding Data, and Application Code
- Programmable Pull-Ups on IO4 – IO7 and 20 mA Sink Current on IO0 – IO3
- Unique 48-Bit ID Number in Every Device to Facilitate Network Installation and Management — ID Number Stored Redundantly for Guaranteed 20-Year Data Retention
- 10 Kbytes ROM
- 32-Pin SOG Package
- Low Operating Current: 15 mA (Typical) at 10 MHz Frequency
3 mA (Typical) at 625 kHz Frequency
- Sleep Mode Operation Reduced Current Consumption (15 μ A Typical)
- 0.8 μ Manufacturing Process

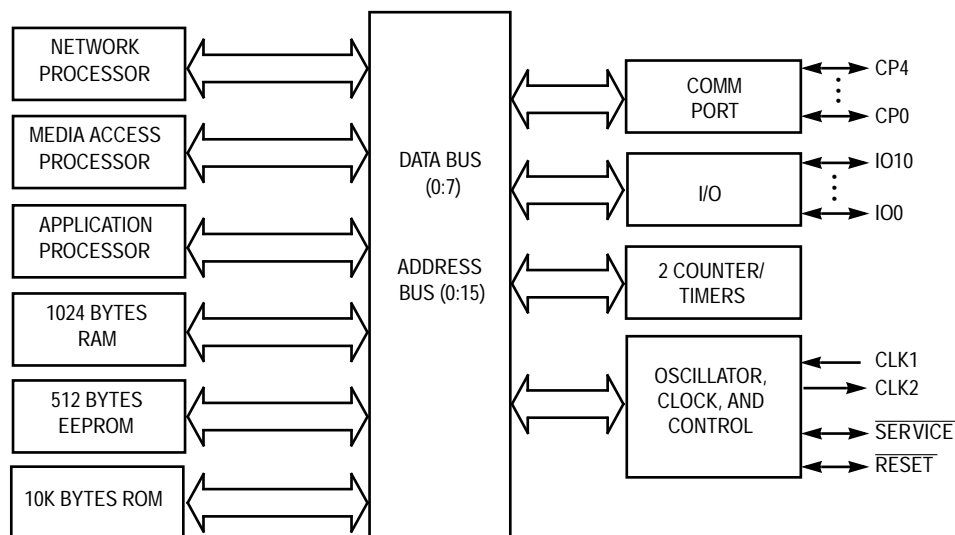
MC143120B1



DW SUFFIX
SOG PACKAGE
CASE 1116-0

ORDERING INFORMATION
MC143120B1DW SOG Package

BLOCK DIAGRAM

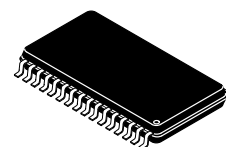


MC143120E2

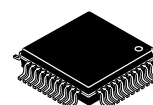
Neuron[®] Chip Distributed Communications and Control Processor

The MC143120E2 contains the LonTalk[®] protocol in 10K ROM. It differs from the MC143120B1 in that it contains 21 I/O models. Bit Shift, Serial Input, and Serial Output I/O models are not in the ROM firmware, but can be loaded into EEPROM by the LonBuilder[®] tool if needed. In addition, 21 I/O models are integrated with ROM to provide simplified sense and control device interfacing.

- Three 8-Bit Pipelined Processors for Concurrent Processing of Application Code and Network Packets
- 11-Pin I/O Port Programmable in 21 Models for Fast Application Program Development
- Two 16-Bit Timer/Counters for Measuring and Generating I/O Device Waveforms
- 5-Pin Communications Port that Supports Direct Connect and Network Transceiver Interfaces
- 2048 Bytes of Static RAM for Buffering Network Data and Storing Network Variables
- 2048 Bytes of EEPROM for Flexible Storage of Address and Binding Data
- Programmable Pull-Ups on IO4 – IO7 and 20 mA Sink Current on IO0 – IO3
- Unique 48-Bit ID Number in Every Device to Facilitate Network Installation and Management — ID Number Stored Redundantly for Guaranteed 20-Year Data Retention
- 32-Pin SOG or 44-Pin PQFP Package
- Low Operating Current: 16 mA (Typical) at 10 MHz Frequency
1.8 mA (Typical) at 625 kHz Frequency
- Sleep Mode Operation Reduced Current Consumption (9 μ A Typical)
- Maximum Clock Operation of 10 MHz, Over a – 40 to + 85°C Temperature Range
- Either LonBuilder Release 3.x or NodeBuilder[®] Software is Required for Programming
- On-Chip LVI Circuit Enabled Reset Below 4.1 V \pm 300 mV Range



DW SUFFIX
SOG PACKAGE
CASE 1116-0

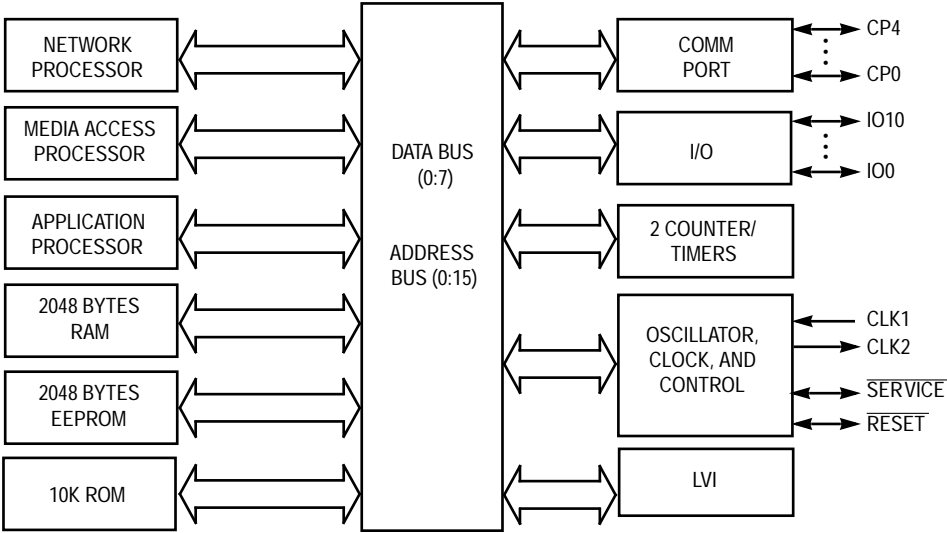


FB SUFFIX
PQFP PACKAGE
CASE 824E-02

ORDERING INFORMATION

MC143120E2DW	SOG Package
MC143120E2FB	PQFP Package

BLOCK DIAGRAM



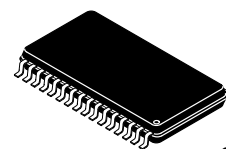
Product Preview

Neuron[®] Chip Distributed Communications and Control Processor

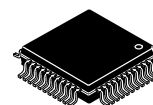
The MC143120FE2 contains the LonTalk[®] protocol in 10K ROM. It differs from the MC143120B1 in that it contains 21 I/O models. Bit Shift, Serial Input, and Serial Output I/O models are not in the ROM firmware, but can be loaded into EEPROM by the LonBuilder[®] tool if needed. In addition, 21 I/O models are integrated with ROM to provide simplified sense and control device interfacing.

- Maximum Clock Rate of 20 MHz Over – 40 to + 85°C
- 5.0 V Operation (4.5 to 5.5 V)
- CMOS Level Input (0.7 V_{DD} and 0.2 V_{DD})
- Three 8-Bit Pipelined Processors for Concurrent Processing of Application Code and Network Packets
- 11-Pin I/O Port Programmable in 21 Models for Fast Application Program Development
- Two 16-Bit Timer/Counters for Measuring and Generating I/O Device Waveforms
- 5-Pin Communications Port that Supports Direct Connect and Network Transceiver Interfaces
- 2048 Bytes of Static RAM for Buffering Network Data and Storing Network Variables
- 2048 Bytes of EEPROM for Flexible Storage of Address and Binding Data
- Programmable Pull-Ups on IO4 – IO7 and 20 mA Sink Current on IO0 – IO3
- Unique 48-Bit ID Number in Every Device to Facilitate Network Installation and Management — ID Number Stored Redundantly for Guaranteed 20-Year Data Retention
- 32-Pin SOG or 44-Pin PQFP Package
- Low Operating Current: 32 mA (Typical) at 20 MHz Frequency
1.8 mA (Typical) at 625 kHz Frequency
- Sleep Mode Operation Reduced Current Consumption (9 µA Typical)
- Either LonBuilder Release 3.X or NodeBuilder[®] Software is Required for Programming — For Downloadable Files for Programming the FE2 Series, See Echelon's Web Site at www.echelon.com/toolbar/up3120p.asp
- On-Chip LVI Circuit Enabled Reset Below 4.1 V ± 300 mV Range

MC143120FE2



DW SUFFIX
SOG PACKAGE
CASE 1116-0

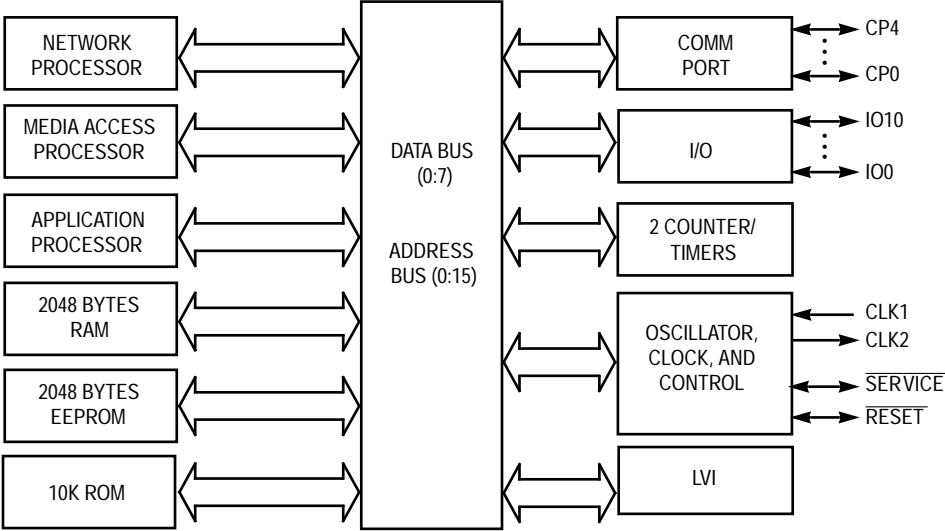


FB SUFFIX
PQFP PACKAGE
CASE 824E-02

ORDERING INFORMATION

MC143120FE2DW	SOG Package
MC143120FE2FB	PQFP Package

BLOCK DIAGRAM



MC143120LE2

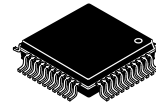
Product Preview

Neuron[®] Chip Distributed Communications and Control Processor

The MC143120LE2 is a communications and control processor which enables the development of interoperable products. It is designed for battery and hand-held applications. It provides systems designers with many features to accelerate development for distributed sense and control applications. Services at every layer of the OSI networking model are implemented in the included LonTalk[®] firmware-based protocol and are easily and optionally invoked. In addition, 21 I/O models are integrated with ROM to provide simplified sense and control device interfacing.

The MC143120LE2 is designed for a maximum clock operation of 5 MHz over a temperature range of -40 to +85°C.

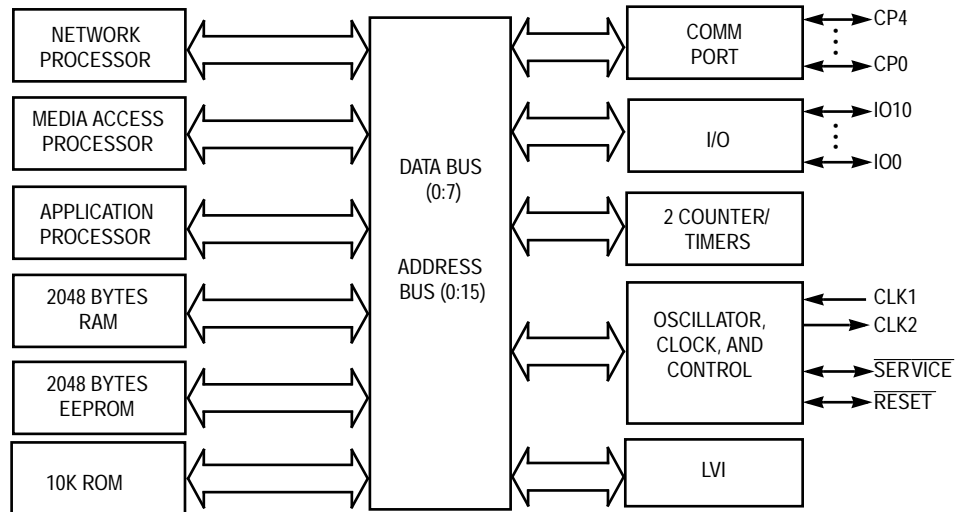
- 3.0 V Operation (2.8 – 3.3 V)
- CMOS Level Inputs (0.7 V_{DD} and 0.2 V_{DD})
- 625 kHz to 5 MHz Operation — Fully Functionally Compatible with MC143120E2
- Three 8-Bit Pipelined Processors for Concurrent Processing of Application Code and Network Packets
- 11-Pin I/O Port Programmable in 21 Models for Fast Application Program Development
- Two 16-Bit Timer/Counters for Measuring and Generating I/O Device Waveforms
- 5-Pin Communications Port That Supports Direct Connect and Network Transceiver Interfaces
- 2048 Bytes of Static RAM for Buffering Network Data and Storing Network Variables
- 2048 Bytes of EEPROM for Flexible Storage of Address and Binding Data
- Programmable Pull-Ups on IO4 – IO7 and 15 mA Sink Current on IO0 – IO3
- Unique 48-Bit ID Number in Every Device to Facilitate Network Installation and Management — ID Number Stored Redundantly for Guaranteed 20-Year Data Retention
- 10 Kbytes ROM
- 44-Pin PQFP Package
- Low Operating Current: 4 mA (Typical) at 5 MHz Frequency
0.8 mA (Typical) at 625 kHz Frequency
- Sleep Mode Operation Reduced Current Consumption (4 µA Typical)
- Low Voltage Inhibit (LVI) Circuit 2.55 V ± 150 mV
- Transceiver Pins Allow Network Connections Without Use of External Relays



FB SUFFIX
PQFP PACKAGE
CASE 824E-02

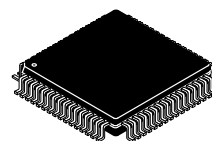
ORDERING INFORMATION
MC143120LE2FB PQFP Package

BLOCK DIAGRAM



MC143150B1

Neuron[®] Chip Distributed Communications and Control Processor

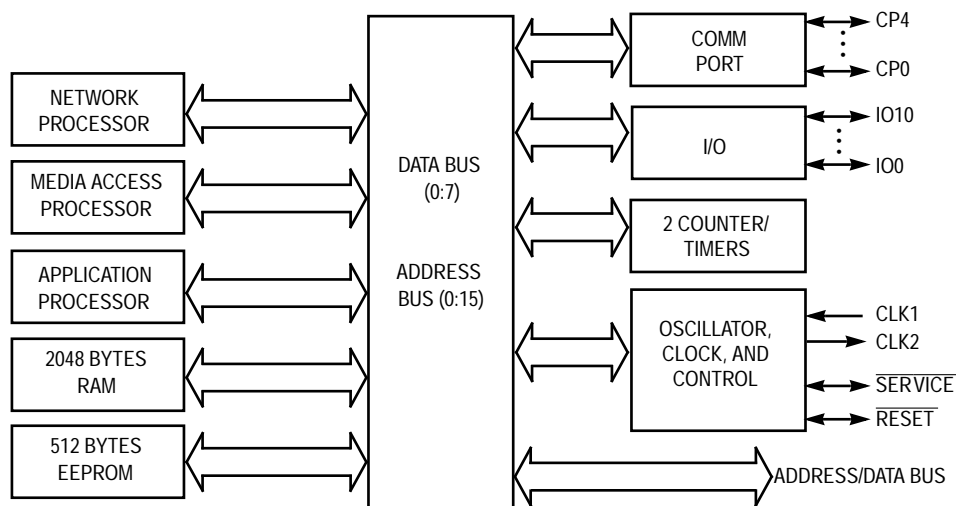


FU SUFFIX
PQFP PACKAGE
CASE 840C-04

ORDERING INFORMATION
MC143150B1FU1 PQFP Package

- Three 8-Bit Pipelined Processors for Concurrent Processing of Application Code and Network Packets
- 11 I/O Pins are Customer Programmable or the 34 Models Available in the LonBuilder[®] Development Tool May be Used
- Two 16-Bit Timer/Counters for Measuring and Generating I/O Device Waveforms
- 5-Pin Communications Port That Supports Direct Connect and Network Transceiver Interfaces
- 2048 Bytes of Static RAM for Buffering Network Data and Storing Network Variables
- 512 Bytes of EEPROM with On-Chip Charge Pump for Flexible Storage of Address and Binding Data
- Programmable Pull-Ups on IO4 – IO7 and 20 mA Sink Current on IO0 – IO3
- Unique 48-Bit ID Number in Every Device to Facilitate Network Installation and Management — ID Number Stored Redundantly for Guaranteed 20-Year Data Retention
- 64-Pin PQFP Package
- Low Operating Current: 16 mA (Typical) at 10 MHz Frequency
4 mA (Typical) at 625 kHz Frequency
- Sleep Mode Operation Reduced Current Consumption (15 μ A Typical)
- Can Address Up to 65,536 Bytes of Memory — 6,144 Bytes of the Address are Mapped Internally
- Supports Slower External Memory (120 ns with No Decode Logic)
- 16,384 Bytes of External Memory are Required for LonTalk[®] Firmware
- Maximum Clock Operation of 10 MHz, Over a – 40 to + 85°C Temperature Range
- Supported by All Releases of LonBuilder Software
- **Motorola Recommends the Use of an External LVI. When Using External Flash Memory, an LVI with Extended Delay Time Should be Used, i.e., Dallas Semiconductor (Part No. DS1233-10)**

BLOCK DIAGRAM



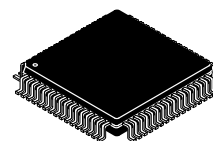
MC143150B2

Neuron[®] Chip Distributed Communications and Control Processor

The MC143150B2 is a communications and control processor which enables the development of interoperable products. It provides systems designers with many features to accelerate product development for distributed sense and control applications. Services at every layer of the OSI networking model are implemented in the IC, including the LonTalk[®] firmware-based protocol, which are easily and optionally invoked. In addition, 34 I/O routines are integrated within hardware to provide simplified sense and control device interfacing.

The MC143150B2 is designed for operation over a temperature range of – 40 to + 85°C.

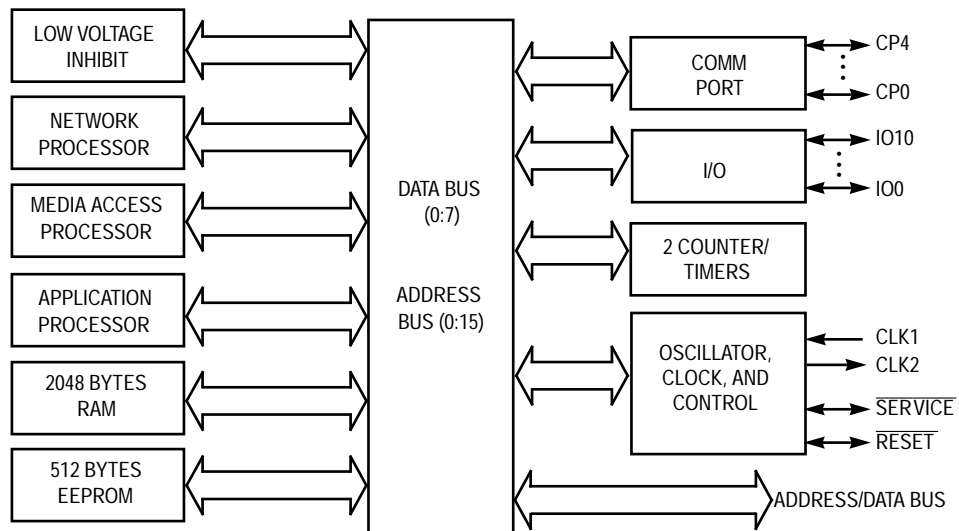
- 10 MHz Crystal Operation — Full Function Compatibility with MC143150B1FU1
- Three 8-Bit Pipelined Processors for Concurrent Processing of Application Code and Network Packets
- 11 I/O Pins are Customer Programmable or the 34 Models Available in the LonBuilder[®] Development Tool May be Used
- Two 16-Bit Timer/Counters for Measuring and Generating I/O Device Waveforms
- 5-Pin Communications Port that Supports Direct-Connect and Network Transceiver Interfaces
- 2048 Bytes of Static RAM for Buffering Network Data and Storing Network Variables
- 512 Bytes of EEPROM with On-Chip Charge Pump for Flexible Storage of Address and Binding Data
- Programmable Pull-Ups on Pins IO4 – IO7 and 20 mA Sink Current on Pins IO0 – IO3
- Unique 48-Bit ID Number in Every Device to Facilitate Network Installation and Management — ID Number Stored Redundantly for Guaranteed 20-Year Data Retention
- Low Operating Current: 16 mA (Typical) @ 10 MHz Frequency
8 mA (Typical) @ 5 MHz Frequency
- Sleep Mode Operation Reduced Current Consumption (15 μ A Typical)
- On-Chip Low Voltage Inhibit (LVI) Circuit with Programmable Sleep Mode On/Off Operation (Trip Point 4.1 ± 0.3 V)
- 0.8 μ m Design Process
- Transceiver Pins Allow Network Connections Without Use of External Relays
- **Motorola Recommends the Use of an External LVI. When Using External Flash Memory, an LVI with Extended Delay Time Should be Used, i.e., Dallas Semiconductor (Part No. DS1233-10)**



FU SUFFIX
PQFP PACKAGE
CASE 840C-04

ORDERING INFORMATION
MC143150B2FU PQFP Package

BLOCK DIAGRAM



Hardware Resources

3

SECTION 3

Neuron CHIP PROCESSOR FAMILY — HARDWARE RESOURCES

The first members of the Neuron Chip processor family are the MC143150 and the MC143120. The MC143150 supports external memory for more complex applications, while the MC143120 is a complete system on a chip. The major hardware blocks of both processors are the same, except where noted; see Table 3-1 and Figure 3-1.

Table 3-1. Comparison of Neuron Chip Processors

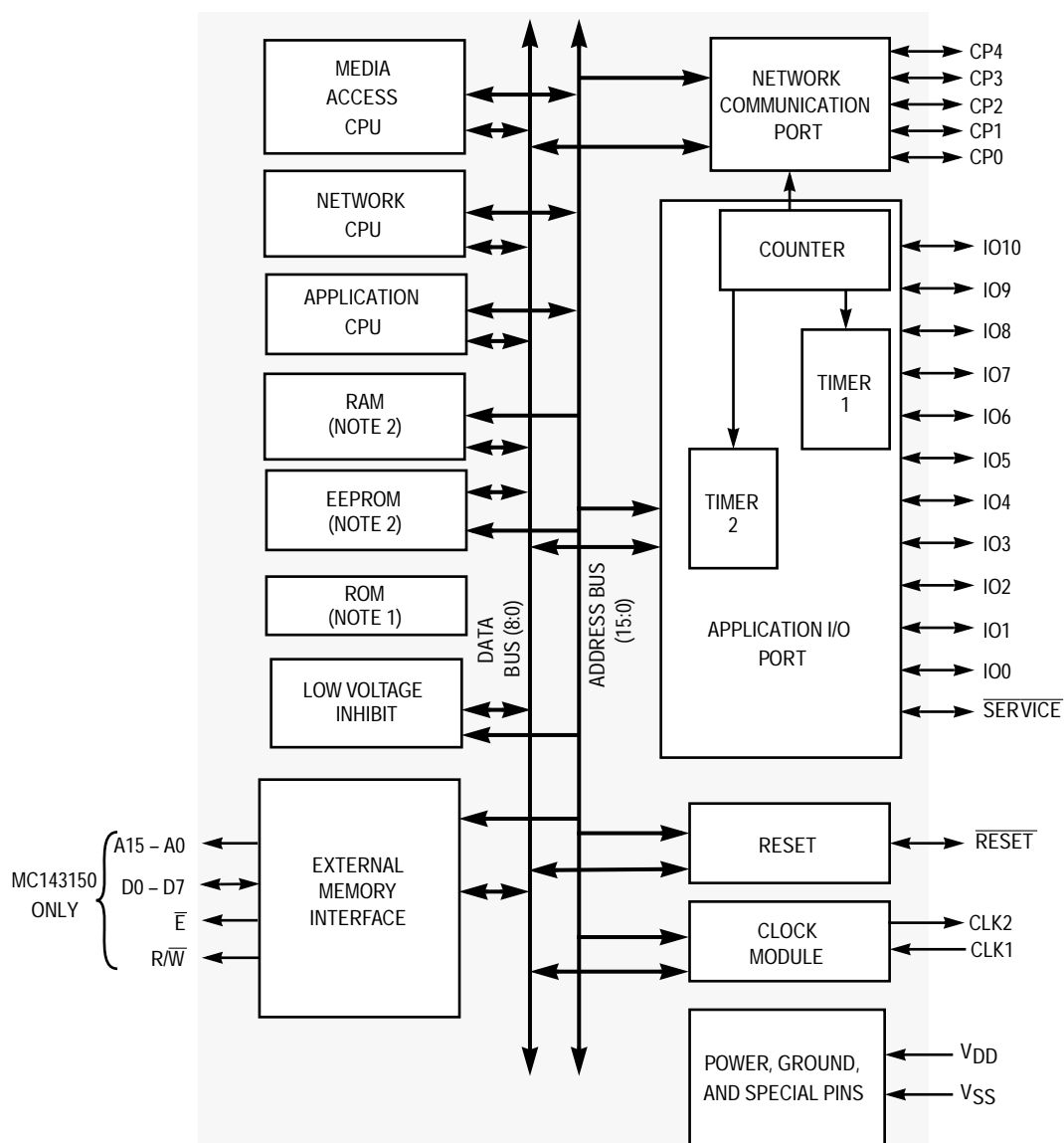
Characteristic	MC143150B1/B2	MC143120B1	MC143120E2/FE2/LE2
Processors	3	3	3
RAM Bytes	2,048	1,024	2,048
ROM Bytes	—	10,240	10,240
EEPROM Bytes	512	512	2,048
16-Bit Timer/Counters	2	2	2
External Memory Interface	Yes	No	No
Package	PQFP	SOG	SOG/QFP
Pins	64	32	32/44

3.1 PROCESSING UNITS

The three processors are dedicated to the following functions by the system firmware (Figure 3-2).

Processor 1 is the MAC layer processor that handles layers 1 and 2 of the 7-layer network protocol stack. This includes driving the communications subsystem hardware as well as executing the collision avoidance algorithm. Processor 1 communicates with Processor 2 using network buffers located in shared memory.

Processor 2 is the Network Processor that implements layers 3 through 6 of the network protocol stack. It handles network variable processing, addressing, transaction processing, authentication, background diagnostics, software timers, network management, and routing functions. Processor 2 uses network buffers in shared memory to communicate with Processor 1, and application buffers to communicate with Processor 3. The buffers are also located in shared memory. Access to them is mediated with hardware semaphores to resolve contention when updating shared data.



NOTES:

1. ROM only on MC143120 series.
2. See Table 1-1.

Figure 3-1. Neuron Chip Block Diagram

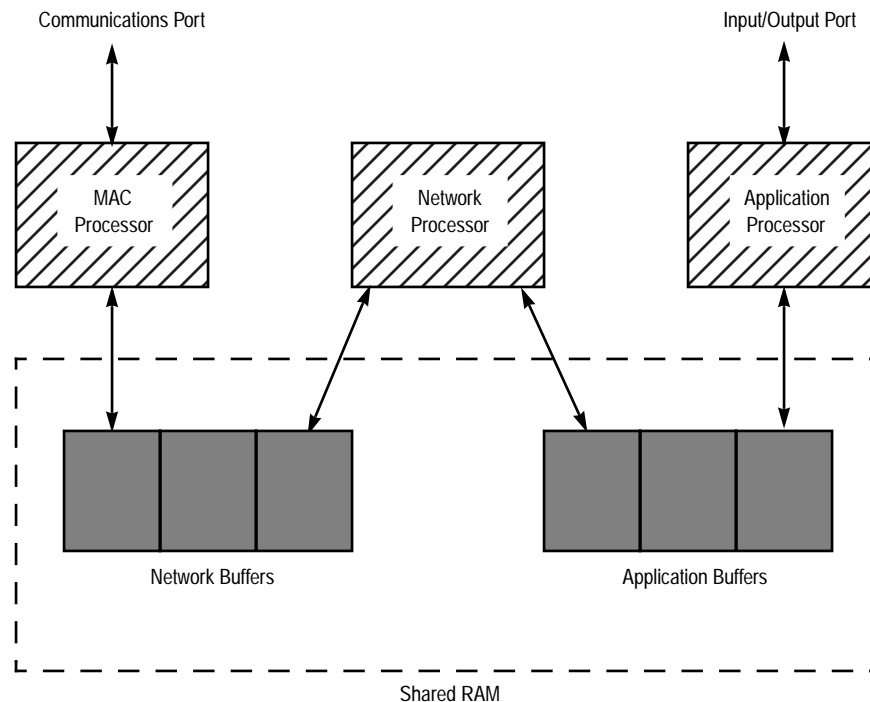


Figure 3-2. Processor Organization Memory Allocation

Processor 3 is the application processor. It executes the code written by the user, together with the operating system services called by user code. The primary programming language used by most applications is Neuron C, a derivative of the ANSI C language optimized and enhanced for LON distributed control applications. The major enhancements are:

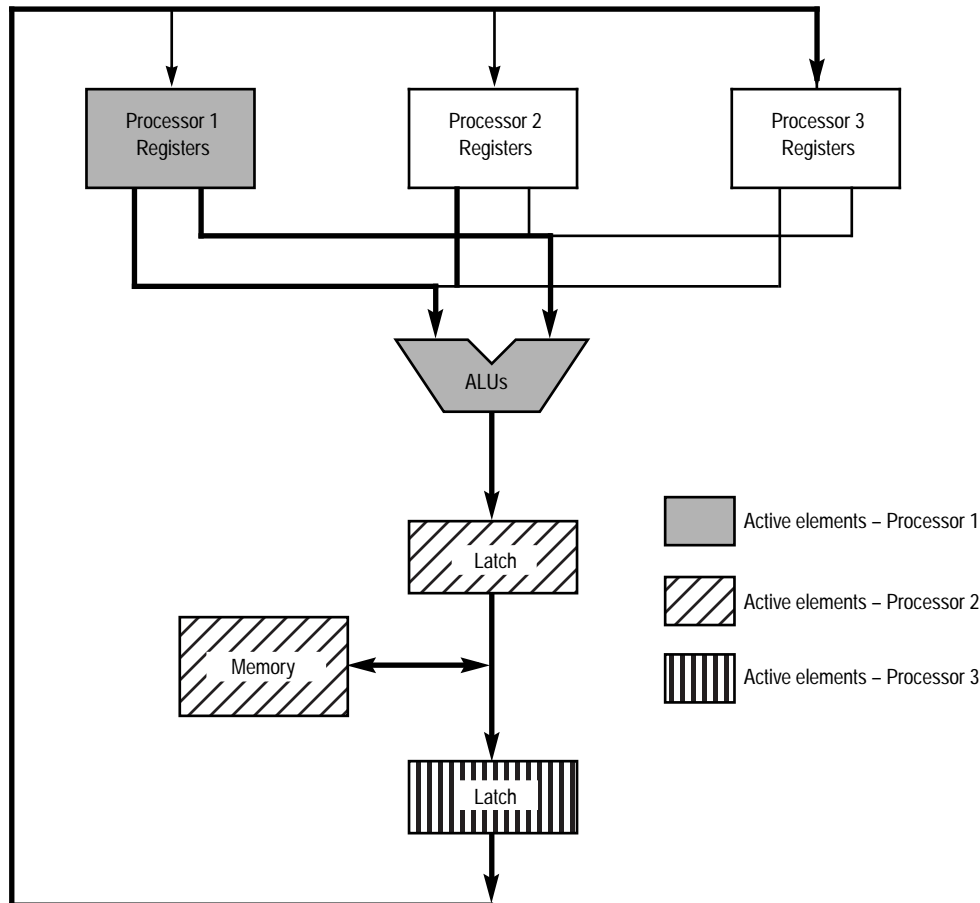
- A built-in multitasking scheduler that allows the programmer to express logically parallel event-driven tasks in a natural way, and to control the priority execution of these tasks.
- A declarative syntax for input/output objects directly mapping into the input/output capabilities of the processor — see Section 7 for details.
- A declarative syntax for network variables, which are Neuron C language objects whose values are automatically propagated over the network whenever values are assigned to them.
- A declarative syntax for millisecond and second timer objects which activate user tasks on expiration.
- A runtime library of function calls to perform event checking, manage input/output activities, send and receive messages across the network, and to control miscellaneous functions of the Neuron Chip.

The support for all these capabilities is part of the LONWORKS firmware, and does not need to be written by the programmer. This allows even complex applications to be written in the application program memory space of the MC143120 processor.

Each of the three identical processors has its own register set (Table 3-2), but all three processors share data and ALUs (address arithmetic and logic units) and memory access circuitry (Figure 3-3). On the MC143150, the internal address, data, and R/\overline{W} lines are reflected on the corresponding external lines when utilized by any of the internal processors. Each CPU *minor* cycle consists of *three system clock* cycles, or phases; each system clock cycle is two input clock cycles. The minor cycles of the three processors are offset from one another by one system clock cycle, so that each processor can access memory and ALUs once during each instrument cycle. Figure 3-3 shows the active elements for each processor during one of the three phases of a minor cycle. Therefore, the system pipelines the three processors, reducing hardware requirements without affecting performance. This allows the execution of three processes in parallel without time-consuming interrupts and context switching.

Table 3-2. Register Set

Mnemonic	Bits	Contents
FLAGS	8	CPU Number, Fast I/O Select, and Carry Bit
IP	16	Next Instruction Pointer
BP	16	Address of 256-Byte Base Page
DSP	8	Data Stack Pointer Within Base Page
RSP	8	Return Stack Pointer Within Base Page
TOS	8	Top of Data Stack, ALU Input

**Figure 3-3. Processor/Memory Activity During One of the Three System Clock Cycles of a Minor Cycle**

The architecture is stack-oriented; one 8-bit wide stack is used for data references, and the ALU operates on the TOS (Top of Stack) register and the next entry in the data stack which is in RAM. A second stack stores the return addresses for CALL instructions, and may also be used for temporary data storage. This zero-address architecture leads to very compact code. Tables 3-3, 3-4, and 3-5 outline the instruction set.

Figure 3-4 shows the layout of a base page, which may be up to 256 bytes long. Each of the three processors uses a different base page, whose address is given by the contents of the BP register of that processor. The top of the data stack is in the 8-bit TOS register, and the next element in the data stack is at the location within the base page at the offset given by the contents of the DSP register. The data stack grows from low memory towards high memory, and the return stack grows from high memory towards low memory.

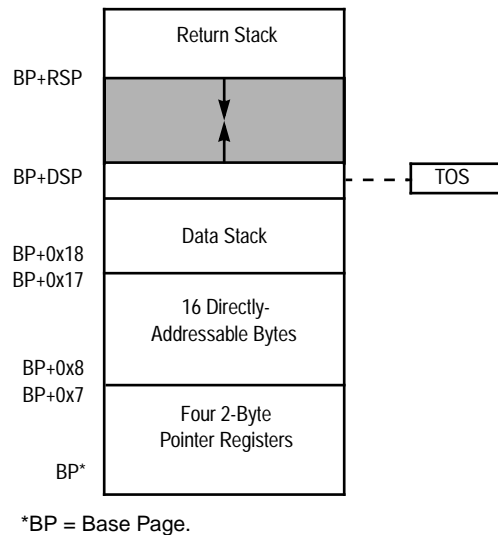


Figure 3-4. Base Page Memory Layout

A processor instruction cycle is three system clock cycles, or six input clock (CLK1) cycles. Most instructions take between one and seven processor instruction cycles. At a maximum input clock rate of 10 MHz, instruction times are between 0.6 μ s and 4.2 μ s. The formula for instruction time is:

$$\text{<Instruction Time> } (\mu\text{s}) = \text{<\# Cycles>} \times 6 / \text{<Input Clock>} (\text{MHz})$$

Tables 3-3, 3-4, and 3-5 list the processor instructions and their timings. This is provided for purposes of calculating response times to events on the I/O pins. All programming of the Neuron Chip is done with Neuron C using a LonBuilder development system.

Table 3-3. Program Control Instruction Timings

Mnemonic	Cycles	Description
NOP	1	No operation
SBR	1	Short branch
BR/BRC/BRNC	2	Branch, branch on (not) carry
SBRZ/SBRNZ	3	Short branch on (not) zero
BRF	4	Branch far
BRZ/BRNZ	4	Branch on (not) zero
RET	4	Return from subroutine
BRNEQ	4/6	Branch if not equal (taken/not taken)
DBRNZ	5	Decrement and branch if not zero
CALLR	5	Call subroutine relative
CALL	6	Call subroutine
CALLF	7	Call subroutine far

Table 3-4. Memory/Stack Instruction Timings

Mnemonic	Cycles	Addressing Mode
PUSH TOS	3	TOS register
PUSH/POP cpu reg	4	Any CPU register
PUSH/POP !D	4	Base page plus displacement (8 – 23)
PUSH/POP !TOS	4	Base page plus contents of TOS
PUSH [RSP]	4	Data on top of return stack
PUSHS/PUSH #literal	4	3- or 8-bit literal value
PUSHPOP	5	Pop from return stack, push to data stack
POPPUSH	5	Pop from data stack, push to return stack
LDBP address	5	Load base pointer register
PUSH/POP [DSP][D]	5	Contents of DSP minus displacement (1 – 8)
PUSHD #literal	6	16-bit literal value
POPD/PUSHD [PTR]	6	16-bit indirect through base page pointer (0 – 3)
PUSH/POP [PTR][TOS]	6	Indirect through base page pointer plus TOS
PUSH/POP [PTR][D]	7	Indirect through base page pointer plus displacement
PUSH/POP absolute	7	Push memory data to data stack
IN/OUT	7 + 4n	Transfer n bytes to I/O

Table 3-5. ALU Instruction Timings

Mnemonic	Cycles	Operation
INC/DEC/NOT	2	Increment/decrement/negate
TOSROL/RORC	2	Rotate left/right TOS through carry
SHL/SHR	2	Logical left/right shift TOS
SHLA/SHRA	2	Arithmetic left/right shift TOS
DROP NEXT	2	Drop next element from data stack
DROP [RSP]	2	Drop top of return stack
ADD/AND/OR/XOR #literal	3	Operate with literal on TOS
DROP TOS	3	Drop top of data stack
ALLOC #literal	3	Move data stack pointer
ADD/AND/OR/XOR	4	Operate with next element on TOS
ADC/SBC/SUB/XCH	4	Operate with next element on TOS
INC [PTR]	6	Increment base page pointer (0 – 3)
DEALLOC_R #literal	6	Move data stack pointer and return
(ADD/AND/OR/XOR) – R	7	Operate with next element on TOS and return

3.2 MEMORY

3.2.1 Memory Allocation Overview

3.2.1.1 MC143150 MEMORY ALLOCATION (SEE FIGURE 3-5).

- 512 bytes of in-circuit programmable EEPROM that store:
 - Network configuration and addressing information.
 - Unique 48-bit LON identification code — written at the factory.
 - User-written application code and read-mostly data.
- 2048 bytes of static RAM that store:
 - Stack segment, application, and system data.
 - LON protocol network buffers and application buffer.
- Up to 65,536 bytes of memory address space — any mix of ROM, EPROM, EEPROM, or RAM. The processor can access 59,392 bytes of this memory via the external memory interface; 6144 bytes of the address space are mapped internally.
- 16,384 bytes of external memory are required to store:
 - The LONWORKS operating system, including the system firmware executed by the MAC and Network processors, and the executive supporting the application program.
- The rest of external memory is available for:
 - User-written application code.
 - Additional application read/write data.
 - Additional network buffers and application buffer.

3.2.1.2 MC143120 MEMORY ALLOCATION (SEE FIGURES 3-6 AND 3-7).

- 512 bytes of in-circuit programmable EEPROM (2048 bytes for MC143120E2/FE2/LE2) that store:
 - Network configuration and addressing information.
 - Unique 48-bit LON identification code — written at the factory.
 - User-written application code and read-mostly data.
- 1024 bytes of static RAM (2048 bytes for MC143120E2/FE2/LE2) that store:
 - Stack segment, application, and system data.
 - LON protocol network buffers and application buffer.
- 10,240 bytes of masked ROM that store:
 - LONWORKS firmware executed by the MAC and Network processors.
 - Operating system supporting the application program.

3.2.2 EEPROM

All versions of the Neuron Chip have internal EEPROM containing:

- Network configure and addressing information.
- Unique 48-bit Neuron Chip identification code.
- Optional user-written application code and data tables.

All but 8 bytes of the EEPROM can be written under program control using an on-chip charge pump to generate the required programming voltage. The charge pump operation is transparent to the user. The remaining 8 bytes are written during manufacture, and contain a unique 48-bit identifier for each part, plus 16 bits for the manufacturer's device code. Erase and write time are each 10 ms per byte. The EEPROM may be written up to 10,000 times with no data loss (– 40 to + 85°C). For both the MC143120 and the MC143150, the EEPROM stores the installation-specific information such as network addresses and communications parameters. For the MC143120, EEPROM memory also stores the application program

generated by the LonBuilder Developer's Workbench software. The application code for the MC143150 may be stored either on-chip in the EEPROM memory, or off-chip in external memory, or both.

For all write operations to the internal EEPROM, the Neuron Chip firmware automatically compares the value in the EEPROM location to the value to be written. If the two are the same, the write operation is not performed. This prevents unnecessary write cycles to the EEPROM, and reduces the average EEPROM write cycle latency.

Note that when the Neuron Chip is not within the specified power supply voltage range, a pending or on-going EEPROM write is not guaranteed. On the MC143120E2, MC143120FE2, MC143120LE2, and MC143150B2, there is a built-in LVI that holds the chip in reset below a certain voltage. This helps prevent EEPROM data corruption, although additional external protection may be appropriate, such as an MC33164. See Section 4.6.3 for information on an LVI circuit.

The EEPROM of the MC143150 can usually be reset to its factory defaults (10 MHz, differential, 1.25 Mbps) in the event of a fault by executing the EEBLANK program, which is supplied with the LonBuilder Developer's Workbench software (EEBLANK.NRI).

The `set_eeprom_lock()` function can also be used for additional protection against accidental EEPROM data corruption. This function allows the application program to set the state of the lock on the checksummed portion of the EEPROM. Refer to the *Neuron C Programmer's Guide* for more information.

The internal EEPROM of a Neuron Chip contains a fixed amount of overhead, a network image (configuration), user code, and user data. The following table shows the maximum amount of EEPROM space available for user code and user data assuming a minimally-sized network image. Also shown is the minimum segment size for user data. Note that constant data is assumed to be part of the code space.

Device	EEPROM Use (Bytes)	Segment Size (Bytes)
MC143120	418 code/255 data	1
MC143120E2/FE2/LE2	1949 code or data	8
MC143150B1/B2	413 code or data	2

The segment size is the smallest unit of EEPROM that is allocated for user data, and therefore provides a measure of the amount of fragmentation possible. For example, if there is a single 1-byte user variable used in an MC143120E2/FE2/LE2, then 8 EEPROM bytes are allocated for variable space. (This depends on the firmware version.)

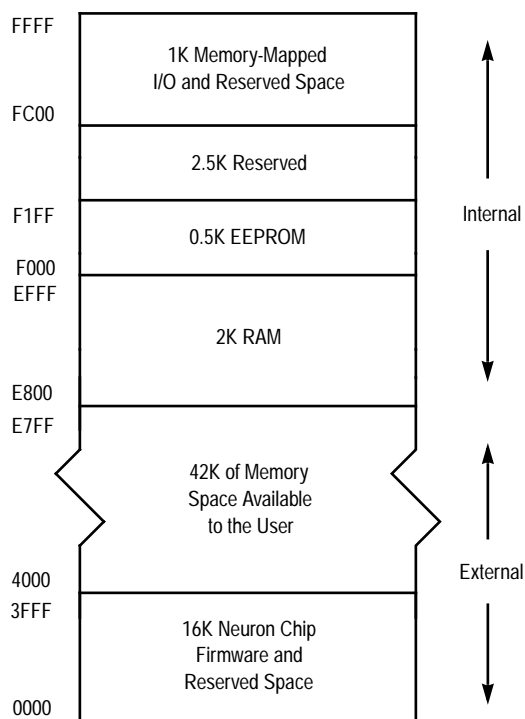


Figure 3-5. MC143150 Processor Memory Map

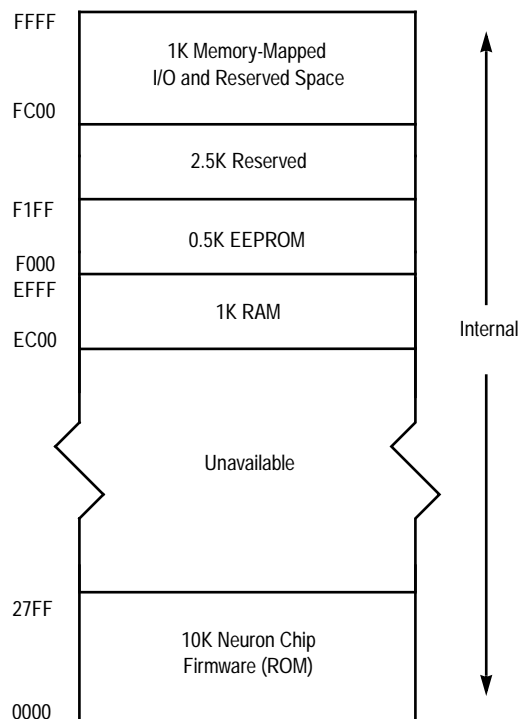


Figure 3-6. MC143120B1 Processor Memory Map

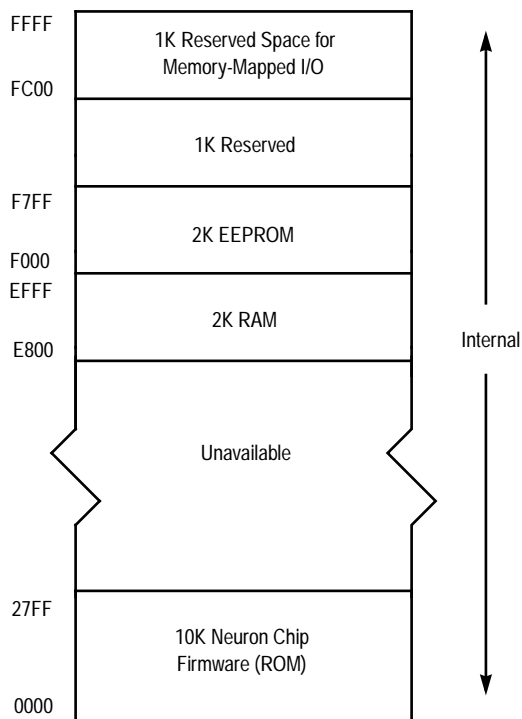


Figure 3-7. MC143120E2/FE2/LE2 Processor Memory Map

3.2.3 Static RAM

The MC143120B1 has 1024 bytes of static RAM. The MC143150 and MC143120E2/FE2/LE2 chips contain 2048 bytes of static RAM.

The RAM is used to store:

- Stack segment, application, and system data
- LonTalk protocol network buffers and application buffers

The RAM state is retained as long as power is applied to the device, even in “sleep” mode. After reset, releasing the Neuron initialization sequence will clear the RAM (see Section 4.6.4).

3.2.4 Preprogrammed ROM

The MC143120 contains 10,240 bytes of pre-programmed ROM. This memory contains the LONWORKS firmware, including the LonTalk protocol code, real time task scheduler, and application function libraries. The MC143150 accesses external memory for all of these. The object code is supplied with the LonBuilder Developer's Workbench.

3.2.5 External Memory (MC143150 Only)

This interface supports up to 42K bytes of external memory space for additional user program and data. The total address space is 64K bytes. However, 6K of address space is reserved for on-chip, leaving 58K of external address space. Of this space, 16K is used by the Neuron Chip firmware, LonBuilder development debugger, and reserved space. The external memory space can be populated with RAM, ROM, PROM, EPROM, or EEPROM in 256-byte increments. The memory map for the MC143150 is shown in Figure 3-5. The bus has 8 bidirectional data lines and 16 address lines driven by the processor. Two interface lines (R/\overline{W} and \overline{E}) are used for external memory access (see Figure 3-1). Refer to Section 6.2.4 for the required memory access times. If the input clock is scaled down, slower memory can be used. The suggested input clock rates are 10 MHz, 5 MHz, 2.5 MHz, 1.25 MHz, and 625 kHz. The Enable Clock (\overline{E}) runs at the system clock rate, which is one-half the input clock rate. The Enable Clock is low whenever data is being transferred between the Neuron Chip and external memory. All memory, both internal and external, may be accessed by any of the three processors at the appropriate phase of the instruction cycle. Since the instruction cycles of the three processors are offset by one-third of a cycle, with respect to each other, the memory bus is in use by only one processor at a time.

Appendix C, *External Memory Interfacing*, shows diagrams of interfacing the MC143150 to different types of memory. A minimum hardware configuration would use one external ROM (EPROM), containing both the protocol code and user application code (see Figure C-1). This configuration would **not** allow the system engineer to change the *application code* after installation. *Binding* and *address* information, however, could be altered because this information resides in the internal 512 bytes of EEPROM. When developing nodes that must be rewritten over the network with application code, external EEPROM is required if the user code will not fit in the 512 bytes of internal EEPROM. Refer to Chapter 7 in the *Neuron C Programmer's Guide* for ideas to reduce code size. When designing a node using a slower crystal clock rate where possible, will significantly reduce memory speed and cost; in addition to reducing power consumption. The pins used for external memory interfacing are listed in Table 3-6. The \overline{E} clock signal is used to generate read (or write) signals to external memory. The A15 (address line 15) or the programmable array logic (PAL) decoded signal gated with R/\overline{W} can be used to generate read signals to external memory.

Table 3-6. External Memory Interface Pins

Pin Designation	Direction	Function
A0 – A15	Output	Address Pins
D0 – D7	Input/Output	Data Pins
\overline{E}	Output	Enable Clock
R/W	Output	Read/Write Select Low

Because one of the internal processors is always using the address bus, sharing the address and data bus (and memory) with another MPU address and data bus, can only be accomplished with the use of external three-state bus drivers on the 16 address lines. This method is **not** recommended or emulated with the LonBuilder Developer's Workbench. The preferred method of interfacing the Neuron Chip to another MPU is through the 11 I/O pins. There are parallel modes and serial modes for this purpose which are easily implemented using the Neuron C programming language. Memory mapped I/O is supported in the LonBuilder environment.

3.2.6 Memory Integrity Using Checksums

To ensure the integrity of the Neuron Chip's memory, the system maintains a number of checksums. Each checksum is a single byte and is the two's complement of the sum of all bytes it covers. These checksums are verified during reset processing and also on a continual basis via a background diagnostic process. There are three main checksums used to verify the integrity of the Neuron Chip's memory:

- Configuration image checksum
- Application image checksum
- System image checksum (off-chip system image only)

The configuration image checksum covers the network configuration information and communication parameters residing in on-chip EEPROM. The default behavior is that a configuration checksum error causes the node to go to the unconfigured state. Refer to Table 3-7 for other options.

The application image checksum covers the application code in both on-chip EEPROM and any application code in off-chip EEPROM, NVRAM, or flash memory. This checksum can optionally be extended to cover any application code in off-chip ROM as well. The default behavior is that an application checksum error causes the node to go to the applicationless state. Application read/write data residing in EEPROM, NVRAM, or flash are not checksummed. Refer to Table 3-7 for other options.

The application checksum covers all checksummed memory, including:

- a. The system image (optional, 3150 V6 or later firmware only)
- b. Any off-chip ROM code (optional, 3150 V6 or later firmware only)
- c. Any off-chip flash, EEPROM, or NVRAM code
- d. Any off-chip RAM code
- e. Configuration image
- f. All on-chip EEPROM code

In the 3150 V6 or later firmware, each of (a) through (e) have their own checksum, so that checksum errors can be further isolated. Prior to the MC143150 V6 firmware, (c) and (d) were checksummed, but did not contain their own checksum. An unconfigured or configured node continually checks its application checksum in the background at the rate of 1 byte per iteration through the network processor's main loop (3 bytes per millisecond when running at 10 MHz with no network activity). Any detected checksum error results in a reset, but no state change.

The system image checksum covers the system image. It is only available when the system image resides in off-chip memory and its use is optional. A system image checksum error always forces the node to the applicationless state.

No checksum is computed if the node is in the applicationless state.

The checksums are all verified during reset processing by the network processor and as part of the background diagnostic process. The background diagnostic process simply causes the node to reset when an error is detected; **no state change occurs**. It is assumed that any persistent error will be found by the reset processing.

Upon detecting a checksum error, the reset process will force the appropriate state and log an error in the error log. For the MC143150 running version 6 firmware, a checksum must fail twice during reset processing, in order for it to be deemed bad.

Reboot and Integrity Options Word

An MC143150 running version 6 or greater firmware, has a number of options to select from regarding actions taken following a checksum error or other memory related fatal error. The 16-bit word resides in a system image and is defined as part of the node's export options.

The recovery process relies on the fact that the initial on-chip EEPROM image for the application, configuration, and communication parameter data reside in the off-chip system image. During initial power up, the system image data is copied (booted) to on-chip EEPROM. The recovery process, thus recopies or reboots the suspect areas as dictated by the error and the recovery options. **Any changes made to the on-chip EEPROM (e.g., a network application load or network manager initiated reconfiguration) after the initial boot are lost in the recovery process.** The recovery action is defined by setting a combination of bits as defined by the following bit masks (Table 3-7).

Table 3-7. Recovery Action Bit Masks

Recovery Word	Description
0x0001	Reboot application if application fatal error.
0x0002	Always reboot application on reset (see Note).
0x0004	Reboot configuration if configuration checksum fails.
0x0008	Reboot configuration on an application fatal error.
0x0010	Always reboot configuration on reset.
0x0020	Reboot communication parameters if configuration checksum fails.
0x0040	Reboot communication parameters if type or rate mismatch.
0x0080	Always reboot communication parameters on reset.
0x0100	Reboot EEPROM variables when rebooting application.
0x0200	Go applicationless if an application fatal error occurs (prohibits network loading, used in conjunction with 0x0001). Application fatal errors are defined below (see Note).
0x0400	Checksum all code, including system image.

NOTE: Application exported with these options can not be loaded over the network.

In the above options, “configuration” excludes the communication parameters since their recovery is governed separately. Also, fatal application errors refer to application image checksum errors, memory allocation failures, and memory map failures. Refer to the section on Programming 3150 Chip Memory in the *LonBuilder User's Guide* (v3.0).

The configuration will be rebooted independently of the application, only if all the configuration table sizes match between EEPROM and ROM. This avoids the situation where a new application with different table sizes is loaded over the network and a reboot of the configuration corrupts the program.

When an EEPROM recovery occurs due to a checksum failure or other error, the event will be logged in the Neuron Chip's error table. A test command will show *EEPROM recovery occurred* as the last error logged.

Refer to Appendix B, *Checksum Recalculate*, on how to recalculate the checksum.

Reset Processing

During reset processing, the configuration checksum is checked first. If bad, and no configuration recovery options are set (MC143150 V6 only), a configuration checksum error is logged, the checksum repaired, and the node state changed to unconfigured. If configuration recovery is set, the configuration is recovered.

Next, the application checksum is checked. If bad, and the checksum error is in (a), then a system image checksum error is logged and the node state is changed to applicationless.

If the application checksum is bad, and no application recovery options are set (MC143150 V6 only), an application checksum error is logged and the node state is changed to applicationless.

If the application checksum is bad and application recovery is set and the boot application does not contain references to (b), (c), or (d) below; or there are no checksum errors in (b), (c), or (d); then the application is recovered. Otherwise, an application checksum error is logged and the node goes applicationless.

Location of Checksums

The location of each checksum is as follows:

- a. 0x3ffe
- b. First byte of area
- c. Second byte of area
- d. Third byte of area
- e. Last byte of configuration area
- f. Last byte of on-chip checksummed memory (precedes on-chip EEPROM data)

NOTE: The checksum locations listed above may change in the future.

Signatures

Each of (b), (c), and (d) has a 2-byte cyclic redundancy check (CRC) called the signature, which resides in the first 2 bytes of the area or immediately following the area checksum, if the image supports it. Basically, signatures are stored in the area and in the memory map. Mismatches between the area signature and memory map copy of the signature results in the node going applicationless. This mechanism prevents a partial application load over the network which is incompatible with any unloaded code (such as code in ROM).

3.3 INPUT/OUTPUT

3.3.1 Eleven Bidirectional I/O Pins

These pins are usable in several different configurations to provide flexible interfacing to external hardware and access to the internal timer/counters. The level of output pins may be read back by the application processor. See Section 6 for detailed electrical characteristics.

Pins IO4 – IO7 have programmable pull-up current sources. They are enabled or disabled with a compiler directive (see *Neuron C Programmer's Guide*). Pins IO0 – IO3 have high current sink capability (20 mA @ 0.8 V). The others have the standard sink capability (1.4 mA @ 0.4 V). All pins (IO0 – IO10) have hysteresis. Pins IO0 – IO7 also have low level detect latches.

3.3.2 Two 16-Bit Timer/Counters

The timer/counters are implemented as a load register writable by the processor, a 16-bit counter, and a latch readable by the processor. The 16-bit registers are accessed 1 byte at a time. Both the MC143150 and MC143120 have one timer/counter whose input is selectable among pins IO4 – IO7, and whose output is pin IO0, and a second timer/counter with input from pin IO4 and output to pin IO1 (Figure 3-8). Note that no I/O pins are dedicated to timer/counter functions. If, for example, Timer/Counter 1 is used for input signals only, then IO0 is available for other input or output functions. Timer/counter clock and enable inputs may be from external pins, or from scaled clocks derived from the system clock; the clock rates of the two timer/counters are independent of each other. External clock actions occur optionally on the rising edge, the falling edge, or both rising and falling edges of the input.

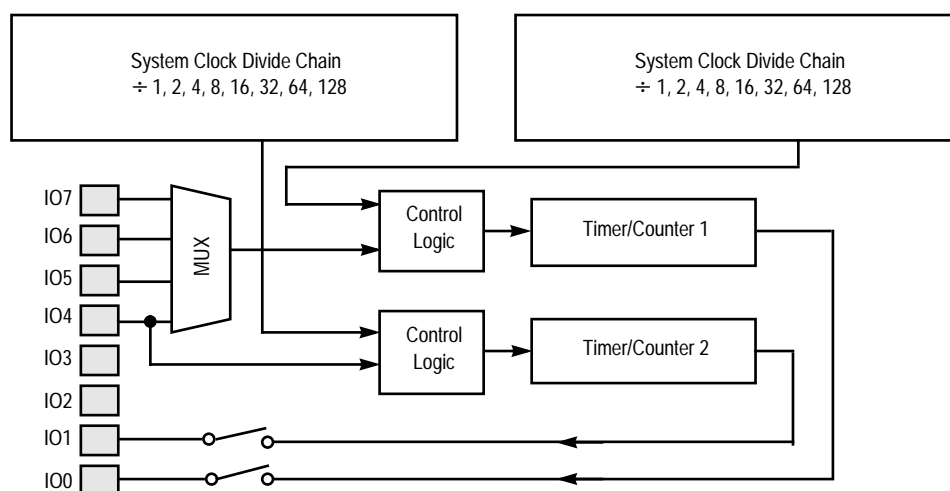


Figure 3-8. Timer/Counter Circuits

Network Communications **4**

SECTION 4

COMMUNICATIONS, CLOCKING, RESET, AND SERVICE

4.1 COMMUNICATIONS

The Neuron Chip can support a wide variety of communications media. The most common are twisted-pair and powerline networks. Other Neuron Chip-based media applications include: radio frequency (RF), infrared (IR), fiber optic, and coaxial (i.e., cable).

Numerous companies offer transceivers, including Echelon (develops and manufactures various types of twisted-pair and powerline transceivers).

Contact either Motorola or Echelon for additional information on sources of transceivers. Table 4-1 below lists common transceivers and the data rates at which they operate.

Table 4-1. Transceiver Types

Transceiver Type	Data Rate
EIA-485	300 bps to 1.25 Mbps
Twisted-Pair Free or Bus Topology (Link Power Optional)	78 kbps
Twisted-Pair with Transformer	78 kbps
Twisted-Pair with Transformer	1.25 Mbps
Powerline	2 kbps
Powerline	5 kbps
Powerline	10 kbps
RF (300 MHz)	1200 bps
RF (450 MHz)	4800 bps
RF (900 MHz)	39 kbps
IR	78 kbps
Fiber Optic	1.25 Mbps
Coaxial	1.25 Mbps

The LONMARK Association, founded to promote interoperability among various companies' LONWORKS technology-based products, has developed a set of recommendations for a number of popular transceivers and procedures for adopting additional transceivers. Usage of LONMARK recommended transceivers fulfills the requirements for physical-layer compliance with the interoperability guidelines.

The benefits of interoperability include:

- A wide selection of products that work together.
- Introduction of development tools to support popular transceivers.
- Increased availability of routers, bridges, and gateways.
- High usage volumes that will drive prices down.

- Thorough evaluation of transceivers by a wide range of companies.
- Elimination of the need for recertification of transceivers.

For more information about interoperability and to request a copy of the LONMARK Interoperability Guidelines, contact the LONMARK Association's web site at <http://www.LONMARK.org>.

4.2 COMMUNICATIONS PORT

The Neuron Chip has a very versatile communications port. It consists of five pins that can be configured to interface to a wide variety of media interfaces (network transceivers) and operate over a wide range of data rates. The communications port can be configured to operate in one of three modes: single-ended, differential, or special-purpose mode. See Table 4-2 and Figure 4-1.

Table 4-2. Communications Port Pin Characteristics

Pin	Drive Current (mA)	Differential	Single-Ended	Special-Purpose Mode
CP0	1.4	Rx+ (in)	Rx (in)	Rx (in)
CP1	1.4	Rx- (in)	Tx (out)	Tx (out)
CP2	40	Tx+ (out)	Tx Enable (out)	Bit Clock (out)
CP3	40	Tx- (out)	Sleep (out)	Sleep (out) or Wake Up (in)
CP4	1.4	CDet (in)	CDet (in)	Frame Clock (out)

Rx — Receiver Tx — Transmitter CDet — Collision Detect

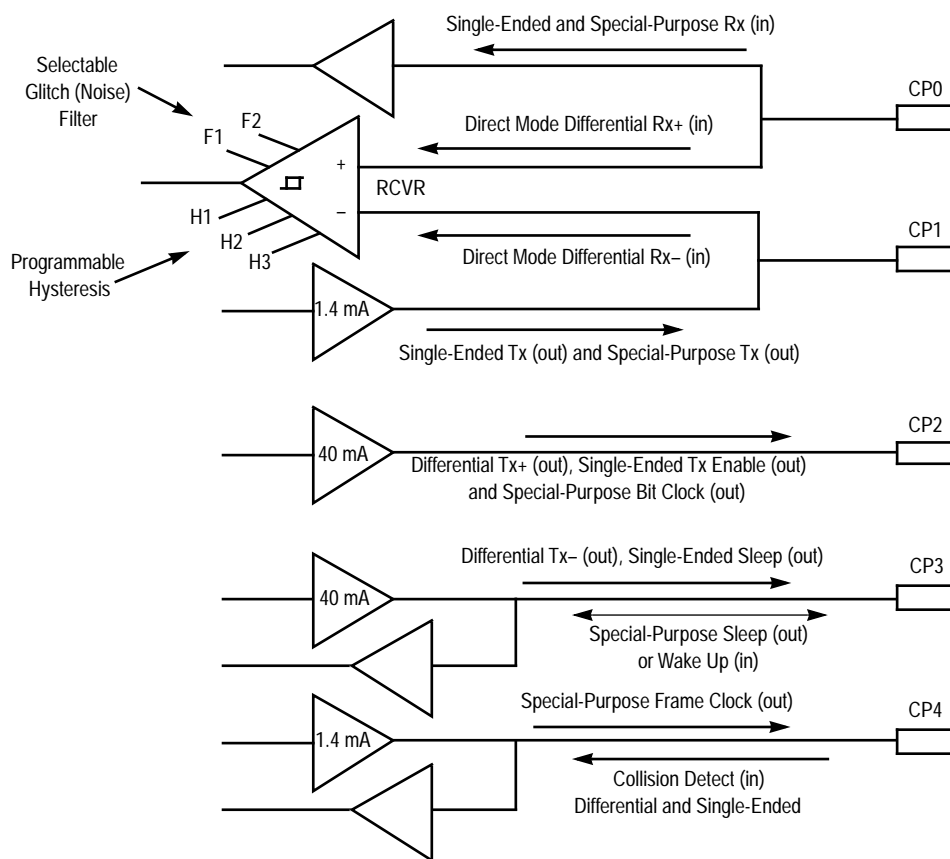


Figure 4-1. Internal Transceiver Block Diagram

4.2.1 Single-Ended Mode

The single-ended mode is most commonly used with external active transceivers interfacing to media such as RF, IR, fiber optic, and coaxial cable. Figure 4-2 shows the communications port configuration for the single-ended mode of operation. Data communication occurs via the single-ended (with respect to V_{SS}) input and output buffers on pins CP0 and CP1. Single-ended and differential modes use Differential Manchester encoding, which is a widely used and reliable format for transmitting data over various media. The available network bit rates for single-ended and differential modes are given in Table 4-3, as a function of the Neuron Chip input clock rate.

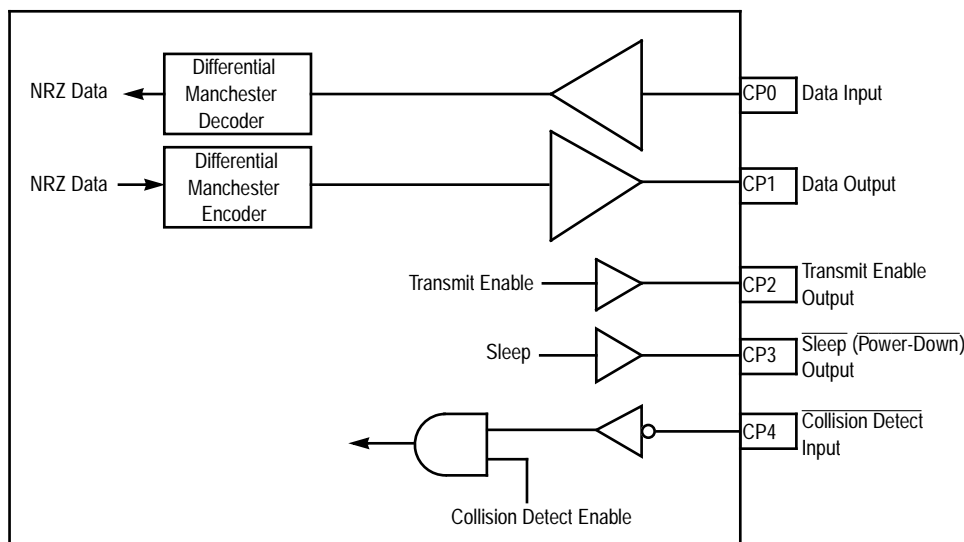


Figure 4-2. Single-Ended Mode Configuration

Table 4-3. Single-Ended and Differential Network Data Rates

Network Bit Rate (kbps)	Minimum Input Clock (MHz)	Maximum Input Clock (MHz)		
2500	20	20	10	5
1250	10	20	10	5
625	5	20	10	5
312.5	2.5	20	10	5
156.3	1.25	20	10	5
78.1	0.625	20	10	5
39.1	0.625	20	10	5
19.5	0.625	20	10	5
9.8	0.625	20	10	5
4.9	0.625	N/A	N/A	5
2.4	0.625	N/A	N/A	2.5
1.2	0.625	N/A	N/A	1.25
0.6	0.625	N/A	N/A	0.625

In single-ended mode, the communications port encodes transmitted data and decodes received data using Differential Manchester coding (also known as bi-phase space coding). This scheme provides a transition at the beginning of every bit period for the purpose of synchronizing the receiver clock. The 0/1 data is indicated by the presence or absence of a second transition halfway between clock transitions. A

mid-cell transition indicates a 0. Lack of a mid-cell transition indicates a 1. Differential Manchester coding is polarity-insensitive. Thus, reversal of polarity in the communication link will not affect data reception. Figure 4-3 shows a typical packet, where T is the bit period, equal to $1/(\text{bit rate})$. Note that clock transitions occur at the beginning of a bit period, and therefore, the last valid bit in the packet does not have a trailing clock edge.

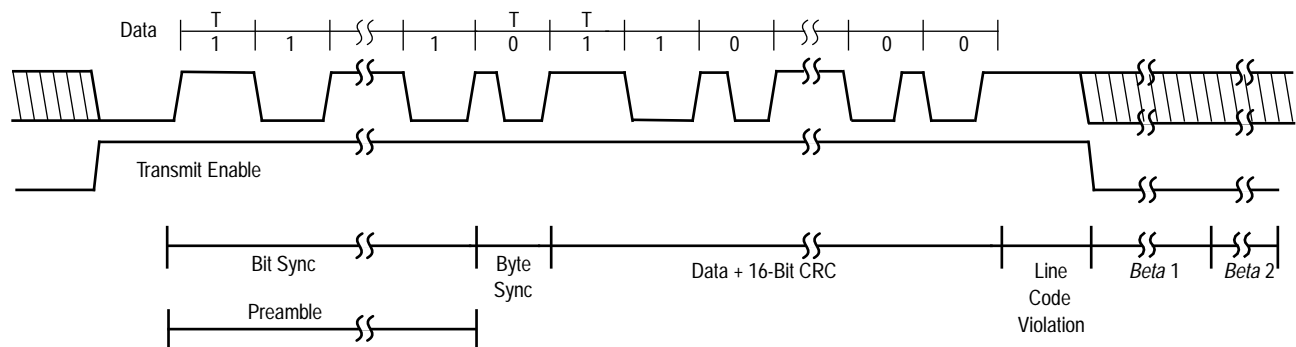


Figure 4-3. Single-Ended Mode Data Format

Before beginning to transmit the packet, the transmitter Neuron Chip initializes the output data pin to start low. It then asserts the Transmit Enable pin (CP2); this ensures that the first transition in the packet is from low to high. This first transition occurs within 1 bit time of asserting Transmit Enable, and marks the beginning of the packet. **Note that Transmit Enable is actively driven at all times in single-ended mode. In single-ended mode, the 1.4 mA driver is connected to CP1 and it is not high impedance when receiving packets.**

The transmitter transmits a *preamble* at the beginning of a packet to allow the other nodes to synchronize their receiver clocks. The preamble consists of a series of Differential Manchester 1s. Its duration is at least 6 bits long (minimum 125 μs ; refer to Section A.6.1), and is selectable by the user.

The preamble ends with a single byte-sync bit, which marks the start of byte boundaries at the bit following. The byte-sync bit is a Differential Manchester 0.

The Neuron Chip terminates the packet by forcing a Differential Manchester code violation. After sending the last CRC bit, it holds the data output transitionless for 2.5 bit times. The Transmit Enable pin on CP2 is then driven low, indicating the end of transmission. For a Neuron Chip, the time to format and begin to send a 12-byte message is from 2.8 ms to 44.8 ms depending on the input clock rate (10 MHz to 625 kHz).

4.2.1.1 COLLISION DETECTION IN SINGLE-ENDED MODE. As an option, the Neuron Chip accepts an active-low Collision Detect input from the transceiver. If collision detection is enabled and CP4 goes low for at least one system clock period (200 ns with a 10 MHz clock) during transmission, the Neuron Chip is signaled that a collision has or is occurring and that the message must be sent again. The node then attempts to reaccess the channel. The collision detect flag is checked by firmware at the end of the preamble and end of packet.

If the node does not use collision detection, then the only way it can determine that a message has not been received is to request an acknowledgment. When acknowledged service is used, retry timer is set to allow sufficient time for a message to be sent and acknowledged (typically 48 ms to 96 ms at 1.25 Mbps when there are no routers in the transmission path). If the retry timer times out, the node attempts to reaccess the channel. The benefit of using collision detection is that the node does not have to wait for the retry timer to time-out before attempting to resend the message, because the node detects the collision when it sends the packet.

4.2.1.2 BETA 1 AND BETA 2 TIMESLOTS IN SINGLE-ENDED MODE. The idle period between packets comprises the *Beta 1* and *Beta 2* timeslots. The *Beta 1* time is the fixed component in the idle period after a packet has been sent. This component is a function of the following:

- Oscillator frequencies and accuracies on the various network nodes.
- Media indeterminate time — that time following packet transmission when the network can appear to be busy due to ringing on the line.
- Minimum interpacket gap — transceiver dependent timing requirements.
- Receive end delay — skew between transmitter and receiver Neuron Chip's view of the end of the packet. This would be due to buffering in the transceivers and is typically found in special-purpose mode transceivers.

A typical value for *Beta 1* time is 370 μ s for a 1.25 Mbps/12-byte packet.

Both priority (P) and non-priority slots are defined by the *Beta 2* time. Nodes listen to the network prior to transmitting a packet. This prevents nodes from transmitting packets on top of each other except when the packets are initiated at nearly the same time. In addition, nodes randomize the time before they start transmitting on the network. When the network is idle, all nodes randomize over 16 slots. When the estimated network load increases, nodes start randomizing over more slots in order to lower the probability of a collision. The number of randomizing slots (R) varies from 16 up to 1008, based on “n,” the estimated channel backlog (the number of slots is $n \bullet 16$ where “n” has a range of 1 to 63). See Figure 4-4.

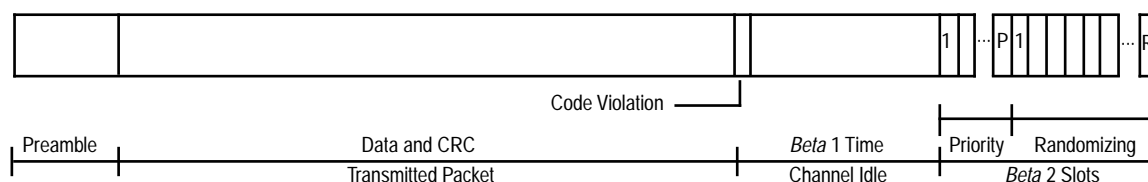


Figure 4-4. Packet Timing

Following a packet and prior to randomizing, nodes wait for a configurable number of priority slots to pass. Nodes with priority packets and a configured priority slot will transmit in that slot. Use of priority substantially reduces the probability of collision. The number of priority slots is fixed for a given channel and can be from 0 to 127.

The *Beta 2* time is defined by the following:

- Oscillator frequencies and accuracies on the various network nodes.
- Number of priority slots on the channel.
- Receive start delay — the time from when a node starts transmitting to when the receiving nodes detect the start of transmission. This is a function of the receive-to-transmit turnaround time of the transceiver, the bit rate and length of the media, delay through the receiver, and initial preamble bits lost.
- For special-purpose mode transceivers, framing delays between the Neuron Chip and the transceiver.

In order for the receiver to detect the edge transitions, two windows are set up for each bit period, T. The first window is set at T/2 and determines if a 0 is being received. The second window is at T and defines a 1. This transition then sets up the next two windows (T/2 and T). If no transition occurs, a Manchester code violation is detected and the packet is assumed to have ended. Section 6.2.8 shows the width of this window as a function of the ratio of the Neuron Chip input clock (MHz) and the network bit rate (Mbps) selected. If a transition falls outside of either window, it is not detected. Timing instability of the transitions,

known as jitter, may be caused by changes in the communications medium, or instability in the transmitting or receiving node's input clocks. The jitter tolerance windows are expressed as fractions of the bit period, T , in Table 4-4.

Table 4-4. Receiver Jitter Tolerance Windows

Ratio of Neuron Chip Input Clock to Network Bit Rate	Next Data Edge			Next Clock Edge			Line Code Violation to Receive
	Min	Nom	Max	Min	Nom	Max	Min
8:1	0.375T	0.500T	0.622T	0.875T	1.000T	1.122T	1.62T
16:1	0.313T	0.500T	0.685T	0.813T	1.000T	1.185T	1.46T
32:1	0.345T	0.500T	0.717T	0.845T	1.000T	1.155T	1.46T
64:1	0.330T	0.500T	0.702T	0.830T	1.000T	1.170T	1.46T
128:1	0.323T	0.500T	0.695T	0.823T	1.000T	1.177T	1.46T
256:1	0.318T	0.500T	0.690T	0.818T	1.000T	1.182T	1.46T
512:1	0.315T	0.500T	0.687T	0.815T	1.000T	1.185T	1.46T
1024:1	0.315T	0.500T	0.687T	0.815T	1.000T	1.185T	1.46T

For the receiver to reliably terminate reception of a packet, the received line-code violation period must have no transitions until the Neuron Chip detects the end of the packet. The receiving Neuron Chip terminates a packet if no clock transitions are detected after the last bit. Table 4-4 shows the minimum duration from the last clock edge to where the Neuron Chip is guaranteed to recognize the line-code violation. Note that data transitions are allowed in this period (and must fall within the data window).

For a Neuron Chip, the time from when an application software call is issued to send a 12-byte message to when the packet is sent, is from 2.8 ms to 44.8 ms, depending on the input clock rate (10 MHz to 625 kHz).

For further information, refer to the *Enhanced Media Access Control with Echelon's LonTalk Protocol* engineering bulletin.

4.2.2 Differential Mode

In the differential mode, the Neuron Chip's built-in transceiver is able to differentially drive and sense a twisted-pair transmission line with external passive components. Differential mode is similar, in most respects, to single-ended mode; the key difference is that the driver/receiver circuitry is configured for differential line transmission. Data output pins CP2 and CP3 are driven to opposite states during transmission and put in a high-impedance (undriven) state when not transmitting. The differential receiver circuitry on pins CP0 and CP1 has selectable hysteresis with eight selectable voltage levels, followed by a selectable low-pass filter with four selectable values of transient pulse (noise) suppression (see Figure 4-5). The selectable hysteresis and filter permit optimizing receiver performance to line conditions. Refer to Section 6.2.5 for hysteresis specifications and Section 6.2.6 for filter specifications.

Figure 4-6 shows a typical packet waveform in differential mode. Note that the packet format is identical to that of the single-ended interface described earlier; the coding and jitter tolerance also apply identically.

The drivers can source 40 mA of current at 4.0 V and can sink 40 mA of current at 1.0 V ($V_{DD} = 5$ V). The programmable hysteresis values shown in Section 6.2.5 correspond to the differential threshold of the receiver. It is recommended that the signal level at the receiver be kept two to three times higher than the programmed threshold.

The receiver input, $V_D = V_{CP0} - V_{CP1}$, must be at least 200 mV greater than hysteresis levels.

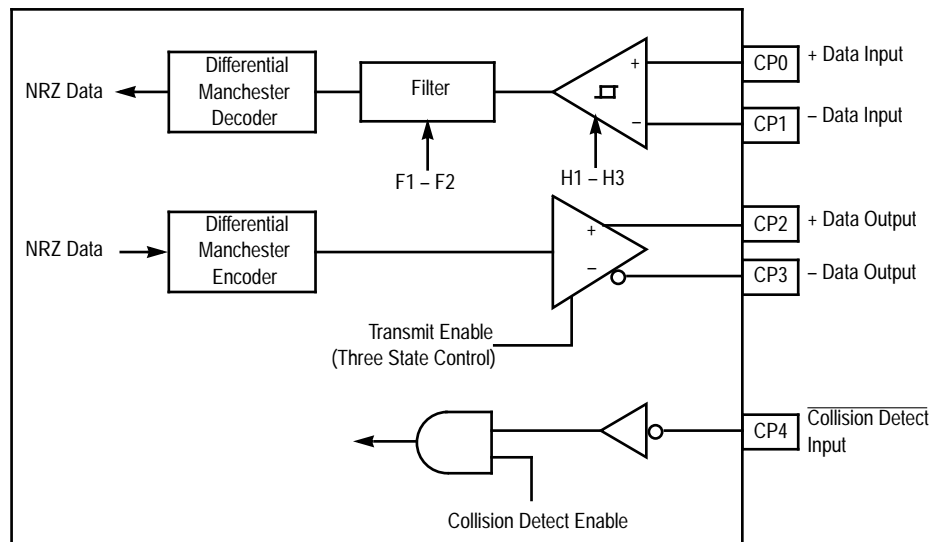


Figure 4-5. Differential Mode

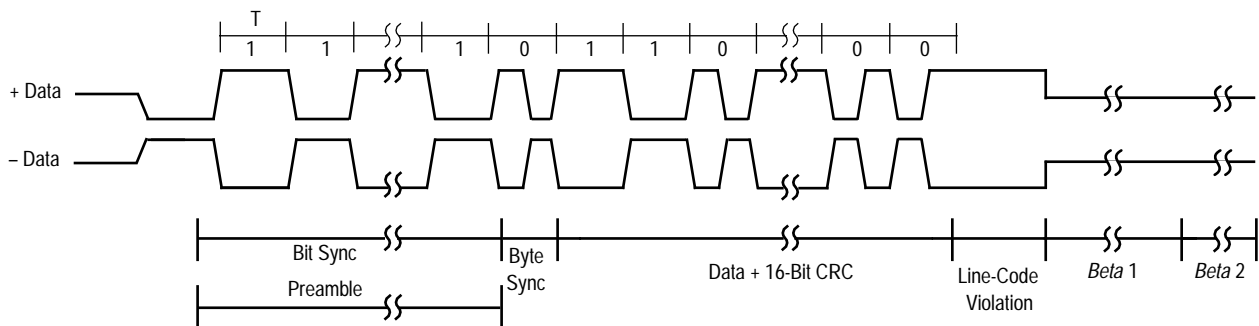


Figure 4-6. Differential Mode Data Format

4.2.3 Special-Purpose Mode

In special situations, it is desirable for the Neuron Chip to provide the packet data in an unencoded format and without a preamble. In this case, an intelligent transmitter accepts the unencoded data and does its own formatting and preamble insertion. The intelligent receiver then detects and strips off the preamble and formatting, and returns the decoded data to the Neuron Chip.

PATENT NOTICE

The Special-Purpose Mode is protected by U.S. Patent No. 5,182,746 and foreign patents based on this patent. No express or implied license is granted herein with respect to such patents. If you are interested in obtaining a non-exclusive, royalty free license to these patents, please call Echelon at (415) 855-7400 and ask for Contracts Management.

Such an intelligent transceiver contains its own input and output data buffers and intelligent control functions, and provides handshaking signals to properly pass the data back and forth between the Neuron

Chip and the transceiver. In addition, there are many features that can be defined by and incorporated into a special-purpose transceiver.

- Ability to configure various parameters of the transceiver from the Neuron Chip
- Ability to report on various parameters of the transceiver to the Neuron Chip
- Multiple channel operation
- Multiple bit rate operation
- Use of forward error correction
- Media specific modulation techniques requiring special message headers and framing
- Collision detection

The special-purpose mode is a restricted protocol that Echelon will license for use only when the Neuron IC and transceiver are sold as one unit. For more details, contact Echelon.

When the special-purpose mode is used, a protocol is utilized between the Neuron Chip and the transceiver. This protocol consists of the Neuron Chip and the transceiver, each exchanging 16 bits (8 bits of status and 8 bits of data, see Figure 4-7) simultaneously and continuously at rates up to 1.25 Mbps (when the Neuron Chip's input clock is 10 MHz). The 1.25 Mbps bit rate allows time-critical flags, such as a Carrier Detect, to be exchanged across the interface with network bit rates up to 156 kbps. The maximum bit rate is 156 kbps due to the overhead associated with the handshaking.

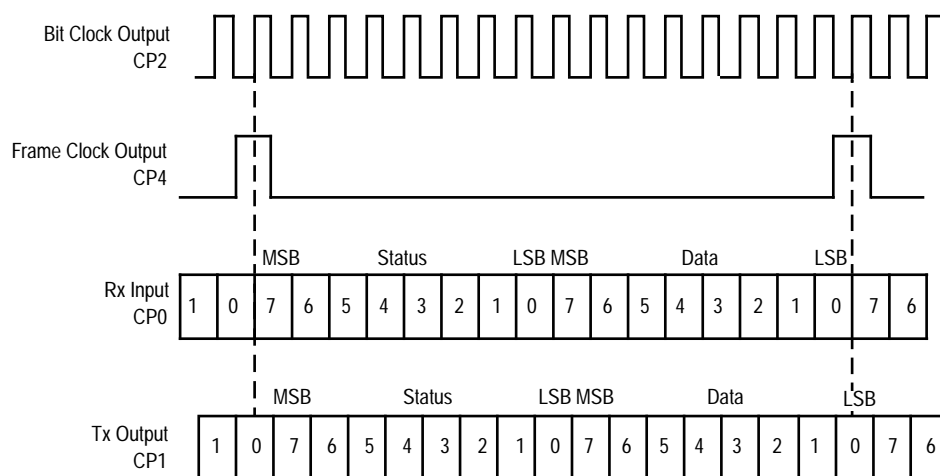


Figure 4-7. Special-Purpose Mode Data Format

The Neuron Chip communicates with the transceiver over CP[4:0]. CP4 and CP2 are synchronizing clocks generated by the Neuron Chip; CP4 is a frame clock and CP2 is a bit clock. CP0 and CP1 contain the exchanged data; CP0 transfers data from the transceiver to the Neuron Chip and CP1 transfers data from the Neuron Chip to the transceiver.

CP3 is used to support low power consumption modes (sleep). Under software control, this pin can be configured as an output to indicate that the Neuron Chip is asleep, or as an input to wake up the Neuron Chip when an incoming packet is detected.

As an output, CP3 high indicates that the transceiver should remain active waiting for incoming or outgoing packets. When CP3 is low, the transceiver can go into a low power mode and ignore network traffic.

As an input, CP3 is used to wake up the Neuron Chip. Any transition (high-to-low or low-to-high) on CP3 will cause the Neuron Chip to wake up. The transceiver should use this to indicate to the Neuron Chip that an incoming packet has been detected on the network.

The Neuron Chip and transceiver continuously exchange data over CP0 and CP1. The bit clock is used to define transitions between bits in the data stream. The Neuron Chip uses the falling edge of the bit clock to both sample CP0 and change CP1 to the next bit. The transceiver should use the rising edge of bit clock to sample CP1 and update CP0.

The serial data streams on CP0 and CP1 are divided into 16-bit frames. The frame clock (CP4) is used to define the boundaries of the frames. The frame clock is active (high) while the Neuron Chip is outputting the least significant bit (LSB) of the frame on CP1. On the falling edge of the frame clock, the Neuron Chip is sampling the most significant bit (MSB) of the next frame on CP0.

The first eight bits of each frame are interpreted as the status field, and the last eight bits are the data field. The status field is used to control transceiver operation and to control passing data between the Neuron Chip and the transceiver. The interpretation of each status bit is shown in Table 4-5.

Table 4-5. Special-Purpose Mode Transmit and Receive Status Bits

Tx Output, Status Bits		
Bit 7	TX_FLAG	Neuron Chip in the process of transmitting packet
Bit 6	TX_REQ_FLAG	Neuron Chip requests to transmit on the network
Bit 5	TX_DATA_VALID	Neuron Chip is passing network data to the transceiver in this frame
Bit 4	Don't Care	Unused
Bit 3	TX_ADDR_R/W	If negated, Neuron Chip is writing internal transceiver register
Bit 2	TX_ADDR_2	Address bit 2 of internal transceiver register (1 ... 7) *
Bit 1	TX_ADDR_1	Address bit 1 of internal transceiver register (1 ... 7) *
Bit 0	TX_ADDR_0	Address bit 0 of internal transceiver register (1 ... 7) *
Rx Input, Status Bits		
Bit 7	SET_TX_FLAG	Transceiver accepts request to transmit packet
Bit 6	CLR_TX_REQ_FLAG	Transceiver acknowledges request to transmit packet
Bit 5	RX_DATA_VALID	Transceiver is passing network data to the Neuron Chip in this frame
Bit 4	TX_DATA_CTS	Transceiver indicates that Neuron Chip is clear to send byte of network data
Bit 3	SET_COLL_DET	Transceiver has detected a collision while transmitting the preamble
Bit 2	RX_FLAG	Transceiver has detected a packet on the network
Bit 1	RD/WR_ACK	Transceiver acknowledges read/write to internal register
Bit 0	TX_ON	Transceiver is transmitting on the network

*Note that internal transceiver register 0 is not valid. Registers 1 – 7 are defined by the transceiver implementation.

There are three types of data that can be sent and received during each frame:

1. Network packet data: Actual data (8 bits at a time) that is to be transmitted or has been received.
2. Configuration data: Information from the Neuron Chip which tells the transceiver how it is to be set up or configured.
3. Status data: Informational parameters reported from the transceiver to the Neuron Chip, when it is requested by the Neuron Chip.

The contents of the configuration data and status data are defined by the transceiver.

The Neuron Chip controls the communication with the transceiver by asserting and examining status bits. There are four basic operations which the Neuron Chip will perform on the transceiver: transmit packet, receive packet, write configuration, and read status.

When the Neuron Chip wants to transmit a packet, it sets the TX_REQ_FLAG bit of its output status field. The transceiver can then accept or reject the request. To reject the request, the transceiver asserts CLR_TX_REQ_FLAG and clears SET_TX_FLAG. The transceiver indicates that it is ready to transmit by asserting CLR_TX_REQ_FLAG and SET_TX_FLAG for one frame. In that same frame, the transceiver must also assert TX_DATA_CTS, indicating that the Neuron Chip may send the first byte of data.

The Neuron Chip will send a packet of data only if the transceiver accepts the transmit request. The Neuron Chip will then assert TX_FLAG for the entire duration of the packet. The transceiver must assert TX_ON as long as it is transmitting a packet.

Each byte is transferred from the Neuron Chip to the transceiver with a handshake protocol. The transceiver indicates that it is ready to accept a byte by asserting TX_DATA_CTS for a single frame. The Neuron Chip uses this flag to cause it to send out another byte in a subsequent frame; the Neuron Chip will also assert TX_DATA_VALID during the frame which contains the data byte.

After the Neuron Chip has sent the last byte in the packet, it clears TX_FLAG to indicate the end of transmission. When the transceiver finishes transmitting the packet, including any error codes, it must clear TX_ON to indicate that it has released the network.

The transceiver may abort transmission if it detects a collision by asserting SET_COLL_DET for one frame. The Neuron Chip will then clear TX_FLAG and prepare to resend the packet.

The transceiver initiates packet reception by asserting RX_FLAG. The transceiver can begin sending data to the Neuron Chip in the frame after asserting RX_FLAG. Each frame which contains valid data must be marked with RX_DATA_VALID asserted. When the transceiver is finished receiving a packet, it clears RX_FLAG and the Neuron Chip terminates reception of the packet.

The Neuron Chip performs a configuration write or status read by using TX_ADDR_R/W and TX_ADDR_[2:0]. TX_ADDR_[2:0] indicates which of seven transceiver registers is being accessed, and TX_ADDR_R/W indicates whether the operation is a configuration register write (0) or status register read (1). Register 0 (TX_ADDR_[2:0] = 000) is unused, so that TX_ADDR_R/W = 0 and TX_ADDR_[2:0] = 000 indicates no read or write operation is to be performed.

To write to a configuration register, the Neuron Chip clears TX_ADDR_R/W and indicates the selected register with TX_ADDR_[2:0]. The transceiver must acknowledge that the operation is complete by asserting RD/WR_ACK. The Neuron Chip will continue to send the configuration write command until it receives a frame with RD/WR_ACK asserted.

To read a status register, the Neuron Chip asserts TX_ADDR_R/W and indicates the selected register with TX_ADDR_[2:0]. The transceiver must acknowledge that the operation is complete by asserting RD/WR_ACK and by placing the requested information in the data field. The Neuron Chip will continue to send the status request command until it receives a frame with RD/WR_ACK asserted.

4.3 TWISTED-PAIR TRANSCEIVERS

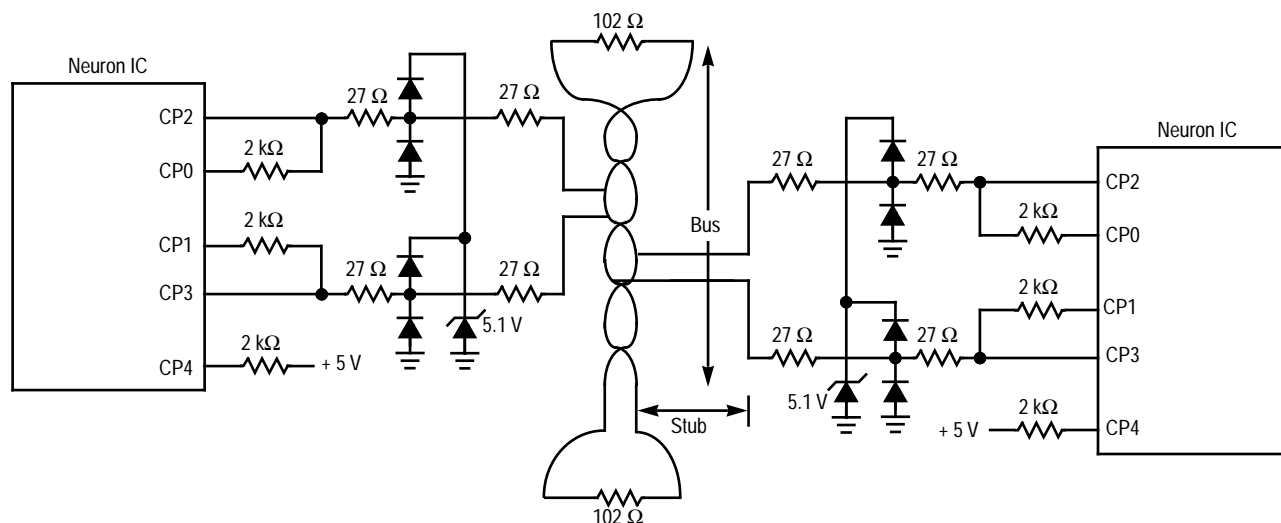
Twisted-pair transceivers are the most common type; they can be configured in many designs to support a wide range of cost and performance requirements. There are three basic types of twisted-pair interfaces for the Neuron Chip: direct-drive, EIA-485, and transformer-coupled.

4.3.1 Direct-Drive

Direct-drive interface uses the Neuron Chip's internal transceivers with external resistors and diodes for current limiting and electrostatic discharge (ESD) protection as shown in Figure 4-8a. Direct-drive is intended only for networks of nodes with a common power source that would be implemented inside equipment for bussing various boards together. Examples include backplanes of telecommunications

equipment, copying machines, vending machines, and appliances. Varying data rates up to 1.25 Mbps can easily be supported up to hundreds of feet (UL Level IV wire), with up to 64 nodes on the network. In this configuration, the common mode range is limited to 0.9 to $V_{DD} - 1.75$ V. The circuit configuration uses a $2\text{ k}\Omega$ resistor for input ESD protection of the receiver input pins, and a $51\ \Omega$ resistor for line-balancing, short-circuit, and overvoltage protection. It is advisable to tie the V_{DD} clamp diodes to a separate 5.1 V zener diode for overvoltage protection.

Figure 4-8b illustrates the use of a low cost direct-drive network connected through a router to a network using transformer isolated transceivers. The importance of this configuration is the complete use of one network manager for managing all of the nodes.



**Figure 4-8a. Simple Direct-Connect Network Interface
(Used with Direct Mode Differential)**

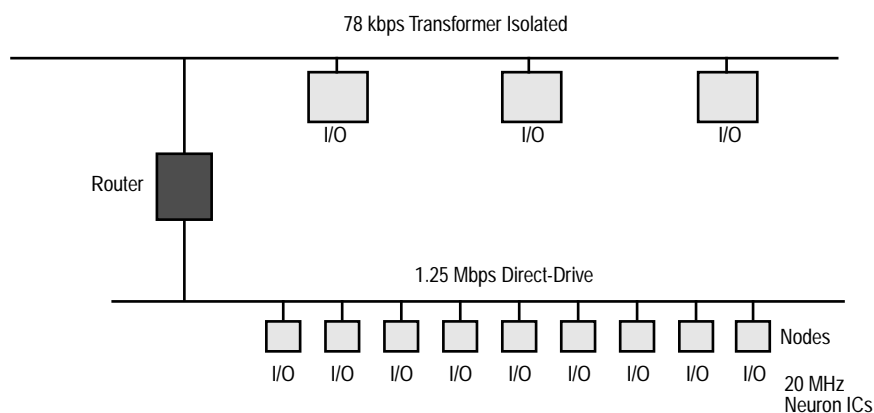
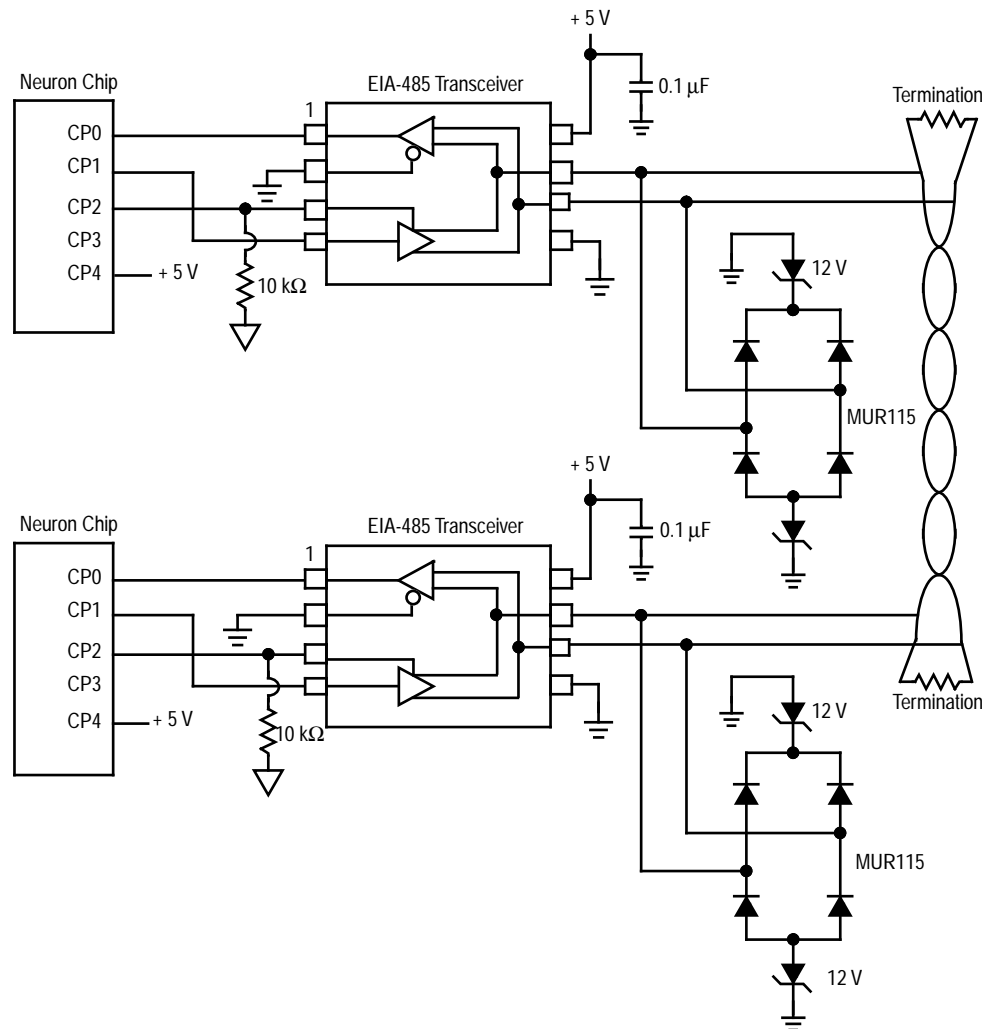


Figure 4-8b. Low Cost/Small Size Network

4.3.2 EIA-485

Commercially available EIA-485 transceivers allow for cost/performance/size capability over the other transceiver solutions. Multiple data rates (up to 1.25 Mbps), and a number of wire types can be supported with no change in component values. With an EIA-485 transceiver, common mode voltage range is better than the range available using direct-drive, but less than with the transformer-coupled transceivers. Common mode voltage ranges are between -7 V to $+12$ V. Optoisolators can be added to increase the common

mode voltage range. EIA-485 chips can be found in both bipolar and CMOS versions. CMOS versions use less power, but do not have the drive output of bipolar devices; and in general, are more expensive than bipolar versions. The communications port of the Neuron Chip must be put in single-ended mode for implementing an EIA-485 network. Available industry standards covering EIA-485 specifications give details on unit loads, data rate, wire size, and wire distances. To ensure interoperability between nodes, the LONMARK guidelines recommend a data rate of 39 kbps for nodes using EIA-485 transceivers. A typical circuit configuration, as shown in Figure 4-9, can support 32 loads. EIA-485 works best if a common power source is used. Individual node power sources create many problems when the common voltage exceeds -7 V , $+12\text{ V}$, or when excess ground faults cause damage to nodes.



**Figure 4-9. EIA-485 Twisted-Pair Interface
(Used with Single-Ended Mode)**

The EIA-485 specification requires a common ground reference at all transceivers. This may be provided by a third conductor in the network cable or a separate connection to common ground at each node.

4.3.3 Transformer-Coupled

Transformer-coupled interfaces work well for applications that require higher performance, high common mode isolation, and noise immunity between nodes. Transformer-coupled transceiver designs can operate at up to 1.25 Mbps and achieve common mode ranges of $\pm 277\text{ Vrms}$. There are various types of transformer configurations. Designers may develop their own transformer-coupled circuits for either two- or four-wire configuration. Echelon also offers a family of transceivers operating at either 78 kbps or 1.25 Mbps

data rates, as shown in Table 4-6. The most versatile of Echelon's transceivers are the free topology type; available in both transformer-isolated or link power versions, supporting bus, loop, or star wiring configurations.

When laying out a circuit board, it is crucial to make certain that the components are close to the Neuron Chip and that the transformer traces run together to the connector, isolated from fast changing signal lines. As shown in Table 4-7, various companies sell transformers optimized for either 78 kbps or 1.25 Mbps.

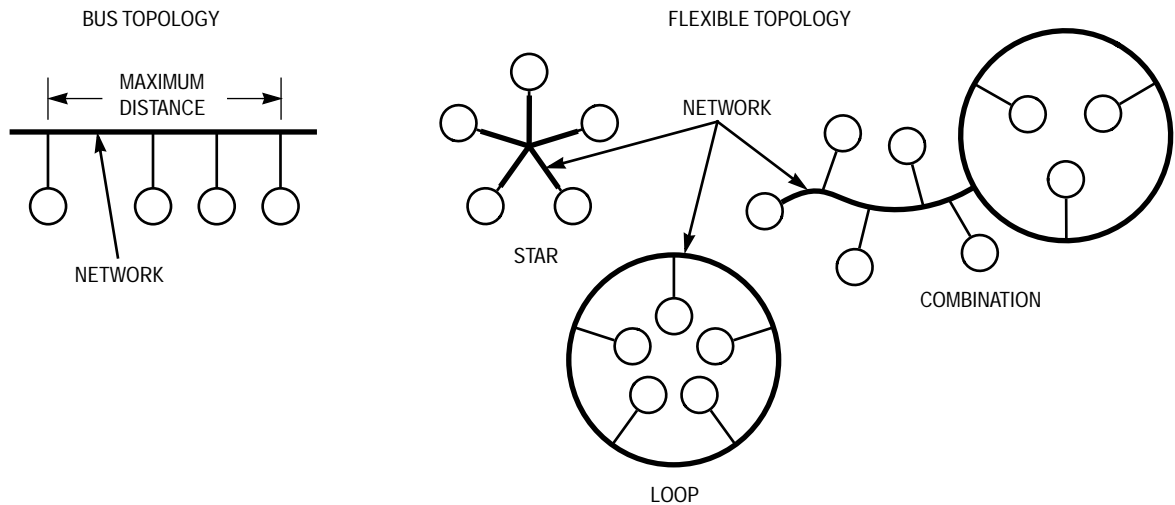


Figure 4-10. Twisted-Pair Topologies

Table 4-6. Echelon Transceiver Products

Product	Rate	Topology	Nodes	Distance	Type
TPT/XF-78	78 kbps	Bus	64	1400 m	Transformer Isolated
TPT/XF-1250	1.25 Mbps	Bus	64	130 m	Transformer Isolated
FTT-10A	78 kbps	Bus	64	2700 m	Transformer Isolated
FTT-10A	78 kbps	Free	64	500 m	Transformer Isolated
LPT-10	78 kbps	Bus	128	2200 m	Link Power
LPT-10	78 kbps	Free	128	500 m	Link Power

Table 4-7. Twisted-Pair Transformer Manufacturers for 78 kbps and 1.25 Mbps

	78 kbps	1.25 Mbps
Part Number	0505-0542 Thru Hole 0505-0642 Surface Mount 0505-0662 Surface Mount	PE-65948 Thru Hole PE-65848 Surface Mount
Turns Ratio Lineside Inductance	2:1 = 35 mH	1:1 = 3.5 mH
Company	Precision Components Inc.	Pulse Engineering, Inc.
Address	400 W. Davy Lane Wilmington, Illinois 60481	7250 Convoy Court San Diego, CA 92111
Phone Number	630-980-6448	619-674-8100

If in design considerations, there is a need to meet the LONMARK interoperability standards, the user may purchase a transceiver from Echelon, or acquire the LONMARK specifications and create a custom design.

Figure 4-11 shows a block diagram of a transformer isolated twisted-pair interface design, along with various recommended protection circuits. The filter conditioning circuit, which needs to be determined based on which data rate frequency, is used.

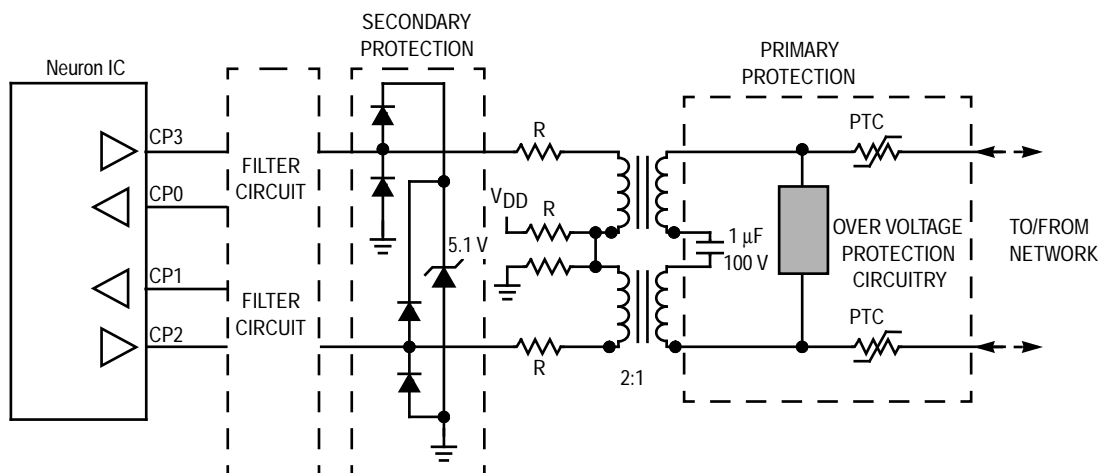


Figure 4-11. Basic Transformer with Signal Conditioning

The Neuron Chip has an enhanced output protection structure on the communications port pins, which eliminates the need for an external relay when implementing a transformer design. (The external relay was previously needed with the 1.2µ Neuron Chip because when the Neuron Chip powered down, the internal diodes would become a low impedance for the transformer, potentially loading down the network and/or possibly latching up the Neuron Chip.)

When designing for communications line protection as well as I/O line protection, there are numerous considerations, either self-imposed or various standards that may have to be met, for example: UL864, UL1459, CSA-C22.2, Bellcore 1089, ITU-T, formerly CCITT K.201, EC101, FCC Part 68, or the National Electric Code (NEC). These specifications address a variety of concerns, including:

- dc voltage being applied accidentally across the communication port
- ESD or other energy surges that can be picked up by a long wire
- 120 V or 220 Vac shorting across the communication wire

These concerns raise various issues regarding cost of the protection circuit, space required on the PC board, and effect of the capacitance that the components may add to the network. Secondary protection, as shown in Figure 4-11, is required at a minimum to reduce any voltage surges that may come across the transformer. The diode structure will clamp any surge to 0.7 V below ground and 0.7 V above the 5.1 V zener. The zener is used in lieu of connections to the + 5 V power supply because most 7805 regulators go unregulated when a surge above + 5 V occurs.

For primary protection against dc voltages being applied across the transformer, a single capacitor could be applied across the center tap of the transformer as shown in Figure 4-11. To protect the capacitor from breakdown caused by up to 20 kV ESD and other surges that can be picked up on long wire runs, would require a back-to-back zener, SIDACtor™, or other transient protection schemes across the line. If a 100-V capacitor is used, then a zener pair or transient voltage suppressor should be used at the appropriate voltage to protect the capacitor.

If the design requires protection against 120 Vac or 220 Vac being shorted across the communication line, then the protection circuit design becomes much more complex. A poly resistor or positive temperature coefficient (PTC) could be used. It changes from low impedance to high impedance when the current rating is exceeded.

4.4 OTHER TRANSCEIVER EXAMPLES

4.4.1 Powerline Transceivers

Harsh noise and wiring constraints complicate the implementation of powerline transceivers. Echelon offers a family of transceivers which meet both North American and European specifications, and perform quite well. When interfacing to powerline transceivers, the Neuron Chip is configured in the special-purpose mode and can achieve data rates of up to 10 kbps.

Table 4-8. Echelon's Powerline Transceivers

Product	Bit Rate (kbps)	Band (kHz)	Region
PLT-10A	10	100 – 450	Special Application
PLT-21	5	125 – 140	Europe/North America
PLT-30	2	9 – 90	Europe (Meter Reading)

4.4.2 Radio Frequency Transceivers

RF applications using LONWORKS technology offer a wide range and choice of frequencies. Consult Echelon's web site for information on RF transceivers.

4.5 CLOCKING SYSTEM

4.5.1 Clock Generation

The Neuron Chip divides the input clock by a factor of two to provide a symmetrical on-chip system clock. The input clock may be generated either by an external free-running source, or by an on-chip oscillator using an external crystal or ceramic resonator.

The Neuron Chip includes an oscillator that may be used to generate an input clock using an external crystal. The valid input clock frequencies are: 20 MHz, 10 MHz, 5 MHz, 2.5 MHz, 1.25 MHz, and 625 kHz. Alternatively, an externally generated clock may drive the CMOS input pin CLK1 of the Neuron Chip, in which case CLK2 must be left unconnected or used to drive no more than one external CMOS load. The accuracy of the clock frequency must be $\pm 0.15\%$ (1500 ppm) or better, to ensure that nodes may correctly synchronize their bit clocks. It is recommended that a 0.02% (200 ppm) or better, clock source be used for LONMARK product interoperability. Crystal values are listed down to 625 kHz. See Figure 4-12 and Table 4-9. A 60/40 duty cycle or better, is required when using an external oscillator as shown in Figure 4-13.

For multiple memory configurations (external EPROM, EEPROM, and/or SRAM), additional cost savings due to slower memory map decode logic, and lower cost of the memory, can be realized at 5 MHz. Power consumption will be 30%–40% lower at 5 MHz than at 10 MHz. Issues such as required data rate and I/O response time must be considered. For example, even if a Neuron Chip is operating at 10 MHz, it can not generate enough 12- to 14-byte-length packets to saturate a 1.25 Mbps channel and there is typically little network-throughput advantage to be gained when operating at this speed over 625 kHz (the maximum data rate with a 5 MHz clock input). End-to-end response time will double (for example, from 20 ms to 40 ms when using acknowledged service) due to slower packet generation and reception at 5 MHz, but not due to congestion on the network.

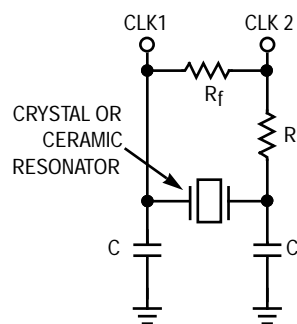


Figure 4-12. Neuron Chip Clock Generator Circuit

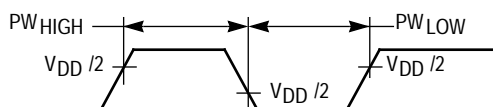


Figure 4-13. Test Point Levels for CLK1 Duty Cycle Measurements

Table 4-9. Typical Clock Generator Component Values

Input Clock Frequency	Crystal			Ceramic Resonator		
	R_f	R_d	C	R_f	R_d	C
20.0 MHz	1 M Ω	120 Ω	30 pF	Open	0 Ω	5 pF
10.0 MHz	100 k Ω	270 Ω	30 pF	Open	270 Ω	30 pF
5.0 MHz	100 k Ω	470 Ω	30 pF	Open	270 Ω	30 pF
2.5 MHz	100 k Ω	1 k Ω	36 pF	Open	1 k Ω	36 pF
1.25 MHz	100 k Ω	1.2 k Ω	47 pF	Open	1.2 k Ω	47 pF
625 kHz	100 k Ω	2.7 k Ω	47 pF	Open	1.2 k Ω	100 pF

NOTES:

- The capacitor values include stray capacitances; crystal or ceramic resonator manufacturers may recommend other values.
- There is internal 1M Ω feedback element (nominal) across CLK1 and CLK2 in the Neuron Chip.
- Values of R_d , the damping resistor, are nominal and can be optimized using crystal/resonator manufacturers' recommendations/data sheets.
- Crystal or ceramic resonator frequency = input clock frequency.
- Crystal may be parallel resonant.
- NPO-type (non-polarized) ceramic capacitors are recommended.
- Resistor and capacitor tolerance is $\pm 5\%$.
- A source for the ceramic resonators is:

Murata	10 MHz	CSA 10.0 MTZ
2200 Lake Park Drive	5 MHz	CSA 5.0 MG
Smyrna, Georgia 30080-7604	2.5 MHz	CSA 2.50 MG
Tel: 770-436-1300	1.25 MHz	CSB 1250J
	0.625 MHz	CSB 625J
- A source for the crystals is:

Fox Electronics
5570 Enterprise Pkwy
Fort Myers, FL 33905
Tel: 941-693-0099
- Motorola can not recommend one supplier over another.

I/O response time will also double when operating at 5 MHz, and this must be considered. Proper prioritizing of *when* statements in the application program and use of timer counter objects can help to minimize latency. For nodes that require faster I/O response time, a 68HC11 or 68HC05 microcontroller unit (MCU),

with their hardware interrupt capability and interfacing to the Neuron Chip in one of the parallel modes is an option.

Typical start-up times for this clock circuit with the Neuron Chip are shown in Table 4-10. Actual start-up times vary depending on the crystal or ceramic resonator used.

Table 4-10. Typical Start-Up Times

Input Clock Frequency (MHz)	Crystal (ms)	Ceramic Resonator (μ s)
10.0	0.45	8
5.0	1.0	15
2.5	2.0	30
1.25	3.0	60
0.625	10	380

4.6 ADDITIONAL FUNCTIONS

4.6.1 Sleep/Wake-Up Circuitry

The Neuron Chip may be put into a low-power sleep mode under software control. In this mode, the system clock and all timer/counters are turned off, but all state information (including the contents of on-chip RAM) is retained. Normal operation resumes when an input transition occurs on any one of the following:

- I/O pins (maskable):
One of IO4 – IO7, selected by the timer/counter multiplexer
- $\overline{\text{SERVICE}}$ pin (not maskable)
- Communications port (maskable):

Differential direct mode	CP0 or CP1
Single-ended direct mode	CP0
Special-purpose mode	CP3

The application software may optionally specify that the programmable pull-ups on the $\overline{\text{SERVICE}}$ pin and I/O pins IO4 – IO7 be disabled to further reduce power consumption.

While sleeping, the I/O pins and $\overline{\text{SERVICE}}$ pin retain the state they had just before sleep was invoked. For example, if IO[7:0] were outputting a byte of data, that byte remains on these pins for the duration of sleep.

If the communications port is transmitting a packet when the application attempts to put the Neuron Chip to sleep, the Neuron Chip waits until the packet has been sent before going to sleep.

On the MC143150B1/B2, the $\overline{\text{E}}$ pin is held inactive (high) during sleep to disable memory operations during this period. The oscillator is disabled with CLK2 driven high, addresses A15 – A0 are forced to FFFF, $\text{R}/\overline{\text{W}}$ is forced to 0, and data is undetermined but driven high or low, so data lines will not float and draw excess current.

When a wake-up event is detected (transition on $\overline{\text{SERVICE}}$ pin, selected I/O pin, or communication port wake-up pin), the Neuron Chip allows the oscillator to start up, waits for it to stabilize, performs internal maintenance, and then resumes operation. Typical oscillator start-up times are discussed in Section 4.5, *Clocking System*. The Neuron Chip allows for up to 15 transitions on CLK1 after the oscillator has started up for the oscillator to stabilize.

The amount of time required for the Neuron Chip to perform internal maintenance after waking up before it resumes execution of the application, depends on several application parameters and on whether the network processor is servicing application timers during this period. The important application parameters

are the comm ignore option, the number of receive transactions (if comm ignore is used), and the number of application timers (if the network processor services the application timers during this period).

If the comm ignore option is not used, the Neuron Chip typically requires about 2000 CLK1 cycles to perform internal maintenance, and worst case about 47,000 CLK1 cycles. The typical case assumes that the network processor does not service the application timers during this period. The worst case assumes that the network processor must service the maximum number of application timers (15) during this period.

If the comm ignore option is used, the Neuron Chip typically requires about 7200 CLK1 cycles, and worst case about 66,000 CLK1 cycles. The typical case assumes that four receive transactions are specified, and the network processor does not service application timers during this period. The worst case assumes the maximum number of receive transactions (16) and that the network processor must service the maximum number of application timers (15) during this period.

4.6.2 Reset Function

The reset function is a critical operation in any microcontroller. In the case of the Neuron Chip, it plays a key role in the following conditions:

- Initial V_{DD} power up (ensures proper initialization of the Neuron Chip).
- V_{DD} power fluctuations (manages proper recovery of Neuron Chip after V_{DD} stabilizes).
- Program recovery (if an application gets lost due to corruption of address or data, an external reset can be used for recovery or the watchdog timer could timeout, causing a software reset).
- V_{DD} power down (ensures proper shut down).
- Help to protect the EEPROM from major corruption.

The Neuron Chips have four mechanisms that reset can be initiated by:

- \overline{RESET} pin (input pin is pulled low for at least 20 ns then returned high).
- Watchdog timeout (timer expires after 840 ms with 10 MHz CLK forcing \overline{RESET} pin low for at least 20 ns, then returning high).
- Software command (application program can initialize a soft reset or a received network command – reset message), when either command is received the \overline{RESET} pin is forced low for at least 20 ns, then returned high.
- LVI (an internal low voltage detect circuit will force the \overline{RESET} pin low if the supply drops below a set level).

During any of the reset functions, when the \overline{RESET} pin is in the low state, the Neuron Chip pins goes to a state that is shown in the list below. Figures 4-16, 4-17a, and 4-17b also illustrate the condition of the pins during reset low and the Neuron Chips initialization sequence after reset is returned high again.

- I/O pins go high impedance
- All processor functions stop immediately
- All output address pins go to 0xFFF immediately
- All data pins become outputs with high or low states
- \overline{E} clock goes high (MC143150 only)
- $\overline{SERVICE}$ pin goes unknown
- Communications pins (CP0 – CP4) go high impedance
- R/\overline{W} goes low
- Oscillator continues to run

When the $\overline{\text{RESET}}$ pin is released back to a high state, the Neuron IC begins its initialization procedure starting at address 0x0001. The time it takes the Neuron IC to complete its initialization differs between the different Neuron ICs and the different firmware versions that are being run. This will be discussed later in this section.

4.6.3 $\overline{\text{RESET}}$ Pin

The $\overline{\text{RESET}}$ pin is both an input and output. As an input, the $\overline{\text{RESET}}$ pin is internally pulled high by a 60 μA to 260 μA current source acting as a pull-up resistor.

The $\overline{\text{RESET}}$ pin has an open-drain N-channel structure capable of sinking 20 mA at 0.8 V or 10 mA at 0.4 V. The $\overline{\text{RESET}}$ pin becomes an output when the N-channel structure is initialized from either:

- Watchdog Timer event.
- Software reset initialization.
- Internal LVI detects a low voltage.
- $\overline{\text{RESET}}$ pin drops below the internal trip point.

4.6.3.1 POWER SEQUENCE. During power up sequences, the $\overline{\text{RESET}}$ pin should be held in the low state until the power supply is stable, to prevent start-up malfunctioning. Likewise, when powering down, the Neuron IC $\overline{\text{RESET}}$ pin should go to a low state before the power supply goes below the minimum operating voltage of the Neuron IC.

It is important to understand that if proper reset recovery circuitry is not used, the Neuron Chip can go applicationless or unconfigured. The applicationless or unconfigured state occurs when the background checksum error checking routine detects a corruption in memory which could have falsely been detected due to improper reset sequence or noise on the power supply. Several options exist on the LonBuilder development tool to allow a reset on checksum failure.

Reset Sequence. After initial power is applied, reset is held low by external or internal LVI, until the voltage reaches LVI trip point (point A in Figure 4-15, approximately 4.3 V), at which time the LVI releases its hold on the $\overline{\text{RESET}}$ pin and begins to charge at a rate determined by the size of the external capacitor and the constant current rate from the Neuron IC $\overline{\text{RESET}}$ pin, between 60 μA and 260 μA . At point B, $V_{\text{DD}} - 0.7 \text{ V}$, the internal reset point is reached and the Neuron Chip begins its reset start-up sequence. If an external reset or an internal software reset (point C) is initiated, the Neuron IC stops functioning when the $\overline{\text{RESET}}$ pin reaches a voltage of $0.3 V_{\text{DD}}$. The restart sequence initiates after the $\overline{\text{RESET}}$ pin has charged back up to $V_{\text{DD}} - 0.7 \text{ V}$ (point D). If the power supply takes a dip and falls below the internal or external LVI trip point (point E), approximately 4.3 V, the LVI will activate. When power falls below $0.3 V_{\text{DD}}$, the Neuron IC resets itself. When the power supply goes above the LVI release voltage (point F), the $\overline{\text{RESET}}$ pin begins recharging. Upon reaching point G, $V_{\text{DD}} - 0.7 \text{ V}$, the Neuron IC will again restart. The final event is powering down, at which point the LVI trip point will activate low and place the Neuron IC in a reset state.

The total capacitance directly connected to the $\overline{\text{RESET}}$ pin, including stray and external device input capacitance, may not exceed 1000 pF. This guarantees the Neuron Chip can successfully output a reset down to below 0.8 V. The 100 pF minimum capacitance is used for noise immunity.

4.6.3.2 SOFTWARE CONTROLLED RESET (ALL PARTS). When the CPU watchdog timer expires, or a software command to reset occurs, output N-channel transistor turns on and pulls down the $\overline{\text{RESET}}$ pin. This transistor has the same drive capability as the other high sink pins (20 mA @ 0.8 V). When the low threshold detect (0.8 V) senses the $\overline{\text{RESET}}$ pin is low, it sends a signal through the control logic to shut off the N-channel transistor. The $\overline{\text{RESET}}$ pin and required capacitor ($100 \leq x \leq 1000 \text{ pF}$) are allowed to begin charging and provide the required duration of reset.

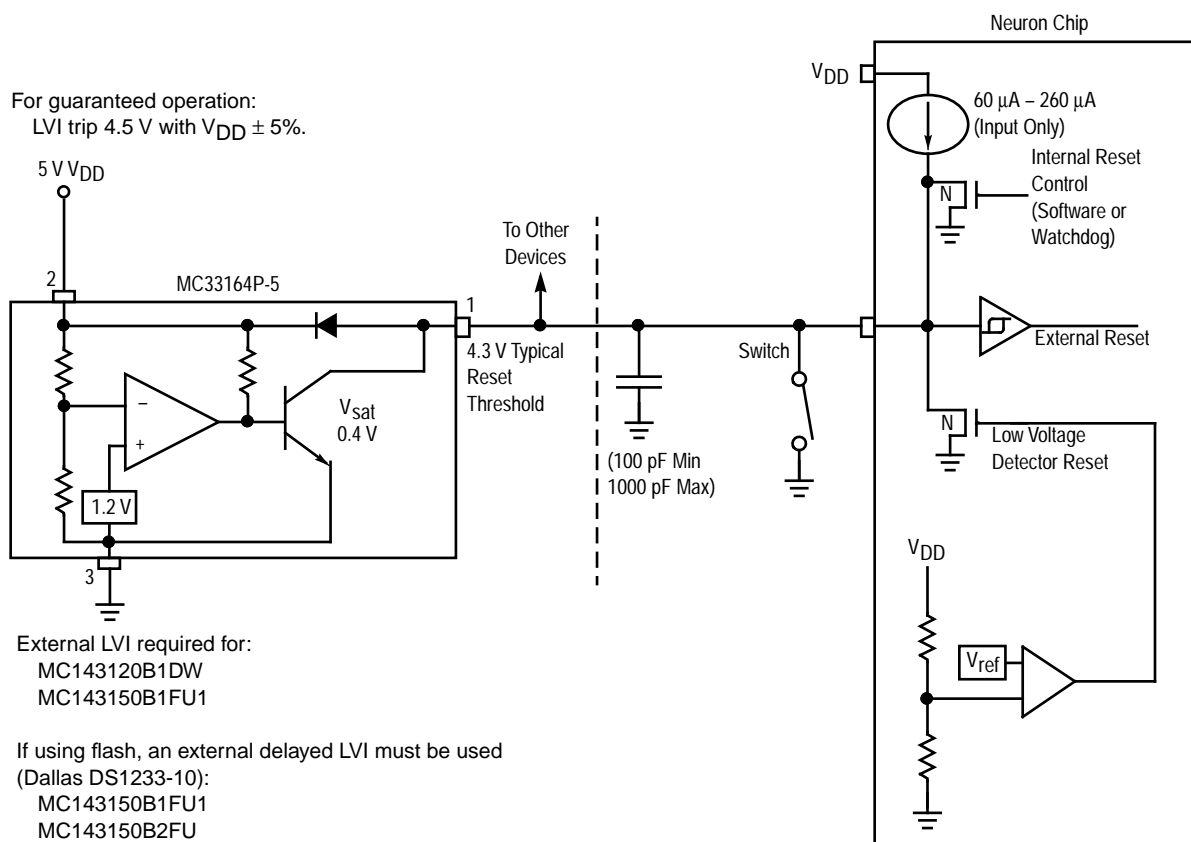


Figure 4-14. Example of Reset Circuit

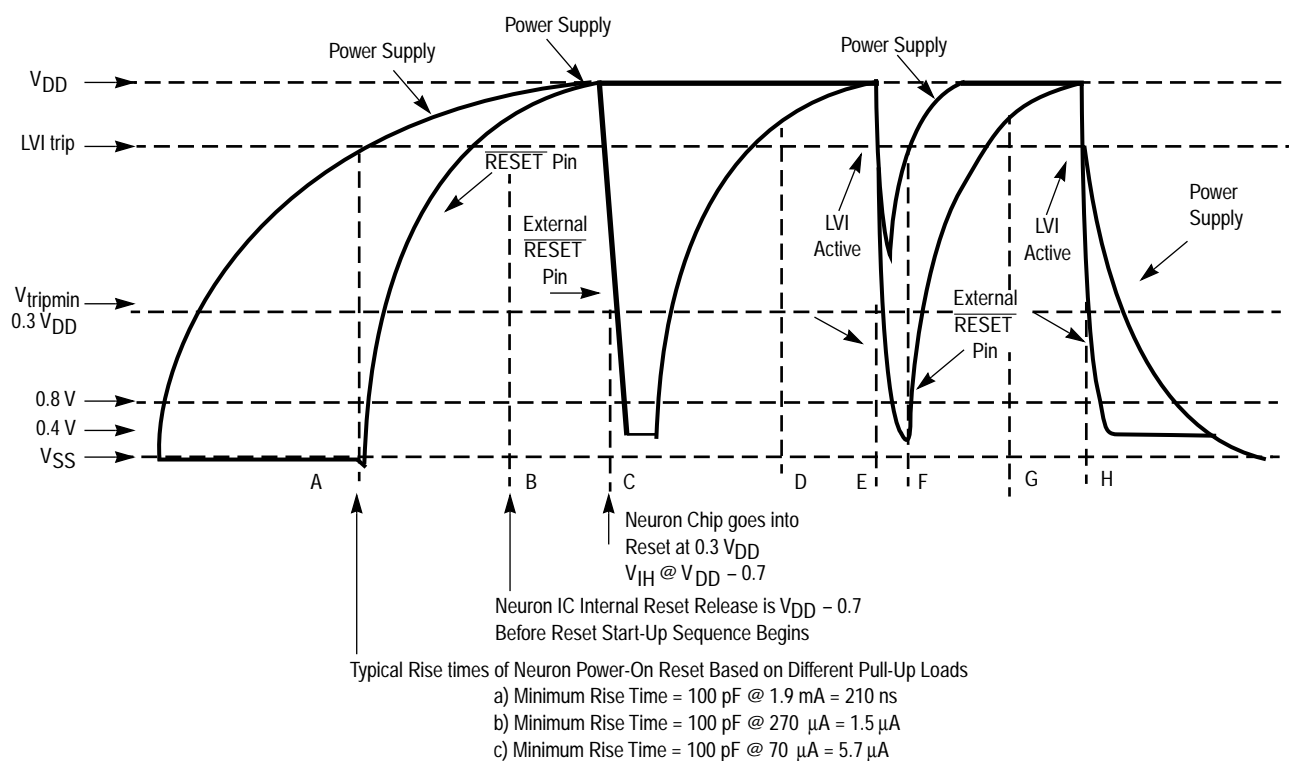


Figure 4-15. Reset Timing Diagram for the Neuron Chip

4.6.3.3 WATCHDOG TIMER. The Neuron Chip processors are protected against malfunctioning software or memory faults by three watchdog timers, one per processor. If application or system software fails to reset these timers periodically, the entire Neuron Chip is automatically reset. The watchdog period is approximately 0.84 seconds at maximum input clock rate (10 MHz — the MC143120FE2 can operate at 20 MHz) and scales inversely with the input clock rate. When the Neuron Chip is in sleep mode, all the watchdog timers are disabled.

4.6.3.4 LVI CONSIDERATIONS. Since the $\overline{\text{RESET}}$ pin of the Neuron Chip is bidirectional, the LVI must have an open-drain or open-collector output. If the LVI actively drives the $\overline{\text{RESET}}$ pin high, then the Neuron Chip will not be able to reliably assert the $\overline{\text{RESET}}$ pin (low) during internal resets. This contention on the Neuron Chip $\overline{\text{RESET}}$ pin can cause anomalous behavior, from “applicationless” errors to physical damage to the Neuron Chip reset circuitry. In applications where the $\overline{\text{RESET}}$ pin is subject to potential ESD effects, proper external protection circuitry should be added. A minimum series resistance of about 300 Ω is recommended.

The purpose of the LVI is to ensure that the Neuron IC only operates above the 4.3 V range. If the circuit is allowed to operate below this voltage, there is the possibility that improper operation could occur. If the Neuron IC is writing to an internal or external EEPROM when a reset event is initiated, then that byte could be corrupted. When using external flash memory for the MC143150 devices, an external delayed LVI of greater than 50 ms should be used (recommend using Dallas Semiconductor Part No. DS1233-10).

Possible Motorola LVIs are listed in Table 4-11. These LVIs have open-collector outputs. If using a different LVI, be sure to check the data sheet carefully to ensure that it has an open-drain or open-collector output.

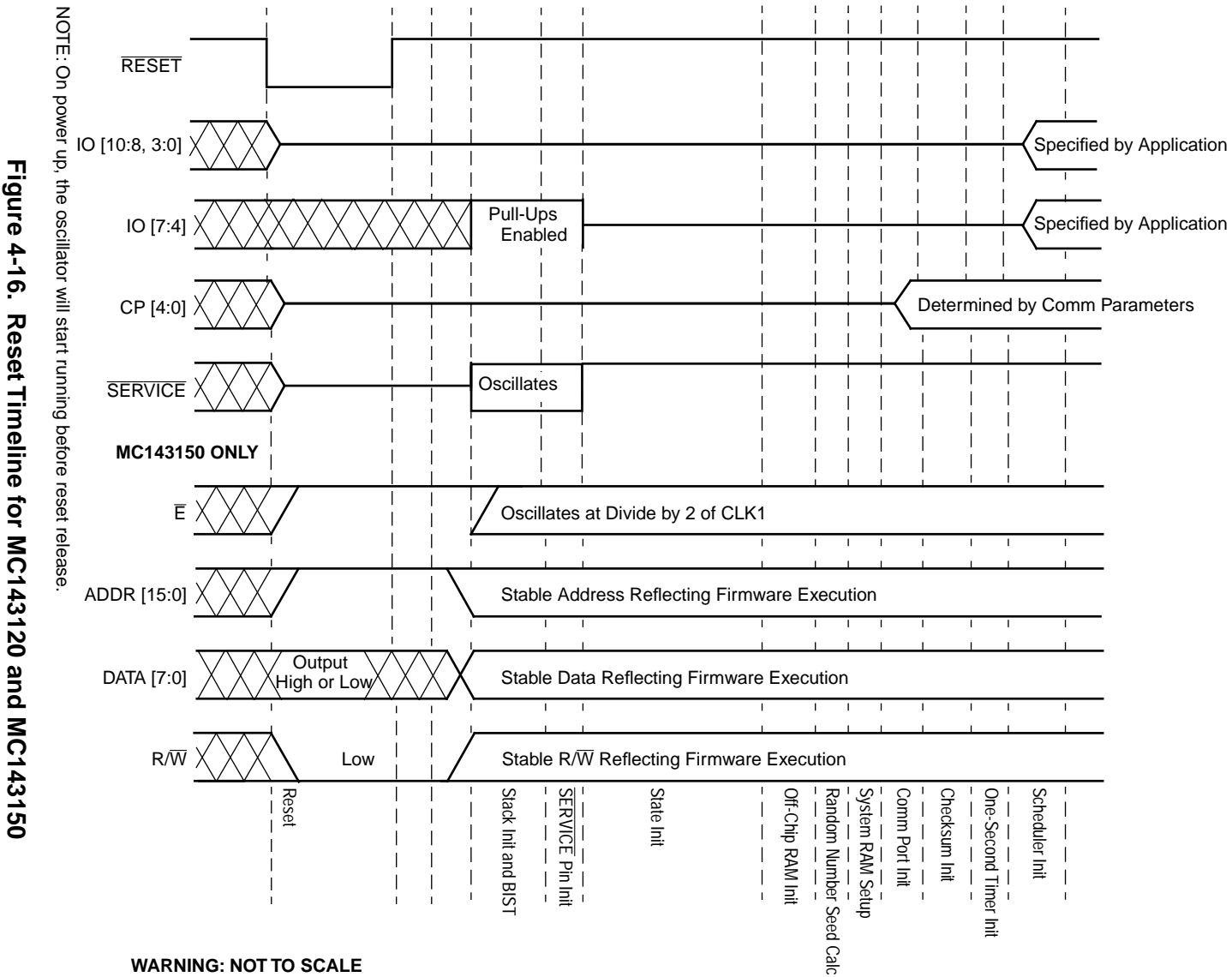
Table 4-11. Comparison of Motorola Low-Voltage Detector ICs

Device	Suffix	Package	Temperature (°C)	Reset Voltage Range (V)			Power Consumption (μA) ($V_{\text{in}} = 5 \text{ V}$)	
				Min	Typ	Max	Typ	Max
MC34064*	D-5 P-5	SO-8 TO-266AA	0 to 70	4.5	4.59	4.7	390	500
MC33064	D-5 P-5	SO-8 TO-226AA	– 40 to + 85	4.5	4.59	4.7	390	500
MC34164	D-5 P-5	SO-8 TO-226AA	0 to 70	4.15	4.33	4.45	12	20
MC33164	D-5	SO-8	– 40 to + 85	4.15	4.33	4.45	12	20

*The recommendation to use an MC34064 is compatible with a 5% V_{DD} specification. When using an MC33164 to accommodate a 10% V_{DD} and the industrial temperature range (– 40 to + 85°C), it must be emphasized that the Neuron Chip (excluding MC143120LE2) is only guaranteed to operate down to a 4.5 V supply.

4.6.4 Reset Processes and Timing

During the reset period, the I/O pins, communication port pins, and $\overline{\text{SERVICE}}$ pin are in a high-impedance state. The MC143150B1/B2, address lines A15 – A0 are forced to 0xFFFF, $\text{R}/\overline{\text{W}}$ is forced to 0, and $\overline{\text{E}}$ is forced to 1. The data is undetermined but driven high or low, so data lines will not float and draw excess current. Reset overrides the effect of $\overline{\text{E}}$ clock on data lines in that, in normal operations the data bus is only driven in a write cycle during the $\overline{\text{E}}$ clock low portion of the bus cycle, while reset forces the data bus to be driven. The steps followed in preparing the Neuron Chip to execute the application code are discussed below. These steps, and their effects on the Neuron Chip pins, are summarized in Figure 4-16.



After the $\overline{\text{RESET}}$ pin is released, the Neuron Chip performs hardware and firmware initialization before executing application programs. These tasks are:

- Stack initialization and built-in self-test (BIST)
- $\overline{\text{SERVICE}}$ pin initialization
- State initialization
- Off-chip RAM initialization
- Random number seed calculation
- System RAM setup
- Communication port initialization
- Checksum initialization
- One-second timer initialization
- Scheduler initialization

During oscillator start up (after power up), the Neuron Chip allows the oscillator enough time to create a signal swing of greater than approximately 1.7 V. This duration depends on the type of oscillator used and its frequency. This period begins as soon as power is applied to the oscillator and is independent of the $\overline{\text{RESET}}$ pin. The oscillator start-up period may end before or after $\overline{\text{RESET}}$ is released, depending on the duration of reset and the time required by the oscillator to start up.

After the oscillator has started up, the Neuron Chip waits 21 transitions on CLK1 to allow the oscillator's frequency to stabilize. From the time reset is asserted until the end of the oscillator stabilization period, the I/O pins, communications port pins, and $\overline{\text{SERVICE}}$ pin are in a high-impedance state. The $\overline{\text{E}}$ signal goes inactive (high) immediately after reset goes low, and the address bus becomes high (0xFFFF) to deselect external devices. Note that pulling the $\overline{\text{RESET}}$ pin low externally when $\overline{\text{E}}$ is low, could result in the $\overline{\text{E}}$ signal going high prematurely. For external devices that depend on a full low duration of the $\overline{\text{E}}$ signal, the external reset signal should be synchronized with the rising edge of $\overline{\text{E}}$.

The stack initialization and BIST task tests the on-chip RAM, the timer/counter logic, and the counter logic. For the test to pass, all three processors and the ROM must be functioning as well. A flag is set to indicate whether the Neuron Chip passed or failed the BIST. The RAM is cleared to all 0s by the end of this step. At the beginning of this task, the pull-ups on IO[7:4] are enabled, so that a weak high state can be observed on these pins. The $\overline{\text{SERVICE}}$ pin oscillates between a solid low and a weak high. The memory interface signals reflect execution of these tasks.

If the RAM self-test fails, the node goes offline, the service LED comes on solid, and an error is logged in the node status structure (see the *query status* message description in Section B.2).

Self-test results are available in the first byte of RAM as follows:

Value	Description
1	RAM failure
2	Timer/counter failure
3	Counter failure

The $\overline{\text{SERVICE}}$ pin initialization task simply turns off the $\overline{\text{SERVICE}}$ pin.

The state initialization task determines if a Neuron Chip boot is required, and, if so, performs it. The Neuron Chip decides to perform a boot if it is blank, or if the boot ID does not match the boot ID in ROM (MC143150

only). The Neuron Chip is assumed to be blank (i.e., the EEPROM is blank) if address 0xF02D is 0x00 or 0xFF; the manufacturer sets this address to the appropriate value during manufacture.

The boot ID consists of 2 bytes starting at 0xF1FE which are set to the boot ID in ROM during the boot process. The user can force a boot process by clearing these 2 bytes and then resetting the Neuron Chip.

The boot process varies as a function of the Neuron Chip model. For the MC143120, it simply copies the default communication parameters and mode table into EEPROM from ROM. For the MC143150, it copies a variable amount of data from ROM to EEPROM. The amount of data varies as a function of the target state of the node. If the node is to come up applicationless, the boot process is the same as that for the MC143120. To come up in the configured or unconfigured state, the boot process must also copy the configuration and code areas of on-chip EEPROM; this can vary from 64 to 504 bytes.

The off-chip RAM initialization task checks the memory map to determine if any off-chip RAM is present and then either tests and clears all of the off-chip RAM or, optionally, only clears the application RAM area. This choice is controlled by the application program via a C pragma statement. This task applies only to the MC143150.

The random number seed calculation task creates a seed for the random number generator.

The system RAM setup task sets up internal system pointers as well as the linked lists of system buffers.

The communication port initialization task initializes the communication port according to the application specified communication port parameters and the MAC processor begins handling packets. For special-purpose mode, the configuration registers are initialized.

The checksum initialization task generates or checks the checksums of the nonvolatile writable memories. If the boot process was executed for the configured or unconfigured states, in the state initialization task, then the checksums are generated; otherwise, they are checked. This process includes on-chip EEPROM, off-chip EEPROM, and off-chip nonvolatile RAM. ROM is not checksummed. There are two checksums, one for the configuration image and one for the application image. In each case, the checksum is a negated two's complement sum of the values in the image. See Appendix A for more details on the configuration and application image. See Section B.1.5, under Checksum Recalculate, for more information on checksums.

The one-second timer initialization task initializes the one-second timer. At this point, the network processor is available to accept incoming packets.

The scheduler initialization task allows the application processor to perform application-related initialization as follows:

- State wait — wait for the node to leave the applicationless state.
- Pointer initialization — perform a global pointer initialization.
- Initialization step — execute initialization task, which is created by the compiler/linker to handle initialization of static variables and the timer/counters.
- I/O pin initialization step — initialize IO pins based on application definition. Prior to this point, IO pins are high impedance.
- State wait II — wait for the node to leave the unconfigured or hard-offline state. If waiting was required, a flag is set to indicate that the node should come up offline.
- Parallel I/O synchronization — nodes using parallel I/O attempt to execute the master/slave synchronization protocol at this point.
- Reset task — execute the application reset task (when (reset{))).

- If the offline flag was set, go offline and execute the offline task. If the BIST flag indicated a failure, then the SERVICE pin is turned on and the offline task is executed. Otherwise, the scheduler starts its normal task scheduling loop.

The amount of time required to perform these steps depends on many factors, including: Neuron Chip model; input clock rate; whether or not the node performs a boot process; whether the node is applicationless, configured, or unconfigured; amount of off-chip RAM; whether the off-chip RAM is tested or simply cleared; the number of buffers allocated; and application initialization. Tables 4-12 and 4-13 summarize the number of input clock cycles (CLK1) required for each of these steps for the MC143120 and the MC143150. The times are approximate and are given as functions of the most significant application variables.

Table 4-12. Time Required for MC143120 to Perform Reset Sequence

Step	Number of CLK1 Cycles	Notes
Stack Initialization and BIST	200,000	
<u>SERVICE</u> Pin Initialization	1000	
State Initialization	250 (for no boot) 2,275,000 (for boot)	
Off-Chip RAM Initialization	0	
Random Number Seed Calculation	0	1
System RAM Setup	$21,000 + 600 \cdot B$	2
Communication Port Initialization	0	1
Checksum Initialization	$3400 + 175 \cdot M$	3
One-Second Timer Initialization	6100	
Scheduler Initialization	≥ 7400	4

NOTES:

1. These tasks run in parallel with other tasks.
2. B is the number of buffers allocated.
3. M is the number of bytes to be checksummed.
4. Assumes a trivial initialization task, no reset task, and the configured state.

For example, the timing of each of these steps is shown for a MC143120 application with the following parameters: 10 MHz input clock, crystal oscillator, no boot required, minimum number of application buffers (10), and 500 bytes of EEPROM checksummed.

Stack Initialization and BIST	20 ms
<u>SERVICE</u> Pin Initialization	0.1 ms
State Initialization	0.025 ms
Off-Chip RAM Initialization	0 ms
Random Number Seed Calculation	0 ms
System RAM Setup	2.7 ms
Communication Port Initialization	0 ms
Checksum Initialization	10.8 ms
One-Second Timer Initialization	0.61 ms
Scheduler Initialization	<u>0.74 ms</u>
Total	35.1 ms

Table 4-13. Time Required for MC143150 to Perform Reset Sequence

Step	Number of CLK1 Cycles	Notes
Stack Initialization and BIST	425,000	
SERVICE Pin Initialization	1000	
State Initialization	1300 (for no boot) 70,000 + 25 ms*E (for boot)	1
Off-Chip RAM Initialization	24,000 + 214*R (for test and clear) 24,000 + 152*R _a (for clear only)	2 3
Random Number Seed Calculation	50,000 max	
System RAM Setup	27,000 + 1500*B	4
Communication Port Initialization	0	5
Checksum Initialization	7200 + 175*M (for no boot) 82,000 + 100 ms + 175*M (for boot)	6 6, 7
One-Second Timer Initialization	6100	
Scheduler Initialization	≥ 7400	8

NOTES:

1. E is the number of non-zero bytes being written (ranges from 10 to 504).
2. R is the number of off-chip RAM bytes.
3. R_a is the number of non-system off-chip RAM bytes.
4. B is the number of buffers allocated.
5. These tasks run in parallel with other tasks.
6. M is the number of bytes to be checksummed.
7. Only if booting to the configured or unconfigured state; if booting to the applicationless state, use the "no boot" equation.
8. Assumes a trivial initialization task, no reset task, and the configured state.

For example, the timing of each of these steps is shown for a Neuron 3150 Chip application with the following parameters: 10 MHz input clock, crystal oscillator, no boot required, 16K external RAM, test and clear external RAM, minimum number of application buffers (10), and 500 bytes of EEPROM checksummed.

Stack Initialization and BIST	42.5 ms
SERVICE Pin Initialization	0.1 ms
State Initialization	0.13 ms
Off-Chip RAM Initialization	353 ms
Random Number Seed Calculation	5 ms
System RAM Setup	4.2 ms
Communication Port Initialization	0 ms
Checksum Initialization	12.5 ms
One-Second Timer Initialization	0.61 ms
Scheduler Initialization	<u>0.74 ms</u>
Total	418 ms

Use the following pragma to disable initializing off-chip RAM:

```
# pragma ram_test_off
```

4.6.4.1 EFFECTS OF `pragma enable_io_pullups` DURING POWER UP AND RESET CONDITIONS.

Refer to Figures 4-17a and 4-17b for the following discussion.

1. During power up, the state of the IO4 – IO7 pull-up resistors are indeterminate until the $\overline{\text{RESET}}$ pin rises to a valid high level. Then, IO4 – IO7 are enabled after the rising edge of the $\overline{\text{RESET}}$ pin + 15 clock cycles and stay enabled during the stack initialization BIST time interval (approximately 40 ms for a 3150 and approximately 20 ms for a 3120.) The pull-ups are then disabled until the I/O initializations step occurs in the scheduler initialization task.

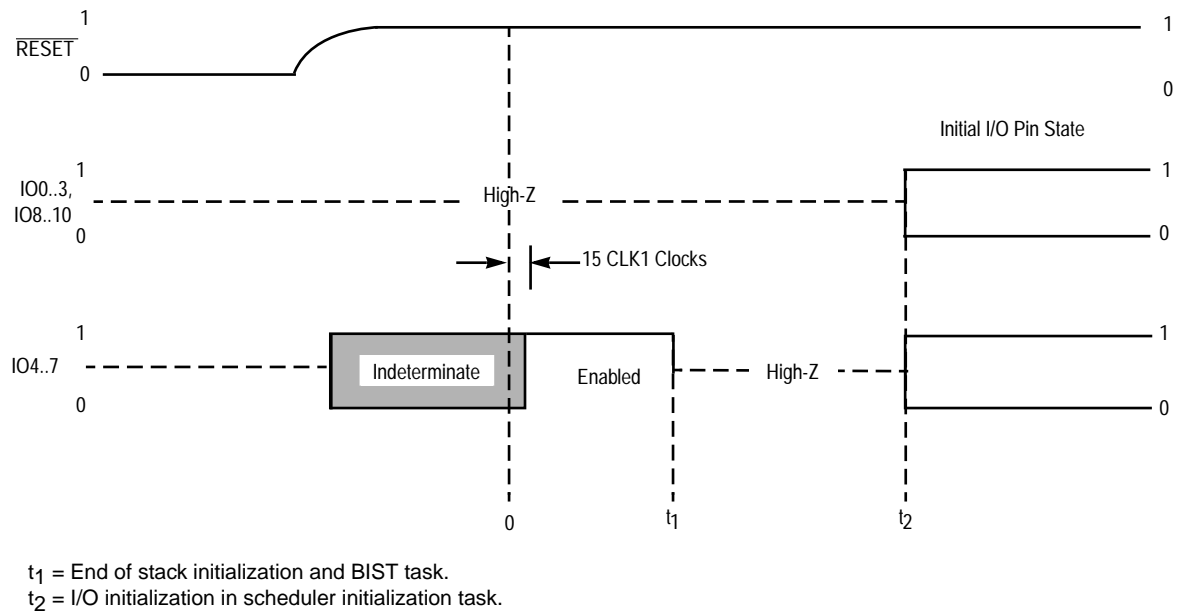


Figure 4-17a. Power Up

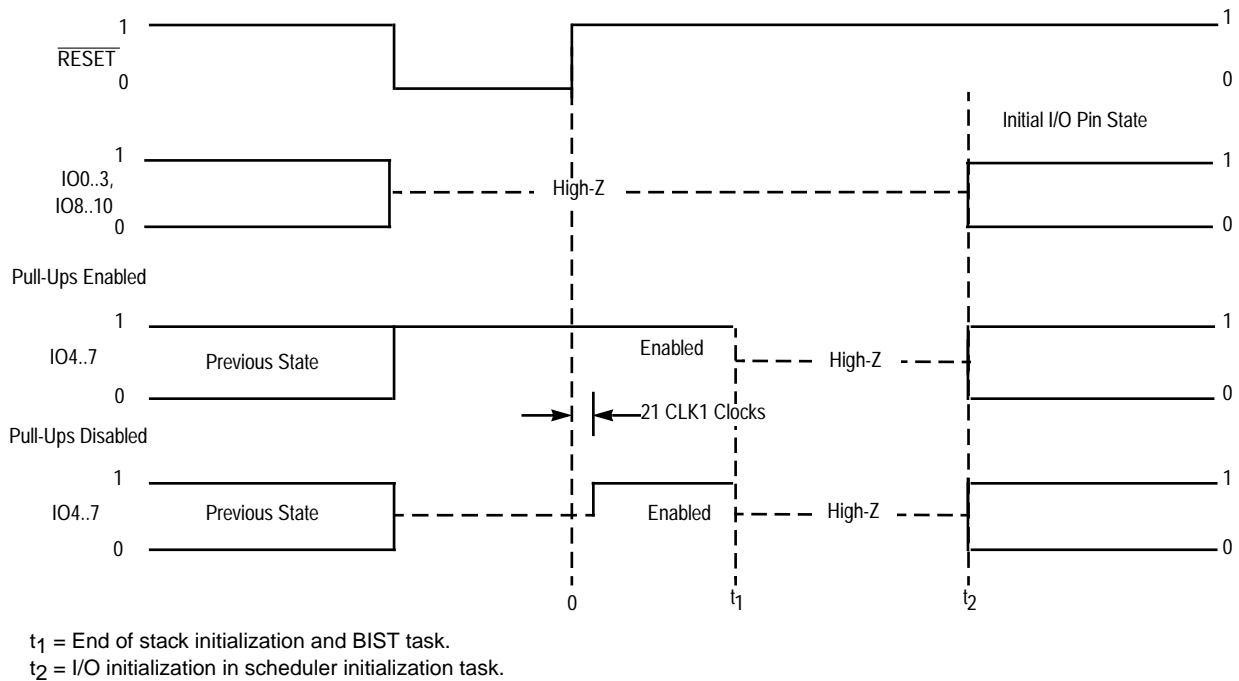


Figure 4-17b. Reset After Power Up

2. Any reset after power up will cause the following conditions on the I/O pins:
 - If pull-ups are enabled, they will stay in that state during the reset condition, stack initialization, and BIST task. Any I/O pins that are programmed as outputs are reverted back to input pins on the falling edge of reset. Since the pull-ups were previously enabled, the I/O pin is then pulled up the V_{DD} level. The pull-up resistors are enabled during the stack initialization and BIST task. Then, the pull-ups are disabled until the I/O initialization step occurs in the scheduler initialization task. Additionally, the I/O pin state is set in this step.
 - If pull-ups are disabled, the I/O pins will revert to a High-Z state during the reset low interval. Any I/O pins that are programmed as outputs are reverted back to input pins on the falling edge of reset. During the state initialization and BIST task, pull-ups are enabled. The I/O pins stay disabled until the I/O initialization step occurs in the scheduler initialization task. Additionally, the I/O pin state is set in this step.

Since the pull-ups are always enabled during the stack initialization and BIST task, this can cause a potential problem in applications where an NPN Darlington transistor configuration is used for a relay driver. One possible solution is to pick a suitable value of a pull-down resistor that will negate the effect of the pull-up being enabled during the initialization sequence. Values of the pull down typically range from 2.4 k Ω to 2.7 k Ω .

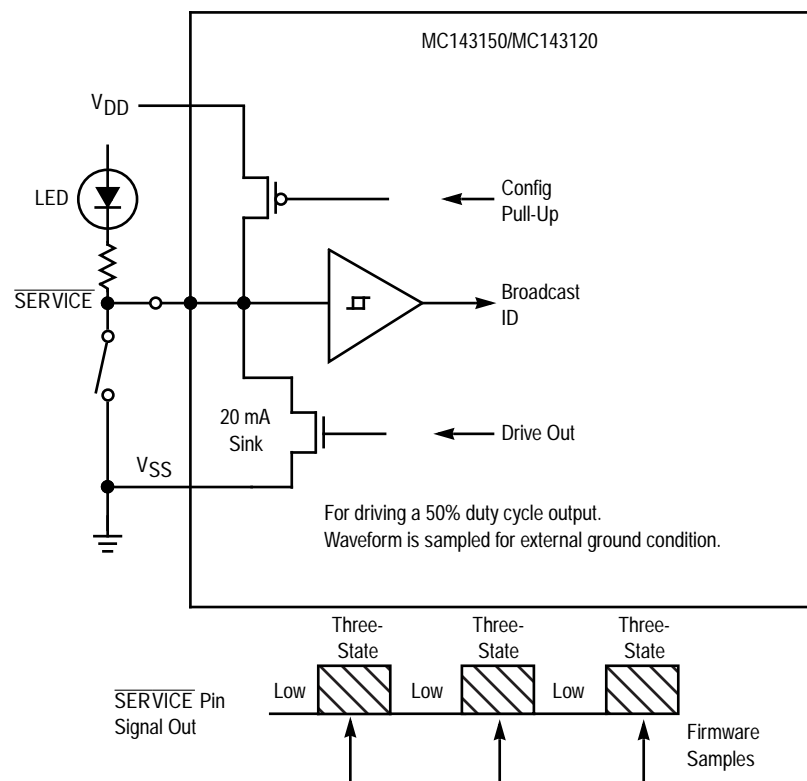
4.6.4.2 RESET FROM CHECKSUM ERROR. If a checksum error occurs, the Neuron Chip will execute a software reset. See Section 3.2.6, and Figure E-2 in Appendix E for further information.

4.7 SERVICE PIN

The SERVICE pin is executed differently depending on which firmware release is implemented in the Neuron IC. This variance should be considered by the user in the proposed system design. Presently there are two variations in how the Neuron IC responds to the SERVICE pin, as discussed below. This pin alternates between input and open-drain output at a 76 Hz rate with a 50% duty cycle. When it is an output, it can sink 20 mA for use in driving a LED. When it is used exclusively as an input, it has an optional on-chip pull-up to bring the input to an inactive-high state for use when the LED and pull-up resistor are not connected. Under control of Neuron Chip firmware, this pin is used during configuration, installation, and maintenance of the node containing the Neuron Chip. The firmware flashes the LED at a 1/2-Hz rate when the Neuron Chip has not been configured with network address information. Grounding the SERVICE pin causes the Neuron Chip to transmit a network management message containing its unique 48-bit ID on the network. This information may then be used by a network management device to install and configure the node. A typical circuit for the SERVICE pin LED and push-button is shown in Figure 4-18. During reset the SERVICE pin state is indeterminate. The default state of the SERVICE pin pull-up is enabled.

A 2.4 kHz (10 MHz clock) rate on the SERVICE pin occurs when a 3150 Neuron Chip is not executing proper read cycles from external memory. During the reset initialization (BIST), the SERVICE pin counter is set to output 76 Hz. Until the 76 Hz is programmed, the SERVICE pin default frequency is 2.4 kHz.

The SERVICE pin state is checked in the network processor main loop by the firmware. This check will occur once following each message sent or received, approximately three times per millisecond when idle at 10 MHz. This processing can occur even when there are priority or non-priority messages queued up. Once an application or network buffer is freed, another application message may take the buffer before the SERVICE pin message is serviced. Hence, the SERVICE pin message does not take priority over an application message.

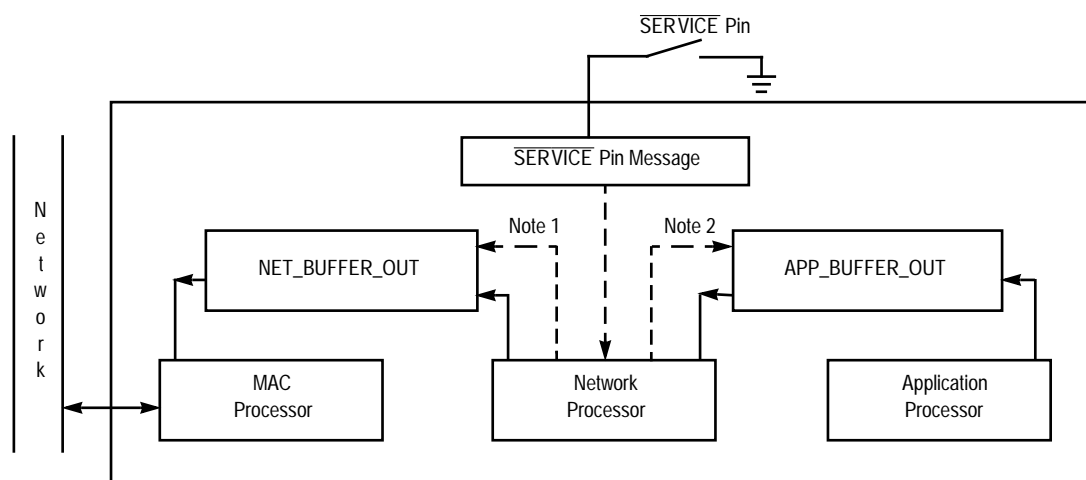


Node State	hF01F State Code	Service LED	Pulse Rate (Hz)
Applicationless and Unconfigured	3	On	76
Unconfigured (but with an Application)	2	Flashing	0.51
Configured, Hard Offline	6	Off	—
Configured	4	Off	—
3150 Defective External Memory	—	On	2400

Figure 4-18. SERVICE Pin Circuit

The $\overline{\text{SERVICE}}$ pin is active low and is sent once maximum, per $\overline{\text{SERVICE}}$ pin transition. The $\overline{\text{SERVICE}}$ pin message may go into the output network buffer or output application buffer, depending on which Neuron Chip and firmware version number is in use. The output network or application buffer is always a non-priority type buffer. The $\overline{\text{SERVICE}}$ pin message will go into the application buffer except for the 3150 V6 or later. The 3150 V6 (or later) $\overline{\text{SERVICE}}$ pin message is placed in a network buffer.

For more information regarding the behavior of the $\overline{\text{SERVICE}}$ pin LED for debugging of the Neuron Chip, refer to EB172, *Custom Node Development*, in the AL section of this data book.



NOTES:

1. MC143150 (V6).
2. MC143150 (V4), MC143120 (V3, V4, V6).

Figure 4-19. Buffers $\overline{\text{SERVICE}}$ Pin Message Written

Starting with the 3120 V4 (version 4 firmware), 3120E1 V6, 3120E2 V6, and 3150 V6; if the $\overline{\text{SERVICE}}$ pin is grounded, the $\overline{\text{SERVICE}}$ pin message will go out when there is an available buffer. As stated previously, the $\overline{\text{SERVICE}}$ pin message does not take priority over an application buffer. For the 3120 V3 and 3150 V4 firmware versions, the $\overline{\text{SERVICE}}$ pin message has only one chance for an application buffer to be available for the service message to be sent out. The 3150 V6 has the feature that once the $\overline{\text{SERVICE}}$ pin is grounded, it is logged until the $\overline{\text{SERVICE}}$ pin message goes out, even if the $\overline{\text{SERVICE}}$ pin goes back high.

The following tables summarize the above information.

MC143150

	MC143150 (V6)	MC143150 (V4)
When $\overline{\text{SERVICE}}$ pin is read	Latched when low	Polled once during a low state
Buffer $\overline{\text{SERVICE}}$ pin message is placed	Network buffer	Application buffer

MC143120

	MC143120 (V4), MC143120E1 (V6), MC143120E2 (V6)	MC143120 (V3)
When $\overline{\text{SERVICE}}$ pin is read	Continuous polling during a low state	Polled once during a low state
Buffer $\overline{\text{SERVICE}}$ pin message is placed	Application buffer	Application buffer

Input/Output Interfaces **5**

SECTION 5 INPUT/OUTPUT INTERFACES

The Neuron Chip connects to application-specific external hardware via 11 pins, named IO0 – IO10. These pins may be configured in numerous ways to provide flexible input and output functions with a minimum of external circuitry. The programming model (Neuron C language) allows the programmer to declare one or more pins as I/O objects. An object is simply an input or output waveform definition. They can be thought of as prewritten firmware routines in ROM which are accessed by the user's application program. The user's program may then refer to these objects in *io_in* and *io_out* system calls, to perform the actual input/output function during execution of the program. There are 34 different I/O objects available in the MC143150 system image. Most are available in the MC143120 internal ROM image. The additional objects can be loaded into the MC143120 EEPROM.

The Neuron Chip has two 16-bit timer/counters on-chip (see Figure 3-8). The input to timer/counter 1, also called the *multiplexed timer/counter*, is selectable among pins IO4 – IO7, via a programmable multiplexer (mux) and its output may be connected to pin IO0. The input to timer/counter 2, also called the *dedicated timer/counter*, may be connected to pin IO4 and its output to pin IO1. The timer/counters are implemented as a 16-bit load register writable by the CPU, a 16-bit counter, and a 16-bit latch readable by the CPU. The load register and latch are accessed a byte at a time. Note that no I/O pins are dedicated to timer/counter functions. If, for example, timer/counter 1 is used for input signals only, then IO0 is available for other input or output functions. Timer/counter clock and enable inputs may be from external pins, or from scaled clocks derived from the system clock; the clock rates of the two timer/counters are independent of each other. External clock actions occur optionally on the rising edge, the falling edge, or both rising and falling edges of the input.

Note that multiple timer/counter input objects may be declared on different pins within a single application. By calling *io_select*, the application can use the first timer/counter to implement up to four different input objects. If a timer/counter is configured to implement one of the output objects, or is configured as a quadrature input object, then it can not be reassigned to another timer/counter object in the same application program.

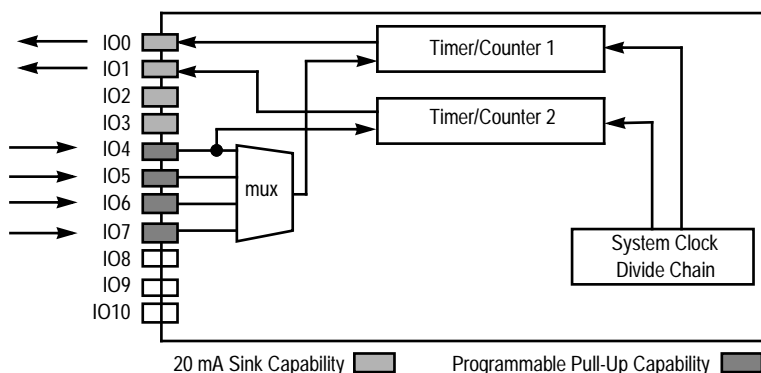


Figure 5-1. Neuron Chip Timer/Counter External Connections

The Neuron C model allows the programmer to declare one or more pins as I/O objects. The user's program may then refer to these objects in *io_in ()* and *io_out ()* system calls to perform the actual input/output operations during execution of the program. Certain events are associated with changes in input values. The task scheduler can thus execute associated application code when these changes occur.

5.1 HARDWARE CONSIDERATIONS

Tables 5-1 through 5-5 list 34 different I/O objects available. Various I/O objects of different types may be used simultaneously. Figure 5-2 summarizes the pin configuration for each of the I/O objects. For the electrical characteristics of these pins, refer to Section 6. The following sections contain detailed descriptions of all the I/O objects. The application program may optionally specify the initial values of digital outputs. Pins configured as outputs may also be read as inputs, returning the value last written. Pins IO4 – IO7 have optional pull-up current sources (60 μ A to 260 μ A) which act like pull-up resistors. These are enabled with a Neuron C compiler directive (`#pragma enable_io_pullups`). Pins IO0 – IO3 have high sink capability (20 mA @ 0.8 V). The others have the standard sink capability (1.4 mA @ 0.4 V). Pins IO0 – IO7 have low-level detect latches. The latency and timing values described in this section are typical at 10 MHz. The accuracy is $\pm 10\%$. Most latency values scale up at lower input clock rates.

Table 5-1. Summary of Direct I/O Objects

I/O Object	Applicable I/O Pins	Input/Output Value	Loaded Into 1: Masked ROM 2: EEPROM (3120B1)	Loaded Into 1: Masked ROM 2: EEPROM (3120E2/FE2/LE2)	Page No.
Bit Input	IO0 – IO10	0, 1 binary data	1	1	5-10
Bit Output	IO0 – IO10	0, 1 binary data	1	1	5-10
Byte Input	IO0..IO7	0 – 255 binary data	1	1	5-12
Byte Output	IO0..IO7	0 – 255 binary data	1	1	5-12
Leveldetect Input	IO0 – IO7	Logic 0 level detected	1	1	5-13
Nibble Input	Any adjacent 4 in IO0 – IO7	0 – 15 binary data	1	1	5-13
Nibble Output	Any adjacent 4 in IO0 – IO7	0 – 15 binary data	1	1	5-13

Table 5-2. Summary of Parallel I/O Objects

I/O Object	Applicable I/O Pins	Input/Output Value	Loaded Into 1: Masked ROM 2: EEPROM (3120B1)	Loaded Into 1: Masked ROM 2: EEPROM (3120E2/FE2/LE2)	Page No.
Muxbus I/O	IO0..IO10	Parallel bidirectional port using multiplexed address technique	2	2	5-15
Parallel I/O	IO0..IO10	Parallel bidirectional handshaking port	1	1	5-15

Table 5-3. Summary of Serial I/O Objects

I/O Object	Applicable I/O Pins	Input/Output Value	Loaded Into 1: Masked ROM 2: EEPROM (3120B1)	Loaded Into 1: Masked ROM 2: EEPROM (3120E2/FE2/LE2)	Page No.
Bitshift Input	Any adjacent pair (except IO7 + IO8)	Up to 16 bits of clocked data	1	1	5-24
Bitshift Output	Any adjacent pair (except IO7 + IO8)	Up to 16 bits of clocked data	1	2	5-24
I ² C	IO8 + IO9	Up to 255 bytes of bidirectional serial data	2	2	5-26
Magcard Input	IO8 + IO9 + IO0 – IO7	Encoded ISO7811 track 2 data stream from a magnetic card reader	2	2	5-28
Magtrack1	IO8 + IO9 + IO0 – IO7	Encoded ISO3554 track 1 data stream from a magnetic card reader	2	2	5-29
Neurowire I/O	IO8 + IO9 + IO10 + IO0 – IO7	Up to 256 bits of bidirectional serial data	1: Master Mode 2: Slave Mode	1: Master Mode 2: Slave Mode	5-30
Serial Input	IO8	8-bit characters at 600, 1200, 2400, or 4800 bps	1	2	5-33
Serial Output	IO10	8-bit characters at 600, 1200, 2400, or 4800 bps	1	2	5-33
Touch I/O	IO0 – IO7	Up to 2048 bits of input or output bits	2	2	5-34
Wiegand Input	Any adjacent pair in IO0 – IO7	Encoded data stream from Wiegand card reader	2	2	5-36

Table 5-4. Summary of Timer/Counter Input Objects

I/O Object	Applicable I/O Pins	Input Signal	Loaded Into 1: Masked ROM 2: EEPROM (3120B1)	Loaded Into 1: Masked ROM 2: EEPROM (3120E2/FE2/LE2)	Page No.
Dualslope Input	IO0, IO1 + IO4 – IO7	Comparator output of the dualslope converter logic	2	2	5-39
Edgelog Input	IO4	A stream of input transitions	2	2	5-40
Infrared Input	IO4 – IO7	Encoded data stream from an infrared demodulator	2	2	5-41
Ontime Input	IO4 – IO7	Pulse width of 0.2 μ s – 1.678 s	1	1	5-42
Period Input	IO4 – IO7	Signal period of 0.2 μ s – 1.678 s	1	1	5-43
Pulsecount Input	IO4 – IO7	0 – 65,535 input edges during 0.839 s	1	1	5-44
Quadrature Input	IO4 + IO5, IO6 + IO7	\pm 16,383 binary Gray code transitions	1	1	5-45
Totalcount Input	IO4 – IO7	0 – 65,535 input edges	1	1	5-46

Table 5-5. Summary of Timer/Counter Output Objects

I/O Object	Applicable I/O Pins	Output Signal	Loaded Into 1: Masked ROM 2: EEPROM (3120B1)	Loaded Into 1: Masked ROM 2: EEPROM (3120E2/FE2/LE2)	Page No.
Edgedivide Output	IO0, IO1 + IO4 – IO7	Output frequency is the input frequency divided by a user-specified number	1	1	5-47
Frequency Output	IO0, IO1	Square wave of 0.3 Hz to 2.5 MHz	1	1	5-48
Oneshot Output	IO0, IO1	Pulse of duration 0.2 μ s to 1.678 s	1	1	5-49
Pulsecount Output	IO0, IO1	0 – 65,535 pulses	1	1	5-50
Pulsewidth Output	IO0, IO1	0 – 100% duty cycle pulse train	1	1	5-51
Triac Output	IO0, IO1 + IO4 – IO7	Delay of output pulse with respect to input edge	1	1	5-52
Triggered- count Output	IO0, IO1 + IO4 – IO7	Output pulse controlled by counting input edges	1	1	5-54

To maintain and provide consistent behavior for external events and to prevent metastability, all 11 I/O pins of the Neuron Chip, when configured as inputs, are passed through a hardware synchronization block sampled by the internal system clock which is always the input clock divided by two (10 MHz \div 2 = 5 MHz). For any signal to be reliably synchronized, it must be at least 220 ns in duration (see Figure 5-3).

All inputs are software sampled during *when* statement processing. The latency in sampling is dependent on the I/O object which is being executed (see I/O timing specification and *Neuron C Programmer's Guide* for more information). These latency values scale up proportionally at lower clock rates. Thus, any event that lasts longer than 220 ns will be synchronized by hardware, but there will be latency in software sampling resulting in a delay detecting the event. If the state changes at a faster rate than software sampling can occur, then the interim changes will go undetected.

There are two exceptions to the synchronization block. One, the chip select (\overline{CS}) input used in the slave B mode of the parallel I/O object; this input will recognize rising edges asynchronously (see Section 5.4.3). Two, the leveldetector input is latched by a flip flop with a 200 ns clock. The leveldetector transition event will be latched, but there will be a delay in software detection (see Section 5.3.3). The input timer/counter functions are also different, in that events on the I/O pins will be accurately measured and a value returned to a register, regardless of the state of the application processor. However, the application processor may be delayed in reading the register. Consult the *Neuron C Programmer's Guide* for detailed programming information.

		I/O Pin	0	1	2	3	4	5	6	7	8	9	10
DIRECT I/O MODES	Bit Input, Bit Output												
	Byte Input, Byte Output		All Pins 0 – 7										
	Leveldetect Input												
	Nibble Input, Nibble Output		Any Four Adjacent Pins										
PARALLEL I/O MODES	Muxbus I/O		Data Pins 0 – 7								ALS	WS	RS
	Parallel I/O {	Master/Slave A	Data Pins 0 – 7								\overline{CS}	R/W	HS
		Slave B	Data Pins 0 – 7								\overline{CS}	R/W	A0
	Bitshift Input, Bitshift Output		C	D	C	D	C	D	C	D	C	D	C
SERIAL I/O MODES	I ² C I/O										C	D	
	Magcard Input		Optional Timeout								C	D	
	Magtrack1 Input		Optional Timeout								C	D	
	Neurowire I/O {	Master	Optional Chip Select								C	D	D
		Slave	Optional Timeout								C	D	D
	Serial Input												
	Serial Output												
TIMER/COUNTER INPUT MODES	Wiegand Input		Any Two Pins (Optional Timeout)										
	Dualslope Input		Control										
	Edgelog Input												
	Infrared Input												
	Ontime Input												
	Period Input												
	Pulsecount Input												
	Quadrature Input					4 + 5	6 + 7						
	Totalcount Input												
	Edgedivide Output					Sync Input							
TIMER/COUNTER OUTPUT MODES	Frequency Output												
	Oneshot Output												
	Pulsecount Output												
	Pulsewidth Output												
	Triac Output		Control			Sync Input							
	Triggeredcount Output		Control			Sync Input							
			0	1	2	3	4	5	6	7	8	9	10
			High Sink			Pull Ups			Standard				

Bitshift, I²C, Magcard, Magtrack, Neurowire: C = Clock, D = Data

Timer/Counter 1 Devices:

either: IO_6 input quadrature
or: IO_4 input edgelog
or: IO_0 output [triac | triggeredcount | edgedivide] sync ([IO_4...IO7])
or: IO_0 output [frequency | oneshot | pulsecount | pulsewidth]
or: up to four of:
IO_4 input [ontime | period | pulsecount | totalcount | dualslope | infrared] mux
[IO_5...IO_7] input [ontime | period | pulsecount | totalcount | dualslope | infrared]

Timer/Counter 2 Devices:

either: IO_4 input quadrature
or: IO_1 output [triac | triggeredcount | edgedivide] sync(IO_4)
or: IO_1 output [frequency | oneshot | pulsecount | pulsewidth]
or: IO_4 input edgelog
or: IO_4 input [ontime | period | pulsecount | totalcount | dualslope | infrared] ded

Figure 5-2. Summary of I/O Objects

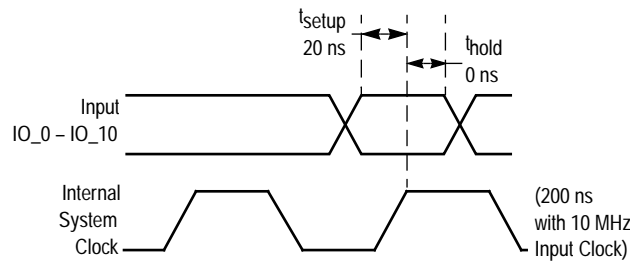


Figure 5-3. Synchronization of External Signals

5.2 I/O TIMING ISSUES

The Neuron Chip I/O timing is influenced by three separate, yet overlapping areas of the overall chip architecture:

- The scheduler
- The I/O function block's firmware
- The Neuron Chip hardware

The contribution of the scheduler to the overall timing characteristic is approximately uniform across all 34 I/O function blocks since its contribution to the overall I/O timing is at a relatively high functional level.

The contribution of firmware and hardware varies from one I/O function block type to another (e.g., Bit I/O versus Neurowire I/O), with one area generally being the dominant factor.

5.2.1 Scheduler-Related I/O Timing Information

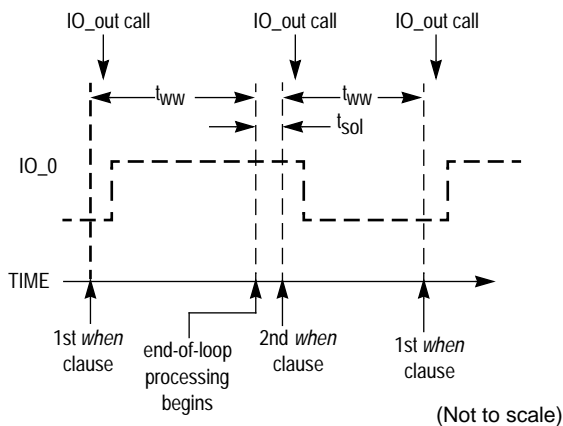
As part of the Neuron Chip firmware, the scheduler provides an orderly and predictable means to facilitate the evaluation of user-defined events. The *when* clause, provided by the Neuron C language, is used to specify such events. For more information on the operation of the scheduler, refer to the *Neuron C Programmer's Guide*.

There is a finite latency associated with the operation of the scheduler. The time required for the scheduler to evaluate the same *when* clause in a particular user application code is, to a large extent, a function of the size of the user code, the total number of *when* clauses, and the state of the events associated with those *when* clauses. It is therefore, impossible to specify a nominal value for this latency, as each application will have its own distinct behavior under different circumstances.

The best case latency can be viewed in several ways, each exposing a different aspect of the scheduler operation. A simple example consists of having an application program consisting of two *when* clauses, both of which always evaluate to TRUE, as shown below.

```
IO_0 output bit testbit;
when (TRUE)
{
  io_out (testbit, 1); }
when (TRUE)
{
  io_out (testbit, 0); }
```

Processing of *when* clauses is done in a round-robin fashion; therefore, the Neuron C code above, performs alternating activation of the IO0 pin in order to isolate and extract the timing parameters associated with the scheduler. The waveform seen on pin IO0 of the Neuron Chip, as a result of the above code, is shown in Figure 5-4.



Symbol	Description	Typ @ 10 MHz
t_{WW}	<i>when</i> -clause to <i>when</i> -clause latency	940 μ s
t_{sol}	Scheduler overhead latency (see text)	54 μ s

Figure 5-4. *when*-Clause to *when*-Clause Latency, t_{WW} and Scheduler Overhead Latency, t_{sol}

Note that the *when*-clause to *when*-clause latency, t_{WW} , in this case, includes the execution time of one `io_out()` function (65 μ s latency) and is for an event that always evaluates to TRUE. The actual t_{WW} for a given application is driven by the actual task within the *when* statement as well as the *when* event which is evaluated.

The above example not only measures the best-case minimum latency between consecutive *when* clauses (whose events evaluate to TRUE), t_{WW} , but also reveals the scheduler's end-of-loop overhead latency, t_{sol} . As shown in Figure 5-4, t_{WW} is the off-time period of the output waveform and t_{sol} is the on-time of the output waveform, minus t_{WW} . This shows that the scheduler overhead latency, or the scheduler end-of-loop latency, occurs just before the execution of the last *when* clause in the program.

The latency associated with the return from the `io_out()` function is small, relative to that of the execution of the function call itself.

NOTE

Some I/O objects suspend application processing until the task is complete. This is because they are firmware-driven. These are Bitshift, Neurowire, Parallel, and Serial I/O objects, I²C, Magcard, Magtrack, Touch I/O, and Wiegand. They do not suspend network communication as this is handled by the network processor and the media access processor.

5.2.2 Firmware and Hardware Related I/O Timing Information

All I/O updates in the Neuron Chip are performed by the firmware using system image function calls.

The total latency for a given function call, from start to end, can be broken down into two separate parts. The first is due to the processing time required before the actual hardware I/O update (read or write) occurs. The second delay is associated with the time required to finish the current function call and return to the application program.

Overall accuracy is always related to the accuracy of the Neuron Chip's CLK1 input. Timing diagrams are provided for all non-trivial cases to clarify the parameters given.

For more information on the operation of each of the I/O objects, usage, and syntax; refer to the *Neuron C Reference Guide*.

5.3 DIRECT OBJECTS (BIT I/O, BYTE I/O, LEVELDETECT, AND NIBBLE)

The timing numbers shown in this section are valid for both an explicit I/O call or an implicit I/O call through a *when* clause, and are assumed to be for a Neuron Chip running at 10 MHz.

5.3.1 Bit I/O

Pins IO0 – IO10 may be individually configured as single-bit input or output ports. Inputs may be used to sense TTL-level compatible logic signals from external logic, contact closures, and the like. Outputs may be used to drive external CMOS and TTL level compatible logic, and switch transistors and very low current relays to actuate higher-current external devices such as stepper motors and lights. The high (20 mA) current sink capability of pins IO0 – IO3 allows these pins to drive many I/O devices directly (refer to Figure 5-5). Figures 5-6 and 5-7 show the bit input and bit output latency times, respectively. These are the times from which IO_in or IO_out is called, until a value is returned. The direction of bit ports may be changed between input and output dynamically under application control.

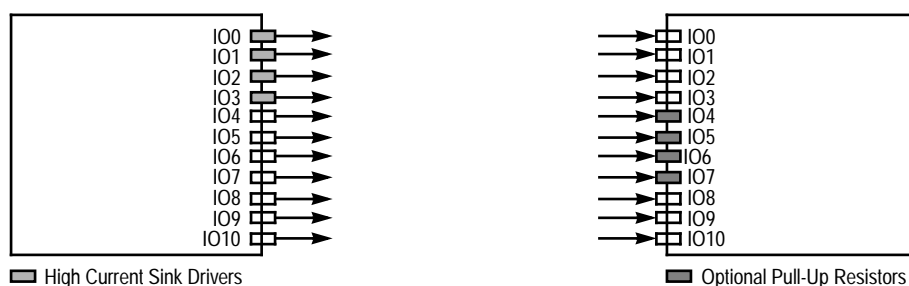
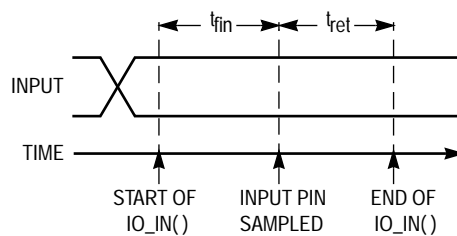


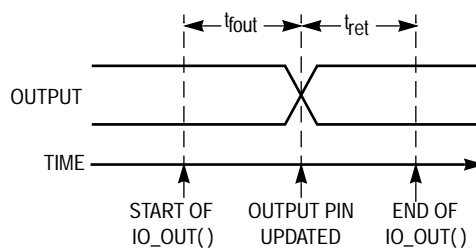
Figure 5-5. Bit I/O

Be aware that after a Reset, the Neuron IC performs a self-test which includes enabling pull-up resistors IO4 – IO7. This could cause a positive level change. See Figure 4-17a.



Symbol	Description	Typ @ 10 MHz
t_{fin}	Function call to sample IO0 – IO10	41 μ s
t_{ret}	Return from function IO0 IO1 IO2 IO3 IO4 IO5 IO6 IO7 IO8 IO9 IO10	19 μ s 23.4 μ s 27.9 μ s 32.3 μ s 36.7 μ s 41.2 μ s 45.6 μ s 50 μ s 19 μ s 23.4 μ s 27.9 μ s

Figure 5-6. Bit Input Latency Values



Symbol	Description	Typ @ 10 MHz
t_{fout}	Function call to update IO3 – IO5 All others	69 μ s 60 μ s
t_{ret}	Return from function IO0 – IO10	5 μ s

Figure 5-7. Bit Output Latency Values

5.3.2 Byte I/O

Pins IO0 – IO7 may be configured as a byte-wide input or output port, which may be read or written using integers in the range 0 to 255. This is useful for driving devices that require ASCII data, or other data, eight bits at a time. For example, an alphanumeric display panel can use byte function for data, and use pins IO8 – IO10 in bit function for control and addressing. See Figures 5-8, 5-9, and 5-10. IO0 represents the LSB of data. The direction of a byte port may be changed between input and output dynamically under application control.

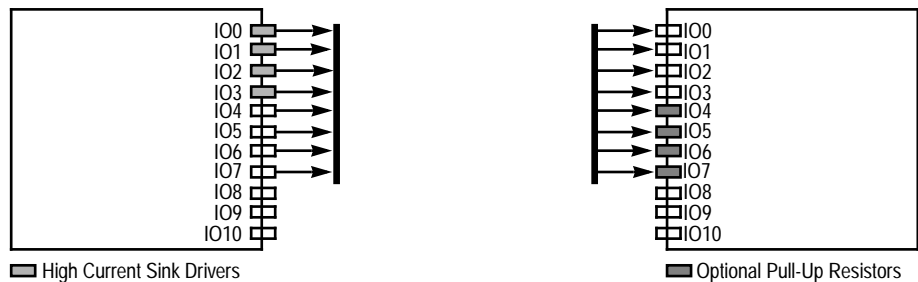
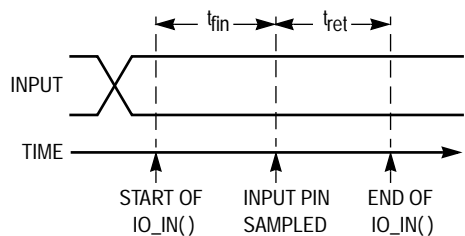
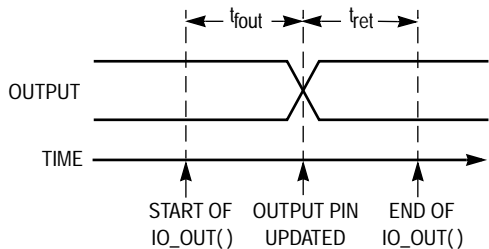


Figure 5-8. Byte I/O



Symbol	Description	Typ @ 10 MHz
t_{fin}	Function call to input sample	24 μ s
t_{ret}	Return from function	4 μ s

Figure 5-9. Byte Input Latency Values

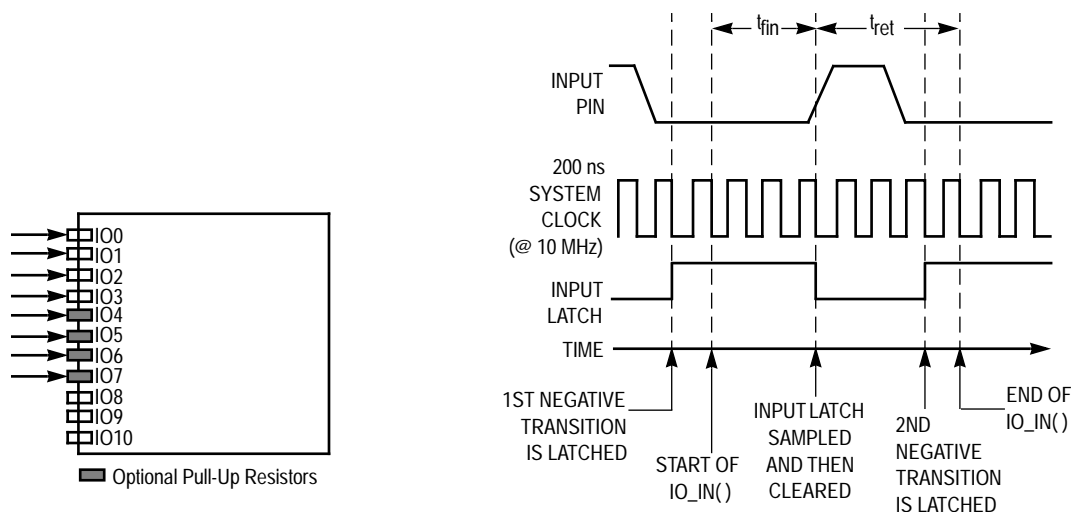


Symbol	Description	Typ @ 10 MHz
t_{fout}	Function call to update	57 μ s
t_{ret}	Return from function	5 μ s

Figure 5-10. Byte Output Latency Values

5.3.3 Leveldetect (Logic Low Level for Input > 200 ns)

Pins IO0 – IO7 may be individually configured as leveldetect input pins, which latch a negative-going transition of the input level with a minimal low pulse width of 200 ns, with a Neuron Chip clocked at 10 MHz. The application can therefore detect short pulses on the input which might be missed by software polling. This is useful for reading devices, such as proximity sensors. **Note that this is the only I/O object which is latched before it is sampled.** The latch is cleared during the *when* statement sampling and can be set again immediately after, if another transition should occur. See Figure 5-11.



Symbol	Description	Typ @ 10 MHz
t_{fin}	Function call to sample IO0 IO1 IO2 IO3 IO4 IO5 IO6 IO7	35 μ s 39.4 μ s 43.9 μ s 48.3 μ s 52.7 μ s 57.2 μ s 61.6 μ s 66 μ s
t_{ret}	Return from function	32 μ s

Figure 5-11. Leveldetect Input Latency Values

5.3.4 Nibble I/O

Groups of four consecutive pins between IO0 – IO7 may be configured as nibble-wide input or output ports, which may be read or written to using integers in the range 0 to 15. This is useful for driving devices that require BCD data, or other data four bits at a time. For example, a 4x4 key switch matrix may be scanned by using one nibble to generate an output (row select — one of four rows), and one nibble to read the input from the columns of the switch matrix. See Figures 5-12, 5-13, and 5-14.

The direction of nibble ports may be changed between input and output dynamically under application control (see the *Neuron C Programmer's Guide*). The LSB of the input data is determined by the object declaration and can be any of the IO0 – IO4 pins.

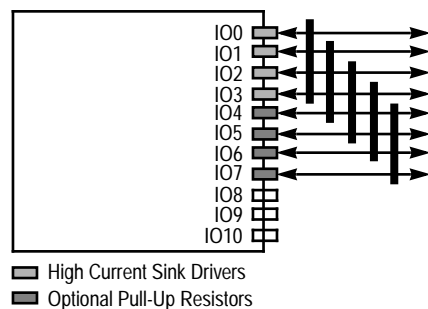
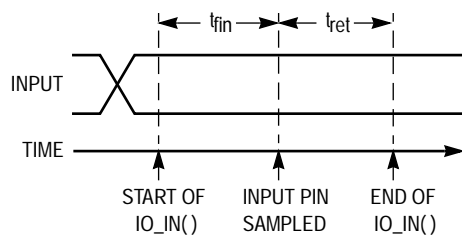
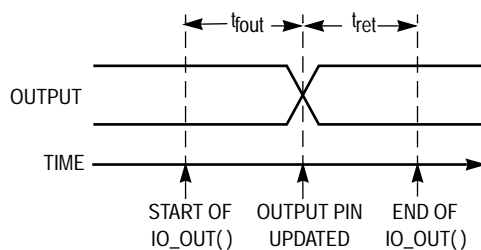


Figure 5-12. Nibble I/O



Symbol	Description	Typ @ 10 MHz
t_{fin}	Function call to sample IO0 – IO4	41 μ s
t_{ret}	Return from function IO0 IO1 IO2 IO3 IO4	18 μ s 22.8 μ s 27.5 μ s 32.3 μ s 37 μ s

Figure 5-13. Nibble Input Latency Values



Symbol	Description	Typ @ 10 MHz
t_{fout}	Function to update IO0 IO1 IO2 IO3 IO4	78 μ s 89.8 μ s 101.5 μ s 113.3 μ s 125 μ s
t_{ret}	Return from function IO0 – IO4	5 μ s

Figure 5-14. Nibble Output Latency Values

5.4 PARALLEL I/O INTERFACE OBJECT

The timing numbers shown in this section are valid for both an explicit I/O call or an implicit I/O call through a *when* clause, and are assumed to be for a Neuron Chip running at 10 MHz.

5.4.1 Introduction

Pins IO0 – IO10 may be configured as a bidirectional 8-bit data and 3-bit control port for connecting to an external processor. The other processor may be a computer, microcontroller, or another Neuron Chip (for bridge, router, gateway, or other applications). The parallel interface may be configured in master, slave A, or slave B mode. Typically, two Neuron Chips interface in master/slave A mode and a Neuron Chip interfaces with a non-Neuron Chip processor in the slave B configuration, with the non-Neuron Chip as the master. Handshaking is used in both modes to control the instruction execution, and application processing is suspended for the duration of the transfer (up to 255 bytes/transfer). Consult the *Neuron C Programmer's Guide* for detailed programming instructions.

Upon a reset condition, the master processor monitors the low transition of the handshake (HS) line from the slave, then passes a CMD_RESYNC (0x5A) for synchronization purposes. This must be done within 0.84 second after reset goes high with a Neuron Chip slave running at 10 MHz, to avoid a watchdog reset error condition (see the *Neuron C Programmer's Guide*). The CMD_RESYNC is followed by the slave acknowledging with a CMD_ACKSYNC (0x07). This synchronization ensures that both processors are properly reset before data transfer occurs. When interfacing two Neuron Chips, these characters are passed automatically (refer to the flow table illustrated later in this section). However, a non-Neuron Chip must duplicate the interface signals and characters that are automatically generated by the parallel I/O function.

For additional information, see Application Note AN1208, *Parallel I/O Interface to the Neuron Chip*.

5.4.2 Master/Slave A Mode

This mode is recommended when interfacing two Neuron Chips. In a master/slave A configuration, the master drives IO8 as a chip select and IO9 to specify a read or write cycle, and the slave drives IO10 as an HS acknowledgment (see Figure 5-15). The maximum data transfer rate is 1 byte per 4 processor instruction cycles, or 2.4 μ s per byte at the maximum input clock rate (10 MHz). The data transfer rate scales proportionally to the input clock rate (note that a master write is a slave read). Timing for the case where the Neuron Chip is the master (Figure 5-16), refers to measured output timing at 10 MHz. After every byte write or byte read, the HS line is monitored by the master, to verify the slave has completed processing (when HS = 0) and the slave is ready for the next byte transfer. This is done automatically in Neuron Chip-to-Neuron Chip (master/slave A mode) data transfers. Slave A timing is shown in Figure 5-17.

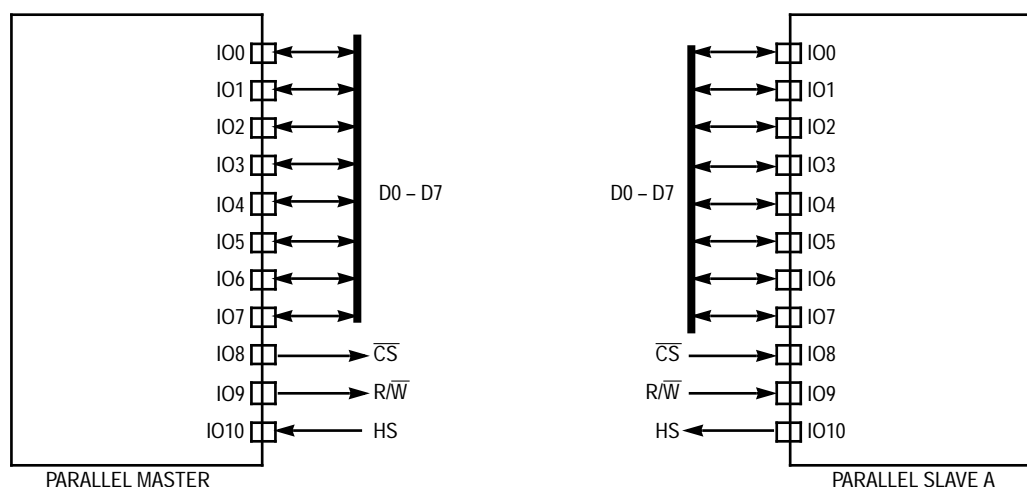
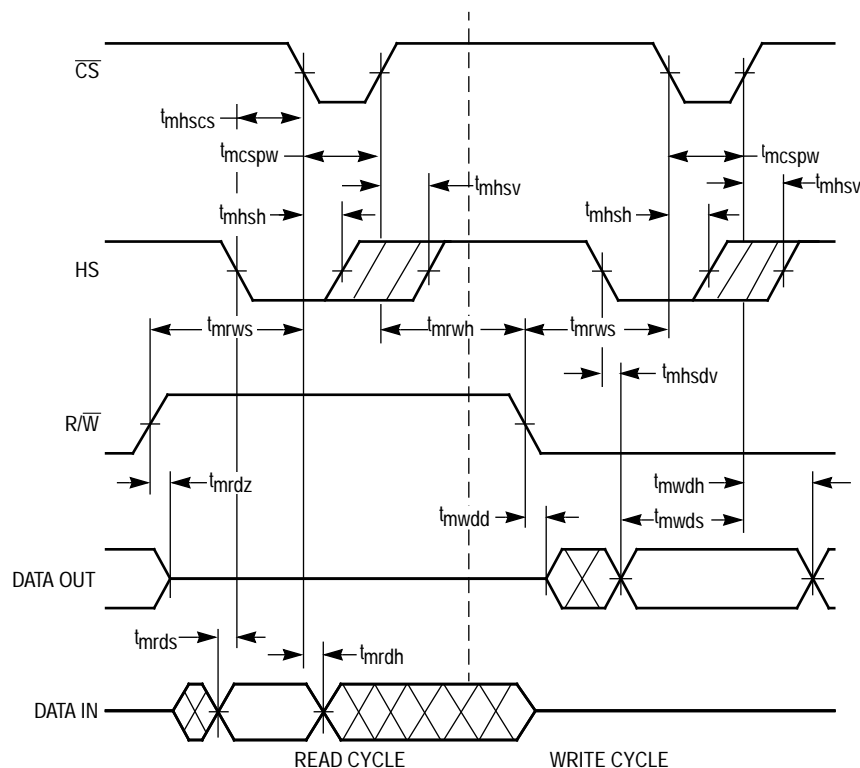


Figure 5-15. Parallel I/O — Master and Slave A

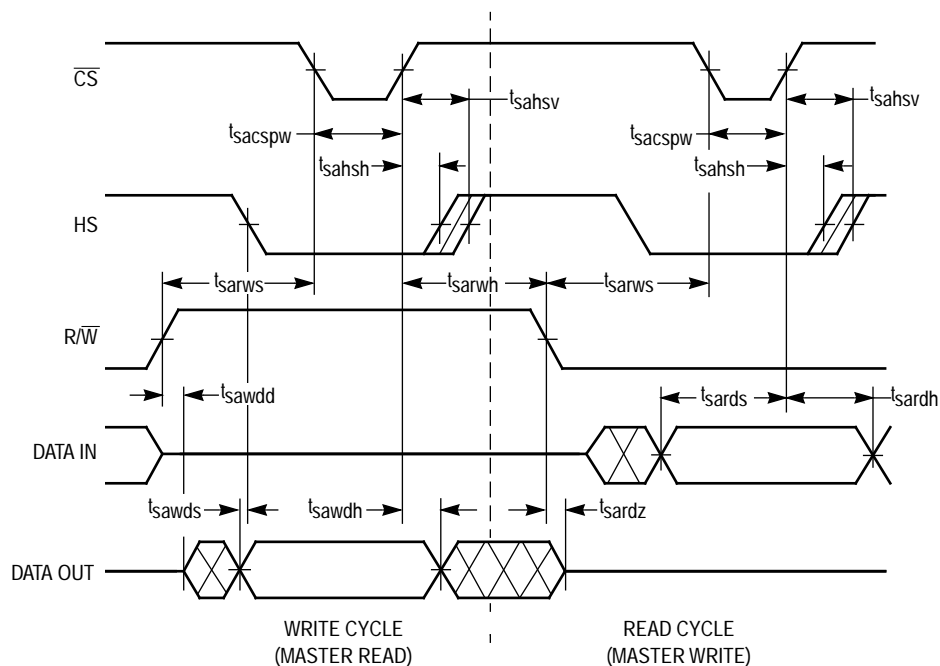


Symbol	Description	Min	Typ	Max
t_{mrws}	R/ \overline{W} setup before falling edge of \overline{CS}	150 ns	3 CLK1	—
t_{mrwh}	R/ \overline{W} hold after rising edge of \overline{CS}	100 ns	—	—
t_{mcspw}	\overline{CS} pulse width	150 ns	2 CLK1	—
t_{mhsh}	HS hold after falling edge of \overline{CS}	0 ns	—	—
t_{mhsv}	HS checked by firmware after rising edge of \overline{CS}	150 ns	10 CLK1	—
t_{mrdz}	Master three-state DATA after rising edge of R/ \overline{W} (Notes 1, 2)	—	0	25 ns
t_{mrds}	Read data setup before falling edge of HS (Note 3)	0 ns	—	—
t_{mhscs}	HS low to falling edge of \overline{CS} (Note 4)	2 CLK1	6 CLK1	—
t_{mrdh}	Read data hold after falling edge of \overline{CS}	0 ns	—	—
t_{mwdd}	Master drive of DATA after falling edge of R/ \overline{W} (Note 5)	150 ns	2 CLK1	—
t_{mhds}	HS low to data valid (Note 4)	—	50 ns	—
t_{mwds}	Write data setup before rising edge of \overline{CS}	150 ns	2 CLK1	—
t_{mwdh}	Write data hold after rising edge of \overline{CS} (Note 6)	Note 1	—	—

NOTES:

1. Refer to Figures 6-6 and 6-7 for detailed measurement information.
2. For Neuron Chip-to-Neuron Chip operation, bus contention (t_{mrdz} , t_{sawdd}) is eliminated by firmware, ensuring that a zero state is present when the token is passed between the master and slave. See AN1208, *Parallel I/O Interface to the Neuron Chip*, for further information.
3. HS high is used as a slave busy flag. If HS is held low, the maximum data transfer rate is 24 CLK1s (2.4 μ s @ 10 MHz) per byte. If HS is not used for a flag, caution should be taken to ensure the master does not initiate a data transfer before the slave is ready.
4. Parameters were added in order to aid interface design with the Neuron Chip.
5. Refer to Figures 6-2 and 6-5 for detailed measurement information.
6. Master will hold output data valid during a write until the Slave device pulls HS low.
7. CLK1 represents the period of the Neuron Chip input clock (100 ns at 10 MHz).
8. In a master read, \overline{CS} pulsing low acts like a handshake to flag the slave that data has been latched in.

Figure 5-16. Master Mode Timing



Symbol	Description	Min	Typ	Max
t_{sarws}	R/\overline{W} setup before falling edge of \overline{CS}	25 ns	—	—
t_{sarwh}	R/\overline{W} hold after rising edge of \overline{CS}	0 ns	—	—
t_{sacspw}	\overline{CS} pulse width	45 ns	—	—
t_{sahsh}	HS hold after rising edge of \overline{CS}	0 ns	—	—
t_{sahsv}	HS valid after rising edge of \overline{CS}	—	—	50 ns
t_{sawdd}	Slave A drive of DATA after rising edge of R/\overline{W} (Notes 1, 2)	0 ns	5 ns	—
t_{sawds}	Write data valid before falling edge of HS	150 ns	2 CLK1	—
t_{sawdh}	Write data valid after rising edge of \overline{CS}	150 ns (Note 3)	2 CLK1	—
t_{sardz}	Slave A three-state DATA after falling edge of R/\overline{W} (Note 4)	—	—	50 ns
t_{sards}	Read data setup before rising edge of \overline{CS}	25 ns	—	—
t_{sardh}	Read data hold after rising edge of \overline{CS}	10 ns	—	—

NOTES:

1. Refer to Figures 6-2 and 6-5 for detailed measurement information.
2. For Neuron Chip-to-Neuron Chip operation, bus contention (t_{mrdz} , t_{sawdd}) is eliminated by firmware, ensuring that a zero state is present when the token is passed between the master and slave. See AN1208, *Parallel I/O Interface to the Neuron Chip*, for further information.
3. If $t_{sarwh} < 150$ ns, then $t_{sawdh} = t_{sarwh}$.
4. Refer to Figures 6-6 and 6-7 for detailed measurement information.
5. CLK1 represents the period of the Neuron Chip input clock (100 ns at 10 MHz).
6. In slave A mode, the HS signal is high a minimum of 4 CLK1 periods. The typical time HS is high during consecutive data reads or consecutive data writes is also 4 CLK1 periods.

Figure 5-17. Slave A Mode Timing

Following is an example program for transferring data in a parallel I/O master/slave A configuration. The code is for two Neuron Chip emulators hardwired as shown in Figure 5-15. A Motorola M143204EVK Direct Connect Board and cable can be used to connect these pins on the emulators. The master program writes the test_data to the slave's input buffer (as the master owns the token after reset and has the first option to write on the bus) and the slave then outputs data to the master's input buffer. The buffers can be viewed through the debug option on the LonBuilder Developer's Workbench to verify the transfer was complete. The master transmits [5,1,1,1,1,1] to the slave and the slave transmits [7,1,2,3,4,5,6,7,0,0,0,0,0] to the master. The first byte indicates the number of bytes being passed; the following non-zero valued bytes in this example are the actual data transferred. The remaining length of the array, if any, is filled with 0s. This program writes once to the slave and reads once from the slave. To implement continuous writes and reads, add an io_out_request statement after the io_in statement on the master's program.

```
/* This is the master program. After reset, the buffer is filled with 1s and then the buffer is written to the
   slave. The master then reads the slave's buffer. The master's output buffer should contain [5,1,1,1,1,1];
   the input buffer should contain [7,1,2,3,4,5,6,7,0,0,0,0,0].
```

```
*/
```

```
IO_0 parallel master parallel_bus;
```

```
#define TEST_DATA 1           // data to be written in output buffer
#define MAX_IN 13            // maximum length of input data expected
#define OUT_LEN 5            // output length can be equal to or less than the max
#define MAX_OUT 5            // maximum array length
```

```
struct parallel_out           // output structure
{
    unsigned int len;         // actual length of data to be output
    unsigned int buffer[MAX_OUT]; // array setup for max length of data to be output
}p_out;                       // output structure name
```

```
struct parallel_in           // input structure
{
    unsigned int len;         // actual length of buffer to be input
    unsigned int buffer[MAX_IN]; // maximum input array
}p_in;                       // input structure name
```

```
unsigned int i;
```

```
when (reset)
{
    p_out.len=OUT_LEN;        // assign output length
    for(i=0; i<OUT_LEN; ++i) // fill output buffer with 1s
        p_out.buffer[i]=TEST_DATA;
    io_out_request(parallel_bus); // request to output buffer
}
```

```
when (io_out_ready(parallel_bus))
{
    io_out(parallel_bus, &p_out); // output buffer when slave is ready
}
```

```
when (io_in_ready(parallel_bus))
{
    p_in.len=MAX_IN;          // declare the maximum input buffer acceptable
    io_in(parallel_bus, &p_in); // store input data in buffer
} //end of program
```

```

/* This is the slave program. After reset, the output buffer is filled with data and then the slave reads from
the master. The slave then writes to the master. The slave's input buffer should contain [5,1,1,1,1,1]; the
output buffer should contain [7,1,2,3,4,5,6,7,0,0,0,0,0,0].
*/

```

```

IO_0 parallel slave parallel_bus;

```

```

#define MAX_IN 5           // maximum length of input data expected
#define OUT_LEN 7         // output length can be equal to or less than the max
#define MAX_OUT 13        // maximum array length

struct parallel_out        // output structure
{
    unsigned int len;       // actual length of data to be output
    unsigned int buffer[MAX_OUT]; // array setup for max length of data to be output
}p_out;                   // output structure name

struct parallel_in         // input structure
{
    unsigned int len;       // actual length of buffer to be input
    unsigned int buffer[MAX_IN]; // maximum input array
}p_in;                    // input structure name

unsigned int i;

when (reset)
{
    p_out.len=OUT_LEN;      // assign output length
    for(i=0; i<OUT_LEN; ++i) // fill output buffer with 1s
        p_out.buffer[i]=i+1;
}

when (io_out_ready(parallel_bus))
{
    io_out(parallel_bus, &p_out); // output buffer
}

when (io_in_ready(parallel_bus))
{
    p_in.len=MAX_IN;        // declare the maximum input buffer acceptable
    io_in(parallel_bus, &p_in); // store input data in buffer
    io_out_request(parallel_bus); // request to output buffer
} //end of program

```

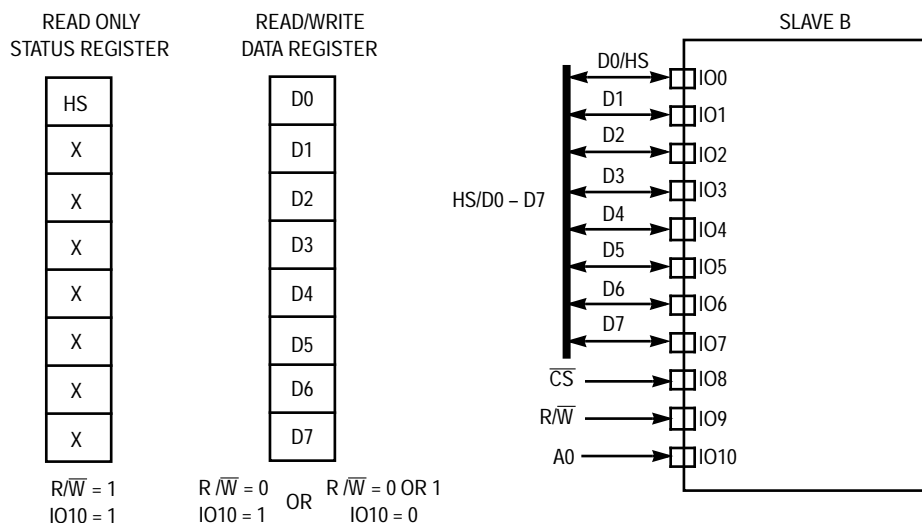
Debugging the Above Programs: On the LonBuilder development workstation, if a watchdog timeout occurs, simultaneously reset the two emulators using the reset pushbutton switches on the face of the emulators. Both JP1 and JP2 on the emulator boards should be disconnected for this application.

5.4.3 Slave B Mode

The slave B mode is recommended for interfacing a Neuron Chip (as the slave) to a non-Neuron Chip processor (as the master). When configured in slave B mode, the Neuron Chip accepts IO8 as a chip select and IO9 to specify whether the master will read or write, and accepts IO10 as a register select input. When \overline{CS} is asserted and either IO10 is low or IO10 is high and R/\overline{W} is low, pins IO0 – IO7 form the bidirectional data bus. When IO10 is high, R/\overline{W} is high, and \overline{CS} is asserted; IO0 is driven as the HS acknowledgment signal to the master.

The Neuron Chip may appear as two registers in the master's address space; one of the registers being the read/write data register, and the other being the read-only status register. Therefore, reads by the master to an odd address, access the status register for handshaking acknowledgments and all other reads or writes access the data register for I/O transfers. The LSB of the control register, which is read through pin IO0, is the HS bit. The master reads the HS bit after every master read or write.

In Neuron Chip-to-non-Neuron Chip interface, the Neuron Chip slave B handles all handshaking and token passing, automatically. However, a non-Neuron Chip master must read the HS bit after each transaction and must also internally track the token passing. This mode is designed for use with a master processor that uses memory-mapped I/O, as the LSB of the master's address bus is typically connected to the Neuron Chip's IO10 pin. This is illustrated in Figures 5-18 and 5-19.



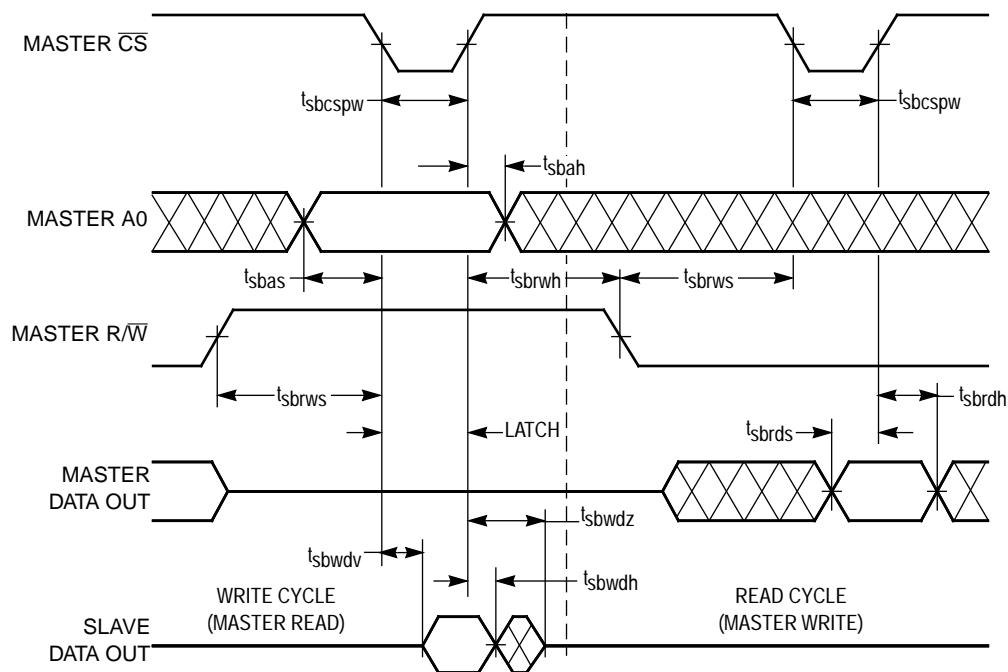
**Figure 5-18. Parallel I/O Master/Slave B
(Neuron Chip as Memory-Mapped I/O Device)**

5.4.4 Token Passing

Virtual token passing is implemented to eliminate the possibility of data bus contention. The token is owned by the master after synchronization and is passed between the master and slave nodes. After each data transfer is completed, the token owner writes an end of message (EOM) (0x00) to indicate transfer is complete. The EOM is never read. Instead, "processing the EOM" indicates passing of the token. Token passing can be achieved by executing either a data packet or a NULL transfer. Only the owner of the token can write to the bus. Therefore, when the master performs two writes of data (1 – 255 bytes each) a dummy read cycle (NULL character = 0x00) must be inserted between them in order to pass the token. Token passing is executed automatically in a Neuron Chip-to-Neuron Chip interface. Refer to Section 5.4.6 for master/slave flow transactions.

5.4.5 Handshaking

Handshaking allows the master to monitor the slave between every byte transfer, ensuring that both processors are ready for the byte to be transferred. If the master owns the token, the master waits for the HS from the slave before writing data to the bus. If the slave owns the token, the master monitors the low transition of the HS before reading the bus. In master or slave A mode, the Neuron Chip HS line is pin IO10. In slave B mode, the Neuron Chip HS bit is monitored on IO0 which corresponds to the least significant data bit of the status register.



Symbol	Description	Min	Typ	Max
t_{sbrws}	R/ \overline{W} setup before falling edge of \overline{CS} MC143150 and MC143120	0 ns	—	—
t_{sbrwh}	R/ \overline{W} hold after rising edge of \overline{CS}	0 ns	—	—
t_{sbcspw}	\overline{CS} pulse width	Note 1	—	—
t_{sbas}	A0 setup to falling edge of \overline{CS} MC143150 and MC143120	10 ns	—	—
t_{sbah}	A0 hold after rising edge of \overline{CS}	0 ns	—	—
t_{sbwdv}	\overline{CS} to write data valid	—	—	50 ns
t_{sbwdh}	Write data hold after rising edge of \overline{CS} (Notes 2, 3)	0 ns	30 ns	—
t_{sbwdz}	\overline{CS} rising edge to Slave B release data bus (Note 2)	—	—	50 ns
t_{sbrds}	Read data setup before rising edge of \overline{CS}	25 ns	—	—
t_{sbrdh}	Read data hold after rising edge of \overline{CS}	10 ns	—	—

NOTES:

1. The slave B write cycle (master read) \overline{CS} pulse width is directly related to the slave B write data valid parameter and master read setup parameter. To calculate the write cycle \overline{CS} duration needed for a special application use:

$$t_{sbcspw} = t_{sbwdv} + \text{master's read data setup before rising edge of } \overline{CS}$$
Refer to the *Master's Specification Data Book* for the master read setup parameter. The slave read cycle minimum \overline{CS} pulse width = 50 ns.
2. Refer to Figures 6-6 and 6-7 for detailed measurement information.
3. The data hold parameter, t_{sbwdh} , is measured to the disable levels shown in Figure 6-7, rather than to the traditional data invalid levels.
4. In a slave B write cycle the timing parameters are the same for a control register (HS) write as for a data write.
5. Special applications: Both the state of \overline{CS} and R/ \overline{W} determine a slave B cycle. If \overline{CS} can not be used for a data transfer, then toggling the R/ \overline{W} line can be used with no changes to the hardware. In other words, if \overline{CS} is held low during a slave B write cycle, a positive pulse (low to high to low) on R/ \overline{W} can execute a data transfer. The low to high transition on R/ \overline{W} causes slave B to drive data with the same timing parameters as t_{sbwdv} (redefined R/ \overline{W} to write data valid). Likewise, the falling edge of R/ \overline{W} causes slave B to release the data bus with the same timing limits as the \overline{CS} rising edge in t_{sbwdz} . This scenario is only true for a slave B write cycle and is not applicable to a slave B read cycle or any slave A data transitions. This application may be helpful if the master has separate read and write signals but no \overline{CS} signal. Caution must be taken to ensure the bus is free before transfers to avoid bus contention.

Figure 5-19. Slave B Mode Timing

5.4.6 Data Transferring

The data transfer operation between the master and the slave is accomplished through the use of a virtual write token-passing protocol. The write token is passed alternatively between the master and the slave on the bus in an infinite ping-pong fashion. The owner of the token has the option of writing a series of data bytes, or alternatively, passing the write token without any data. Figure 5-20 illustrates the sequence of operations for this token passing protocol.

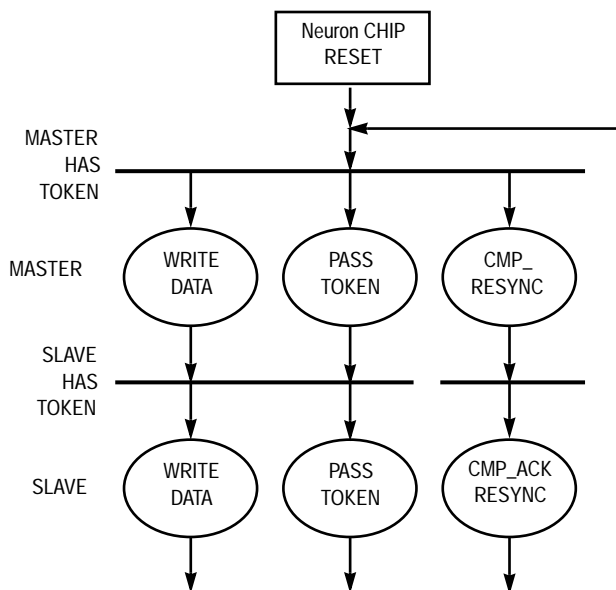


Figure 5-20. Handshake Protocol Sequence Between the Master and the Slave

Once in possession of the write token, the device (Neuron Chip or a host processor) can transfer up to 255 bytes of data. The stream of data bytes is preceded by the command and length bytes. The token holder keeps possession of the token until all data bytes have been written, after which the token is passed to the attached device. The same process may now be repeated by the other side or, alternatively, the token can be passed back without any data. The timing relationship between the various Neuron Chip signals involved is shown in the following timing diagrams.

Resynchronization Procedure: The following procedure applies to master/slave A and master/slave B configuration. The master initiates the resynchronization with a RESYNC (0x5A) command, and the slave acknowledges with an ACKSYNC (0x07). If the slave does not respond, the master continues to send the RESYNC until the slave responds correctly.

MASTER	SLAVE	
(Owns Token)		
Write RESYNC		// master initiates resynchronization (0x5A)
	Read RESYNC	
Write EOM		// end of message (EOM=0x00)
	Process EOM	
	Write ACKSYNC	// slave acknowledges resynching (0x07)
Read ACKSYNC		
	Write EOM	
Process EOM		// master owns token when reset
(Owns Token)		

Master writes buffer to slave: Enter RD/_WR=0.

MASTER (Owns Token)	SLAVE	
Write XFER		// master has data to write (XFER=0x01)
Write (length)	Read XFER	// length=number of bytes of data
Write (data_0)	Read (length)	
.	Read (data_0)	// master begins data transfer to slave
.	.	
.	.	
Write (data_n)	.	// last byte of data to be transferred
Write EOM	Read (data_n)	
	Process EOM	// end of data transfer (EOM=0x00)
	(Owns Token)	// exchange token

Slave writes buffer to master: Enter RD/_WR=1.

MASTER	SLAVE (Owns Token)	
	Write XFER	// slave has data to write (XFER=0x01)
Read XFER	Write (length)	// length=number of bytes of data
Read (length)	Write (data_0)	// slave begins writing data to master
Read (data_0)	.	
.	.	
.	.	
	Write (data_n)	// last byte of data to be transferred
Read (data_n)	Write EOM	
Process EOM		// end of data transfer
(Owns Token)		// exchange token

Master passes token to slave: Entry same as when master writes buffer to slave.

MASTER (Owns Token)	SLAVE	
Write NULL		// master has no data to send to slave
	Read NULL	// NULL=0x00
Write EOM		// end of message (EOM=0x00)
	Process EOM	// exchange token
	(Owns Token)	

Slave passes token to master: Entry same as when slave writes buffer master.

MASTER	SLAVE (Owns Token)	
	Write NULL	// slave has no data to send to the master
Read NULL		// NULL=0x00
	Write EOM	// end of message (EOM=0x00)
Process EOM (Owns Token)		// exchange token

5.5 SERIAL OBJECTS

The timing numbers shown in this section are valid for both an explicit I/O call or an implicit I/O call through a *when* clause, and are assumed to be for a Neuron Chip running at 10 MHz.

5.5.1 Bitshift I/O

Pairs of adjacent pins may be configured as serial input or output lines, the even numbered pin being used for the clock (driven by the Neuron Chip) and the odd numbered pin being used for up to 16 bits of serial data. The data rate may be configured as 1 kbps, 10 kbps, or 15 kbps at maximum input clock rate (10 MHz). The data rate scales proportionally to the input clock rate. The active clock edge may be specified as either rising or falling. This object is useful for transferring data to external logic employing shift registers. This function suspends application processing until the operation is complete. See Figures 5-21, 5-22, and 5-23.

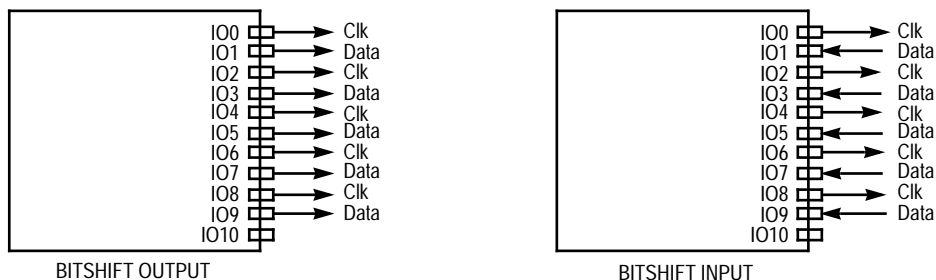
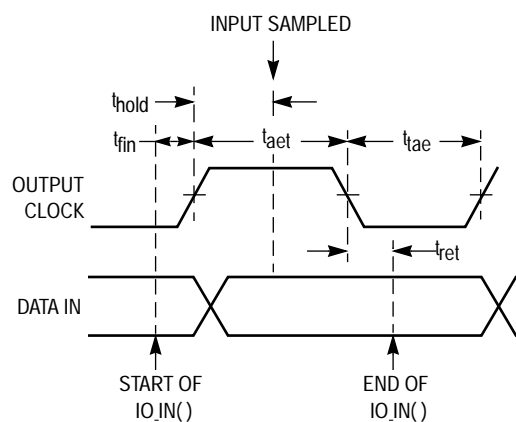


Figure 5-21. Bitshift I/O

For Bitshift input, the clock output is deasserted (to the inactive level) at the same time as the start of the first bit of data.

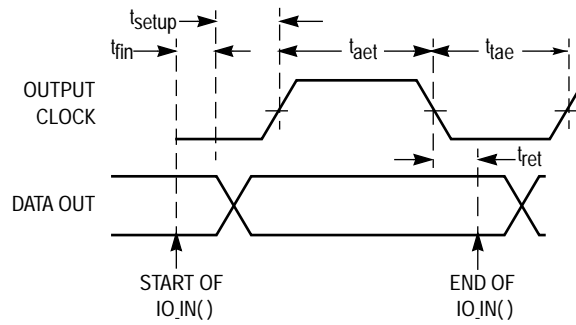
For Bitshift output, the clock output is initially inactive prior to the first bit of data (unless overridden by a bit output overlay).



Active clock edge assumed to be positive in the above diagram

Symbol	Description	Typ @ 10 MHz
t_{fin}	Function call to first edge	156.6 μ s
t_{ret}	Return from function	5.4 μ s
t_{hold}	Active clock edge to sampling of input data 15 kbaud data rate 10 kbaud data rate 1 kbaud data rate	9 μ s 40.8 μ s 938.2 μ s
t_{aet}	Active clock edge to next clock transition 15 kbaud data rate 10 kbaud data rate 1 kbaud data rate	31.8 μ s 63.6 μ s 961 μ s
t_{tae}	Clock transition to next active clock edge 15 kbaud data rate 10 kbaud data rate 1 kbaud data rate	14.4 μ s 14.4 μ s 14.4 μ s
f	Clock frequency = $1/(t_{aet} + t_{tae})$ 15 kbaud data rate 10 kbaud data rate 1 kbaud data rate	21.6 kHz 12.8 kHz 1.03 kHz

Figure 5-22. Bitshift Input Latency Values



Active clock edge assumed to be positive in the above diagram

Symbol	Description	Typ @ 10 MHz
t_{fin}	Function call to first data out stable 16-bit shift count 1-bit shift count	185.3 μ s 337.6 μ s
t_{ret}	Return from function	10.8 μ s
t_{setup}	Data out stable to active clock edge 15 kbaud data rate 10 kbaud data rate 1 kbaud data rate	10.8 μ s 10.8 μ s 10.8 μ s
t_{aet}	Active clock edge to next clock transition 15 kbaud data rate 10 kbaud data rate 1 kbaud data rate	10.2 μ s 42 μ s 939.5 μ s
t_{ae}	Clock transition to next active clock edge 15 kbaud data rate 10 kbaud data rate 1 kbaud data rate	34.8 μ s 34.8 μ s 34.8 μ s
f	Clock frequency = $1/(t_{aet} + t_{ae})$ 15 kbaud data rate 10 kbaud data rate 1 kbaud data rate	22 kHz 13 kHz 1.02 kHz

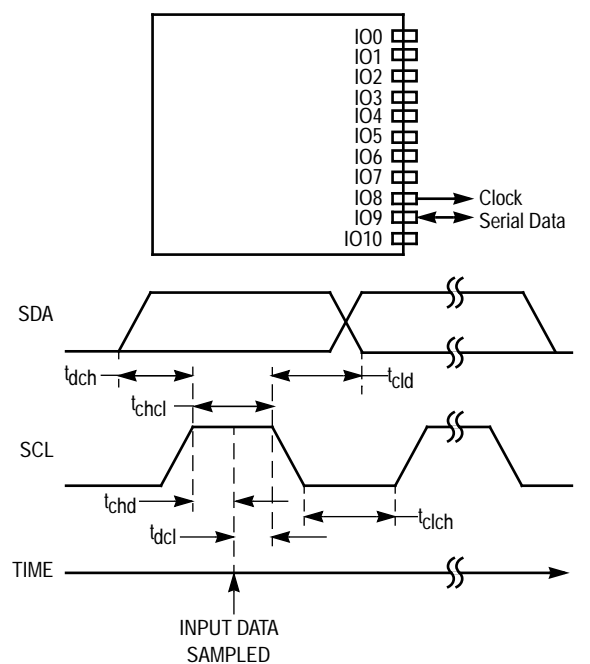
Figure 5-23. Bitshift Output Latency Values

5.5.2 I²C I/O

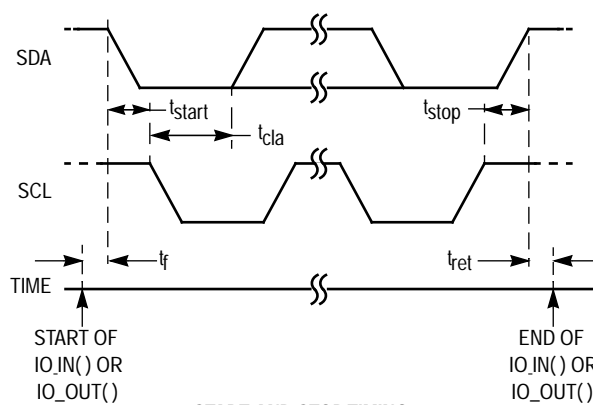
This I/O object is used to interface the Neuron Chip to any device which adheres to Philips Semiconductor's Inter-Integrated Circuit (I²C) bus protocol. The Neuron Chip is always the master, with IO8 being the serial clock (SCL) and IO9 the serial data (SDA). These I/O lines are operated in the open-drain mode in order to accommodate the special requirements of the I²C protocol. With the exception of two pull-up resistors, no additional external components are necessary for interfacing the Neuron Chip to an I²C device.

Up to 255 bytes of data may be transferred at a time. At the start of all transfers, a right-justified 7-bit I²C address argument is sent out on the bus immediately after the I²C "start condition."

For more information on this protocol, refer to Philips Semiconductor's I²C documentation.



BIT TRANSFER TIMING



START AND STOP TIMING

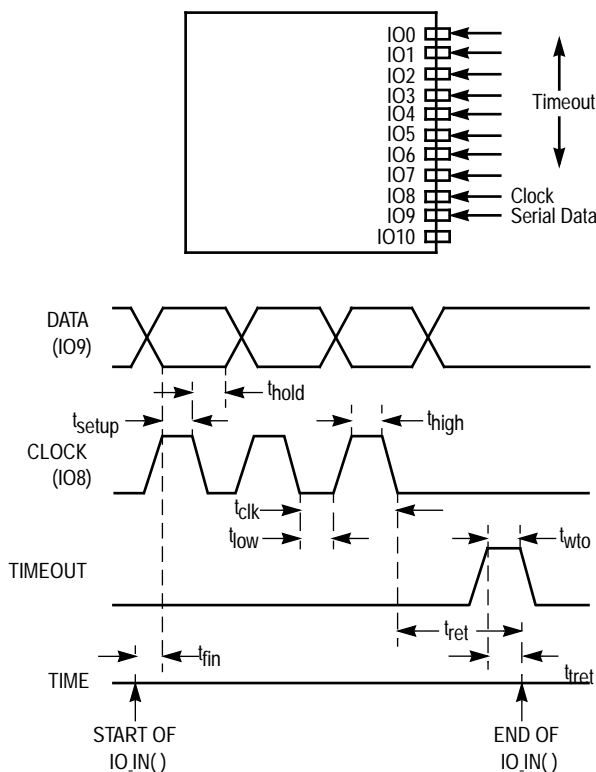
Parameter	Description	Min	Typ	Max
t_f	I/O call to start condition	—	54.6 μ s	—
t_{start}	End of start condition	5.4 μ s	—	—
t_{cla}	End of start to start of address	24.0 μ s	—	—
t_{cld}	SCL low to data for io_out()	24.6 μ s	—	—
t_{dch}	Data to SCL high for io_out()	7.2 μ s	—	—
t_{chcl}	Clock high to clock low for io_out()	12.6 μ s	—	—
t_{chd}	SCL high to data sampling for io_in()	13.2 μ s	—	—
t_{dcl}	Data sample to SCL low for io_in()	7.2 μ s	—	—
t_{clch}	Clock low to clock high for io_in()	24.0 μ s	—	—
t_{stop}	Clock high to data	12.6 μ s	—	—
t_{ret}	SDA high to return from function	—	—	4.2 μ s

Figure 5-24. I²C I/O Object

5.5.3 Magcard Input

This I/O object is used to transfer synchronous serial data from an ISO 7811 Track 2 magnetic stripe card reader in real time. The data is presented as a data signal input on pin IO9, and a clock, or a data strobe, signal input on pin IO8. The data on pin IO9 is clocked on or just following the falling (negative) edge of the clock signal on IO8, with the LSB first. In addition, any one of the pins IO0 – IO7 may be used as a timeout pin to prevent lockup in case of abnormal abort of the input bit stream during the input process.

Up to 40 characters may be read at one time. Both the parity and the Longitudinal Redundancy Check (LRC) are checked by the Neuron Chip.



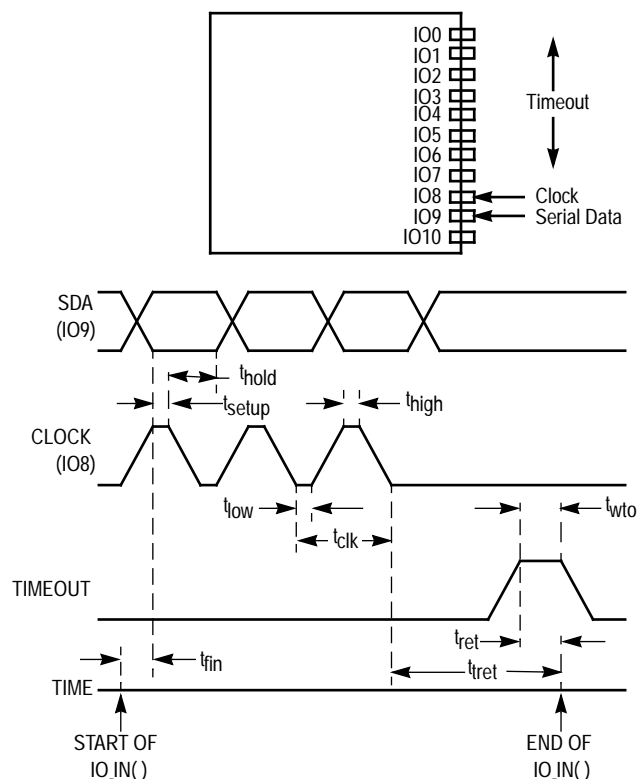
Symbol	Description	Min	Typ	Max
t_{fin}	Function call to first clock input	—	45.0 μ s	—
t_{hold}	Data hold	0 μ s	—	—
t_{setup}	Data setup	0 μ s	—	—
t_{low}	Clock low width	60 μ s	—	—
t_{high}	Clock high width	60 μ s	—	—
t_{wto}	Width of timeout pulse	60 μ s	—	—
t_{clk}	Clock period	120 μ s	—	—
t_{tret}	Return from timeout	21.6 μ s	—	81.6 μ s
t_{ret}	Return from function	—	—	301.8 μ s

Figure 5-25. Magcard Input Object

A Neuron Chip operating at 10 MHz can process a bit rate at up to 8334 bits/second (of a bit density of 75 bits/inch). This equates to a card velocity of 111 inches/second. Most magnetic card stripes contain a 15-bit sequence of zero data at the start of the card, allowing time for the application to start the card reading function. At 8334 bits/second, this period is about 1.8 ms. If the scheduler latency is greater than the 1.8 ms value, the `io_in()` function will miss the front end of the data stream.

5.5.4 Magtrack1 Input

This input object type is used to read synchronous serial data from an ISO3554 magnetic stripe card reader. The data input is on pin IO9, and the clock, or data strobe, is presented as input on pin IO8. The data on pin IO9 is clocked in just following the falling edge of the clock signal on IO7, with the LSB first.



Symbol	Description	Min	Typ	Max
t_{fin}	Function call to first clock input	—	45.0 μ s	—
t_{hold}	Data hold	t_{low}	—	t_{clk}
t_{setup}	Data setup	0 μ s	—	—
t_{low}	Clock low width	31 μ s	—	—
t_{high}	Clock high width	31 μ s	—	—
t_{wto}	Width of timeout pulse	60 μ s	—	—
t_{clk}	Clock period	138 μ s	—	—
t_{ret}	Return from timeout	21.6 μ s	—	81.6 μ s
t_{ret}	Return from function	—	—	301.8 μ s

Figure 5-26. Magtrack1 Input Object

Note that the minimum period for the entire bit cycle (t_{clk}) is greater than the sum of t_{low} and t_{high} . The t_{setup} and t_{hold} times should be such that the data is stable for the duration of t_{low} .

Data are recognized in the IATA format as a series of 6-bit characters plus an even parity bit per character. The process begins when the start sentinel (hex 05) is recognized, and continues until the end sentinel (0x0F) is recognized. No more than 79 characters, including the 2 sentinels and the LRC character, will be read. The data is stored as right-justified bytes in the buffer space pointed to by the buffer pointer argument in the `io_in()` function with the parity stripped, and includes the start and end sentinels. This buffer should be 78 bytes long.

The Magtrack1 input object optionally uses one of the I/O pins IO0 – IO7 as a timeout/abort pin. Use of this feature is suggested since the `io_in()` function will update the watchdog timer during clock wait states, and could result in a lockup if the card were to stop moving in the middle of the transfer process. If a logic 1 level is detected on the I/O timeout pin, the `io_in()` function will abort. This input can be a oneshot timer counter output, an R/C circuit, or a `DATA_VALID` signal from the card reader.

A Neuron Chip with a clock rate of 10 MHz can process an incoming bit rate of up to 7246 bits/second when the strobe signal has a 1/3 duty cycle ($t_{\text{high}} = 46 \mu\text{s}$, $t_{\text{low}} = 92 \mu\text{s}$). At a bit density of 210 bits/inch, this translates to a card speed of 34.5 inches/second.

5.5.5 Neurowire (SPI Interface) I/O Object

The Neurowire object implements a full-duplex synchronous transfer of data to some peripheral device. It can operate as the master (drive a clock out) or as the slave (accept a clock in). In both master and slave modes, up to 255 bits of data may be transferred at a time. The Neurowire I/O suspends application processing until the operation is completed. The Neurowire object is useful for external devices, such as A/D, D/A converters, and display drivers incorporating serial interfaces that conform with Motorola's SPI interface. See Figure 5-27.

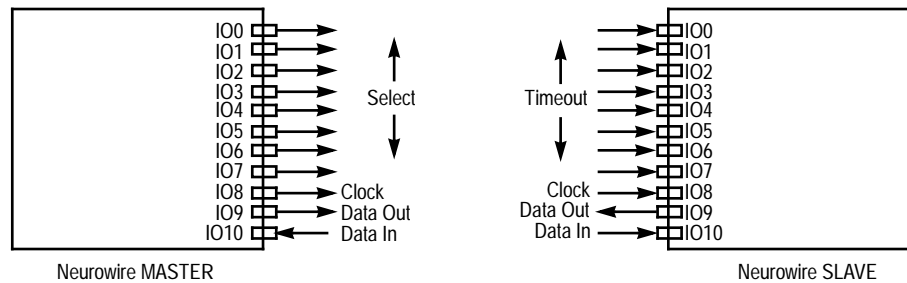
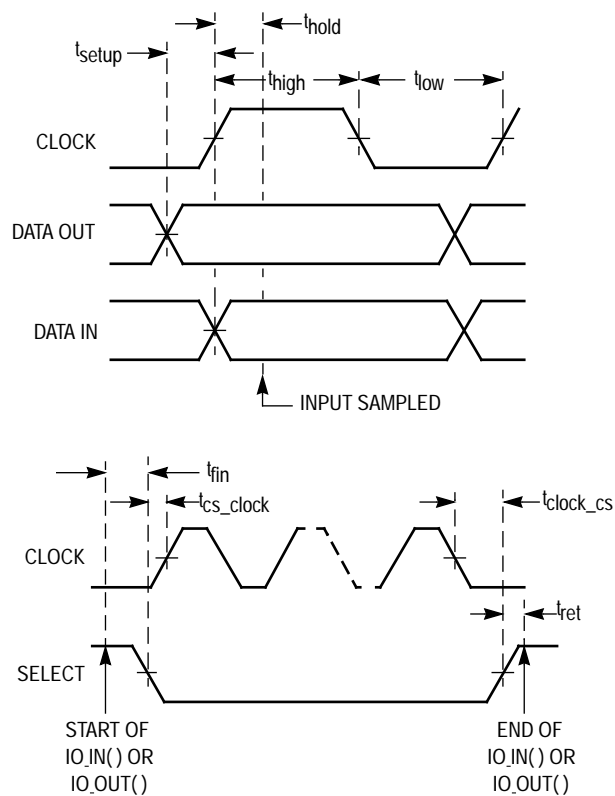


Figure 5-27. Neurowire I/O

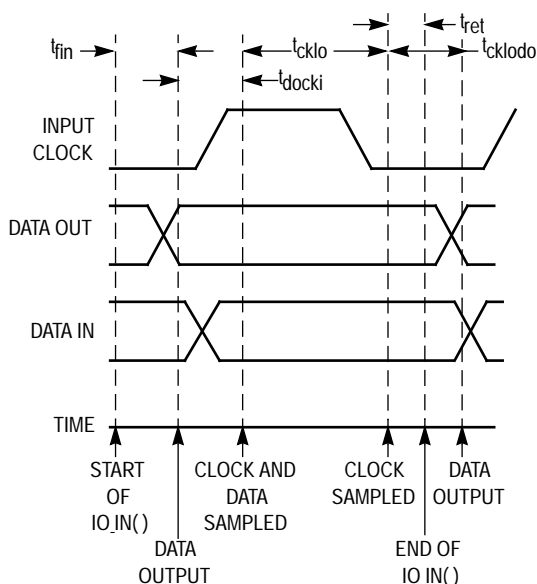
5.5.5.1 Neurowire MASTER MODE. In Neurowire master mode, pin IO8 is the clock (driven by the Neuron Chip), IO9 is the serial data output, and IO10 is the serial data input. Serial data is clocked out on pin IO9 at the same time as data is clocked in from pin IO10. Data is clocked by the rising edge of the clock signal by default. The *clockedge* keyword changes the active edge of the clock to negative. In addition, one or more of the pins IO0 – IO7 may be used as a chip select, allowing multiple Neurowire devices to be connected on a three-wire bus. The clock rate may be specified as 1 kbps, 10 kbps, or 20 kbps at an input clock rate of 10 MHz; these scale proportionally with input clock. See Figure 5-28.



Parameter	Description	Typ
t_{fin}	Function call to \overline{CS} active	69.9 μ s
t_{ret}	Return from function	7.2 μ s
t_{hold}	Active clock edge to sampling of input data 20 kbaud data rate 10 kbaud data rate 1 kbaud data rate	11.4 μ s 53.4 μ s 960.6 μ s
t_{high}	Period, clock high (active clock edge = 1) 20 kbaud data rate 10 kbaud data rate 1 kbaud data rate	25.8 μ s 67.8 μ s 975.0 μ s
t_{low}	Period, clock low (active clock edge = 1)	33.0 μ s
t_{setup}	Data output stable to active clock edge	5.4 μ s
t_{cs_clock}	Select active to first active clock edge	91.2 μ s
t_{clock_cs}	Last clock transition to select inactive	81.6 μ s
f	Clock frequency = $1/(t_{high} + t_{low})$ 20 kbaud data rate 10 kbaud data rate 1 kbaud data rate	17.0 kHz 9.92 kHz 992 Hz

Figure 5-28. Neurowire (SPI) Master Timing

5.5.5.2 Neurowire SLAVE MODE. In Neurowire slave mode, pin IO8 is the clock (driven by the external master), IO9 is the serial data output, and IO10 is the serial data input. Serial data is clocked out on pin IO9 at the same time as data is clocked in from pin IO10. Data is clocked by the rising edge of the clock signal (default), which may be up to 18 kbps. The `invert` keyword changes the active clock edge to negative. One of the pins IO0 – IO7 may be designated as a timeout pin. A logic 1 level on the timeout pin causes the Neurowire slave I/O operation to be terminated before the specified number of bits has been transferred. This prevents the Neuron Chip watchdog timer from resetting the chip in the event that fewer than the requested number of bits are transferred by the external clock. See Figure 5-29.



Parameter	Description	Typ
t_{fin}	Function call to data bit out	41.4 μ s
t_{ret}	Return from function	19.2 μ s
t_{docki}	Data out to input clock and data sampled	4.8 μ s
t_{cklo}	Data sampled to clock low sampled	24.0 μ s
t_{cklodo}	Clock low sampled to data output	25.8 μ s
f	Clock frequency (max)	18.31 kHz

Figure 5-29. Neurowire (SPI) Slave Timing

The algorithm for each bit of output/input for the Neurowire slave objects is described below. In this description, the default active clock edge (positive) is assumed; if the `invert` keyword is used, all clock levels stated should be reversed.

1. Set IO9 to the next output bit value.
2. Test pin IO8, the clock input, for a high level. This is the test for the rising edge of the input clock. If the input clock is still low, sample the timeout event pin and abort if high.
3. When the input clock is high, store the next data input bit as sampled on pin IO10.
4. Test the input clock for a low input level. This is the test for the falling edge of the input clock. If the input clock is still high, sample the timeout event pin and abort if high.
5. When the input clock is low, return to step 1 if there are more bits to be processed.
6. Else return the number of bits processed.

When either clock input test fails (that is, the clock is sampled *before* the next transition), there is an additional timeout check time of 19.8 μs (wait for clock high) or 19.2 μs (wait for clock low) added to that stage of the algorithm.

It is assumed that the chip select logic for the Neurowire slave is handled by the user through a separate bit input object, along with an appropriate handshaking algorithm implemented by the user application program. In order to prevent unnecessary timeouts, the setup and hold times of the chip select line, relative to the start and end of the external clock, must be satisfied.

The timeout input pin can either be connected to an external timer or to an output pin of the Neuron Chip that is declared as a oneshot object.

5.5.6 Serial I/O

Pin IO8 may be configured as an asynchronous serial input line, and pin IO10 may be configured as an asynchronous serial output line. The bit rates for input and for output may be independently specified to be 600, 1200, 2400, or 4800 bits/second at maximum input clock rate (10 MHz). The data rate scales proportionally to the input clock rate. The frame format is fixed at 1 start bit, 8 data bits, and 1 stop bit; and up to 255 bytes may be transferred at a time. Either a serial input or a serial output operation (but not both) may be in effect at any one time. The interface is half-duplex only. This function suspends application processing until the operation is completed. On input, the *io_in* request will time out after 20 character times if no start bit is received. If the stop bit has the wrong polarity (it should be a 1), the input operation is terminated with an error. The application code can use bit I/O pins for flow control handshaking if required. This function is useful for transferring data to serial devices such as terminals, modems, and computer serial interfaces. See Figures 5-30 and 5-31.

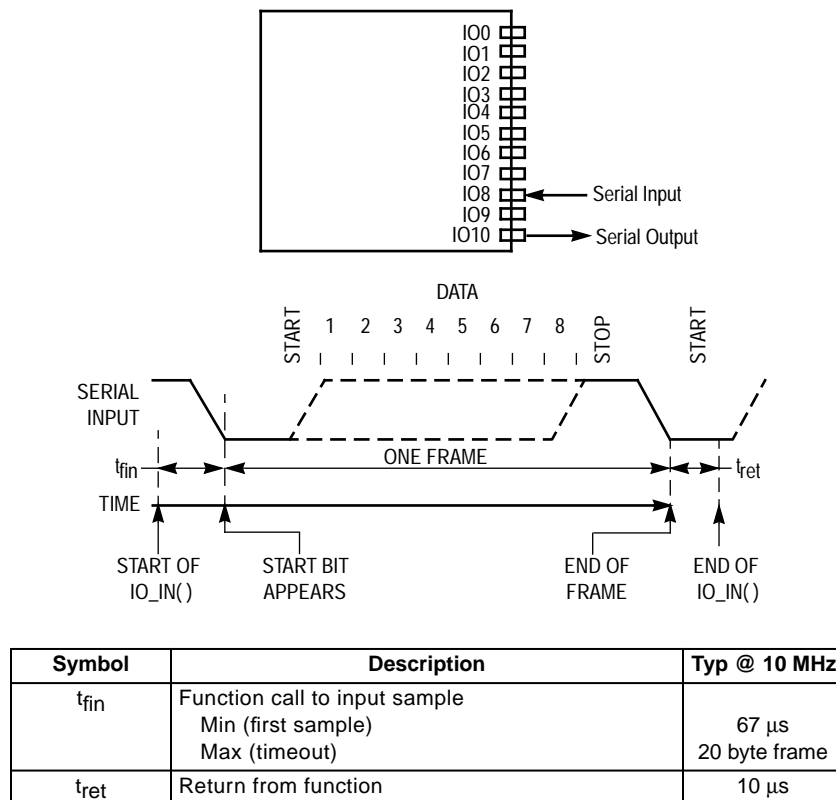


Figure 5-30. Serial Input Object

The duration of this function call is a function of the number of data bits transferred and the transmission bit rate. $t_{fin} \text{ (max)}$ refers to the maximum amount of time this function will wait for a start bit to appear at the input. After this time, the function will return a 0 as data. $t_{fin} \text{ (min)}$ is the time to the first sampling of the input pin. As an example, the timeout period at 2400 bits/second is $(20 \times 10 \times 1/2400) + t_{fin} \text{ (min)}$.

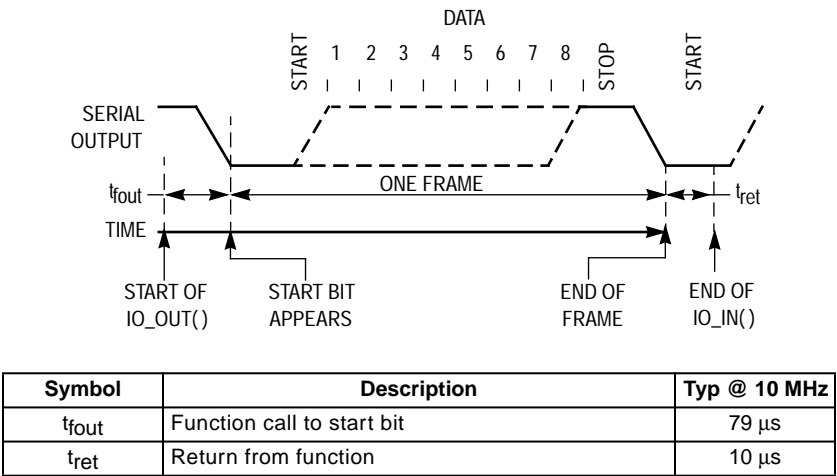


Figure 5-31. Serial Output

The duration of this function call is a function of the number of data bits transferred and the transmission bit rate. As an example, to output 100 bytes at 300 bits/second would require a time duration of $(100 \times 10 \times 1/300) + t_{fout} + t_{ret}$.

5.5.7 Touch I/O

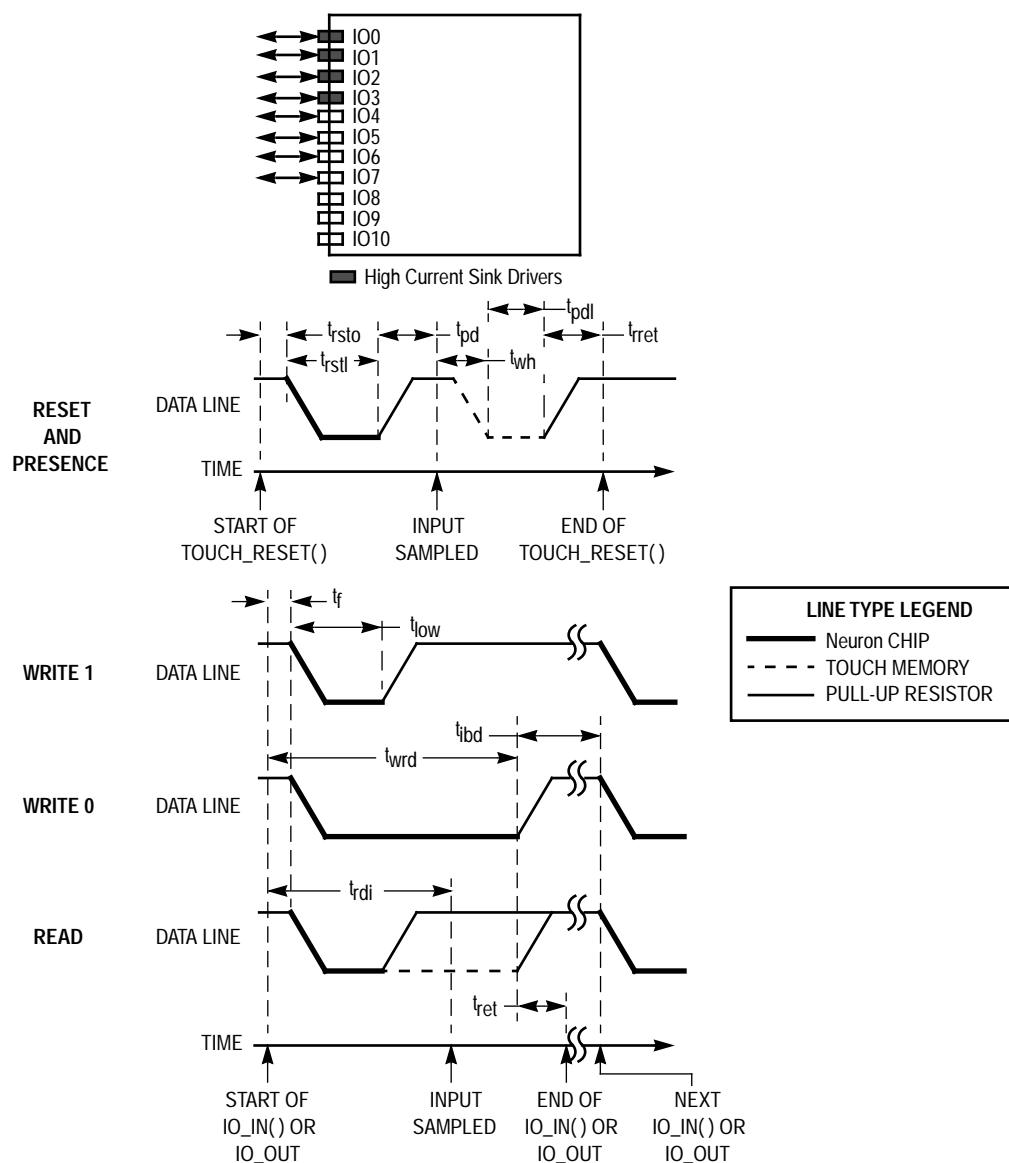
The Touch I/O object enables easy interface to any slave device which adheres to Dallas Semiconductor's Touch Memory™ standard. This interface is a one-wire, open-drain, bidirectional connection.

Up to eight Touch Memory busses may be connected to a Neuron Chip through the use of the first eight I/O pins, IO0 – IO7. The only additional component needed is a pull-up resistor on the data line (refer to the Touch Memory specification below on how to select the value of the pull-up resistor). The high current sink capabilities of IO0 – IO3 pins of the Neuron Chip can be used in applications where long wire lengths are needed between the Touch Memory device and the Neuron Chip.

The slave acquires all necessary power for its operation from the data line. Upon physical connection of a Touch Memory device to a master (in this case the Neuron Chip), the Touch Memory generates a low presence pulse to inform the master that it is awaiting a command. The Neuron Chip can also request a presence pulse by sending a reset pulse to the Touch Memory device.

Commands and data are sent bit by bit to make bytes, starting with the LSB. The synchronization between the Neuron Chip and the Touch Memory devices is accomplished through a negative-going pulse generated by the Neuron Chip.

Figure 5-32 shows the details of the reset pulse in addition to the read/write bit slots.



Symbol	Description	Min	Typ	Max
t_{rsto}	Reset call to data line low	—	60.0 μ s	—
t_{rstl}	Reset pulse width	—	500 μ s	—
t_{pdh}	Reset pulse release to data line high	10 MHz 5 MHz	4.8 μ s 9.6 μ s	275 μ s
t_{pdl}	Presence pulse width	—	120.0 μ s	—
t_{wh}	Data line high detect to presence pulse	—	80 μ s	—
t_{rret}	Return from reset function	—	12.6 μ s	—
t_f	I/O call to data line low (start of bit slot)	—	125.4 μ s	—
t_{low}	Start pulse width	10 MHz 5 MHz	4.2 μ s 8.4 μ s	—
t_{rdi}	Start pulse edge to sampling of input (read operation)	10 MHz 5 MHz	15.0 μ s 18.0 μ s	—
t_{wrd}	Start pulse edge to Neuron Chip releasing the data line	10 MHz 5 MHz	66.6 μ s 72.0 μ s	—
t_{ibd}	Inter-bit delay	10 MHz 5 MHz	61.2 μ s 122.4 μ s	—
t_{ret}	Return from I/O call	—	42.6 μ s	—

Figure 5-32. Touch I/O Object

The leveledetect input object can be used for detection of asynchronous attachments of Touch Memory devices to the Neuron Chip. In such a case, the leveledetect input object is overlaid on top of the Touch I/O object. Refer to the *Neuron C Programmer's Guide* for information on I/O object overlays.

Note that the Touch I/O object can run only at Neuron Chip clock rates of 5 MHz and 10 MHz. This is because the Touch I/O object is designed to meet the Touch Memory timing specification only at those Neuron Chip clock speeds.

For more specific information on the mechanical, electrical, and protocol specifications, refer to the *Book of DS19xx Touch Memory Standards*, available from Dallas Semiconductor Corporation.

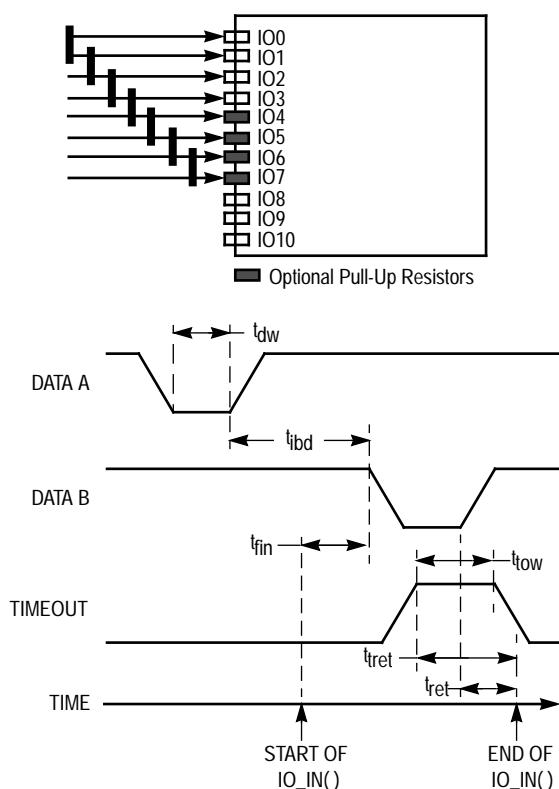
5.5.8 Wiegand Input

This input object provides an easy interface to any card reader supporting the Wiegand standard. Data from the reader is presented to the Neuron Chip through the use of two of its first eight I/O pins, IO0 – IO7. Up to four Wiegand devices may be connected to the Neuron Chip. Data is read MSB first.

Wiegand data starts as a negative-going pulse on one of the two pins selected. One input represents a logical 0 bit and the other pin a logical 1, as selected through the I/O declaration. The bit data on the two lines are mutually exclusive and are spaced at least 150 μ s apart. Figure 5-33 shows the timing relationship of the two data lines with respect to each other and the Neuron Chip.

Any unused I/O pin IO0 – IO7 may be optionally selected as the timeout pin. When the timeout pin goes high, the function aborts and returns. The application processor's watchdog timer is automatically updated during the operation of this input object.

Incoming data on any of the Wiegand input pins is sampled by the Neuron Chip every 200 ns at 10 MHz clock (scales proportionally with the clock frequency). Since the Wiegand data is usually asynchronous, care must be taken in the application program to ensure that this function is called in a timely manner in order that no incoming data is lost.



Symbol	Description	Min	Typ	Max
t_{fin}	Function call to start of second data edge	—	75.6 μ s	—
t_{dw}	Input data width (at 10 MHz)	200 ns	100 μ s	880 ms
t_{ibd}	Inter-bit delay	150 μ s	—	900 μ s
t_{tow}	Timeout pulse width	—	39 μ s	—
t_{tret}	Timeout to function return	—	18.0 μ s	—
t_{ret}	Last data bit to function return	—	74.4 μ s	—

Figure 5-33. Wiegand Input Object

5.6 TIMER/COUNTER OBJECTS

Both the MC143150 and the MC143120 have two 16-bit timer/counters. For the first timer/counter, IO0 is used as the output, and a multiplexer selects one of IO4 – IO7 as the input. The second timer/counter uses IO4 as the input, and IO1 as the output (see Figure 3-8). Note that multiple timer/counter input objects may be declared on different pins within a single application. By calling *io_select*, the application can use the first timer/counter in up to four different input functions. If a timer/counter is configured in one of the output functions, or as a quadrature input, then it can not be reassigned to another timer/counter object in the same application program.

The timing numbers shown in this section are valid for both an explicit I/O call or an implicit I/O call through a *when* clause, and are assumed to be for a Neuron Chip running at 10 MHz.

5.6.1 Timer/Counter Input Objects (Dualslope, Edgelog, Infrared, Ontime, Period, Pulsecount Input, Quadrature, Totalcount)

Input timer/counter objects have the advantage (over non-timer/counter objects) in that input events will be captured even if the application processor is occupied doing something else when the event occurs. A true *when* statement condition for an event being measured by a timer/counter is the completion of the measurement and a value being returned to an event register. If the processor is delayed due to software processing and can not read the register before another event occurs, then the value in the register will reflect the status of the last event. The timer/counters are automatically reset upon completion of a measurement. *The first measured value of a timer/counter is always discarded to eliminate the possibility of a bad measurement after the chip comes out of a reset condition and single events can not be measured with the timer/counters.* Figure 5-34 shows an example of how the timer/counter objects are processed with a Neuron C *when* statement.

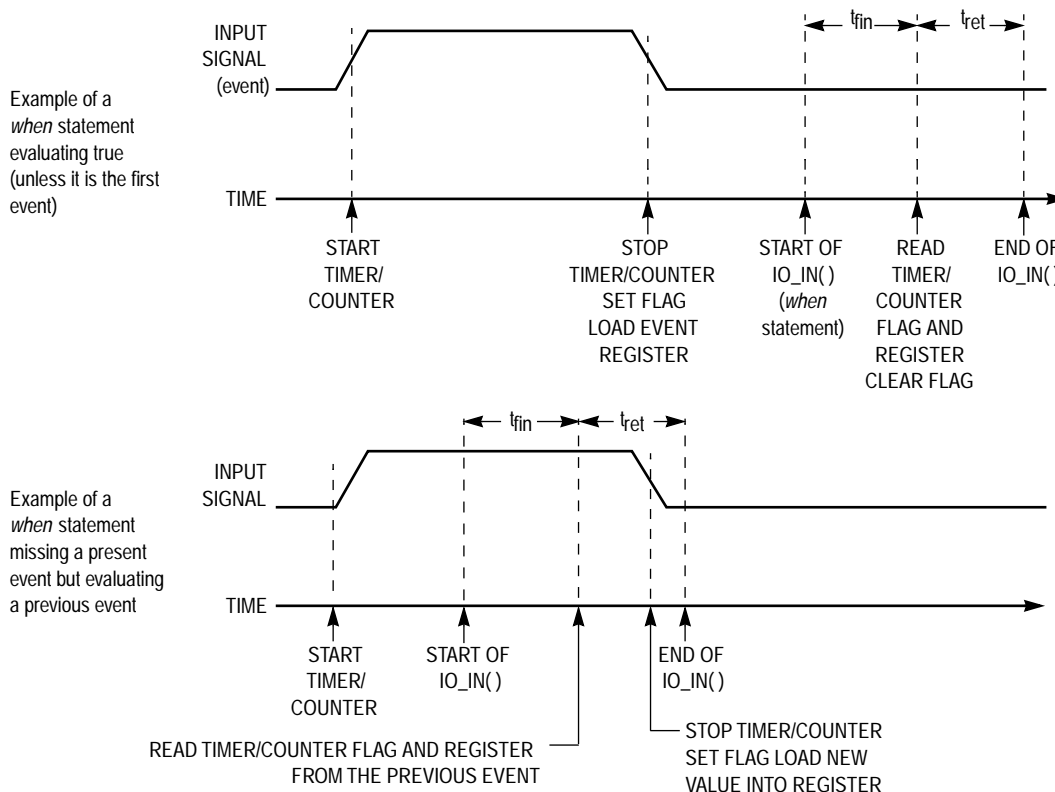
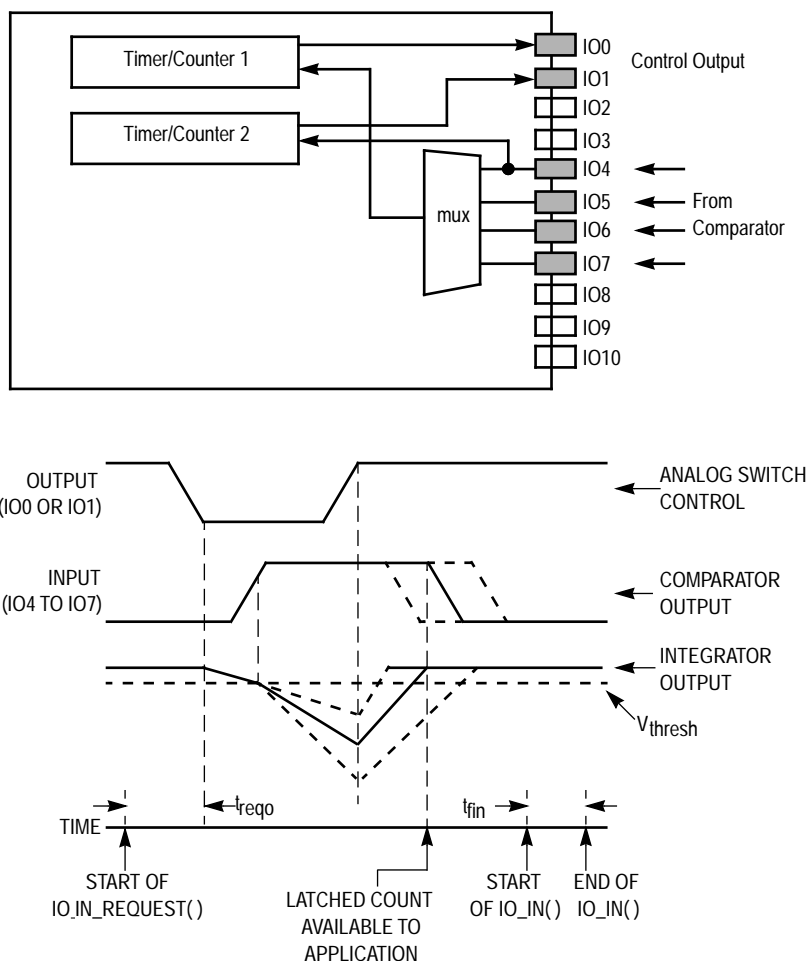


Figure 5-34. *when* Statement Processing Using the Ontime Input Function

5.6.1.1 DUALSLOPE INPUT. This input object uses a timer/counter to control and measure the integration periods of a dualslope integrating analog to digital converter (see Figure 5-35). The timer/counter provides the control output signal and senses a comparator output signal. The control output signal controls an external analog multiplexer which switches between the unknown input voltage and a voltage reference. The timer/counter's input pin is driven by an external comparator which compares the integrator's output with a voltage reference. At the end of conversion, the external comparator will drive a LOW level to one of IO4 – IO7. If external circuitry indicates “end of conversion” with a HIGH level, use the `invert` keyword in the IO declaration.

The resolution and range of the timer/counter period options is shown by Table 5-6 in Section 5.8.

For additional information regarding the dualslope A/D conversion and the Neuron Chip, see EB155/D, *Analog to Digital Conversion with the Neuron Chip*.



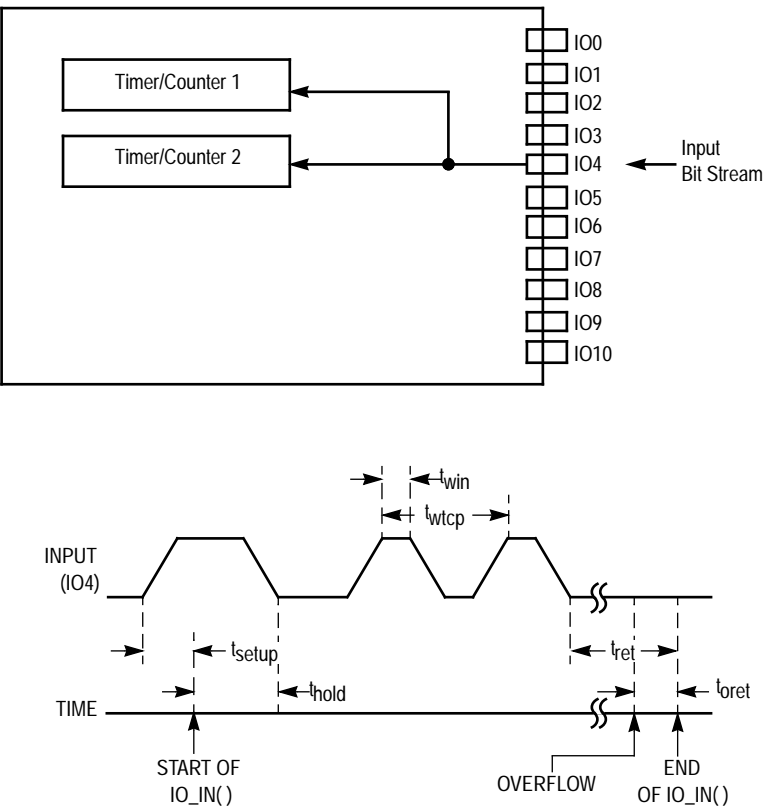
Symbol	Description	Min	Typ	Max
t_{reqo}	<code>io_in_request()</code> to output toggle	—	75.6 μ s	—
t_{fin}	Input function call and return	—	82.8 μ s	—

Figure 5-35. Dualslope Input Object

5.6.1.2 EDGELOG INPUT. The edgelog input object can record a stream of input pulses measuring the consecutive low and high periods at the input and storing them in user-defined storage (see Figure 5-36). The values stored represent the units of clock period between input signal edges, both rising and falling. Both timer/counters of the Neuron Chip are used for this object.

The measurement series starts on the first rising (positive) edge, unless the *invert* keyword is used in the I/O object declaration. The measurement process stops whenever an overflow condition is sensed on either timer/counter.

The resolution and range of the timer/counter period options are shown in Table 5-6 in Section 5.8. This object is useful for analyzing an arbitrarily-spaced stream of input edges (or pulses), such as the output of a UPC bar-code reader or infrared receiver.



Symbol	Description	Min	Typ	Max
t _{setup}	Input data setup	0	—	—
t _{win}	Input pulse width	1 T/C clk	—	65,534 T/C clks
t _{hold}	io_in() call to data input edge for inclusion of that pulse	26.4 μs	—	—
t _{wtcp}	Two consecutive pulse widths	104 μs	—	—
t _{oret}	Return on overflow	—	42.6 μs	—
t _{ret}	Return on count termination	—	49.6 μs	—

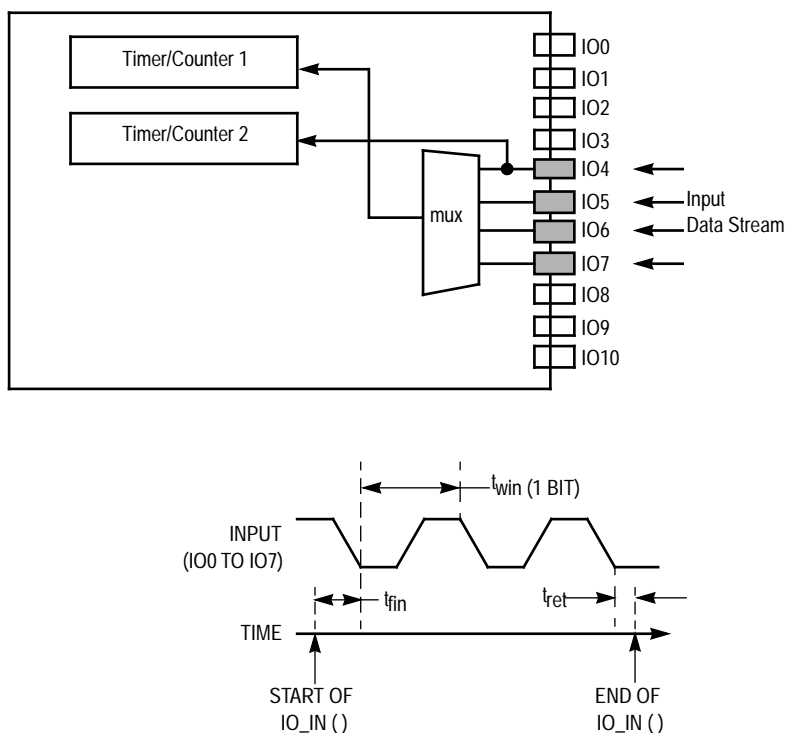
NOTE: T/C clk represents the period of the clock used during the declaration of the I/O object.

Figure 5-36. Edgelog Input Object

5.6.1.3 INFRARED INPUT. The infrared input object is used to capture a stream of data generated by a class of infrared remote control devices (see Figure 5-37). The input to the object is the demodulated series of bits from infrared receiver circuitry. The period of the on/off cycle determines the data bit value, a shorter cycle indicating a 1, and a longer cycle indicating a 0. The actual threshold for the on/off determination is set at the time of the call of the function. The measurements are made between the negative edges of the input bits unless the *invert* keyword is used in the I/O declaration.

The infrared input object, based on the input data stream, generates a buffer containing the values of the bits received. The resolution and range of the timer/counter period options is shown by Table 5-6 in Section 5.8.

This function can be used with an off-the-shelf IR demodulator (e.g., NEC μ PD1913 or Sharp GP1U50X) to quickly develop an infrared interface to the Neuron Chip.

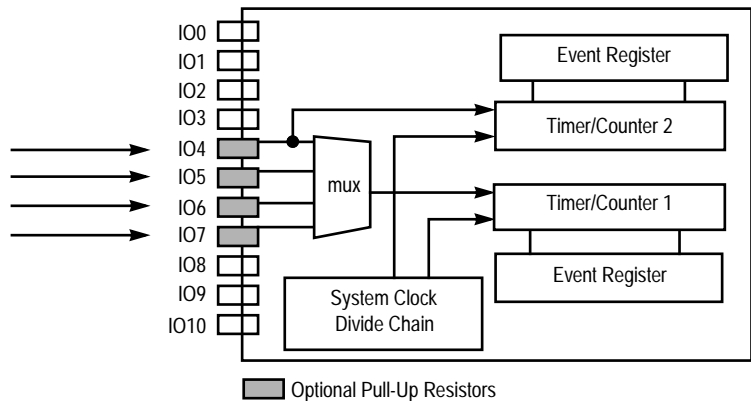


Symbol	Description	Min	Typ	Max
t_{fin}	Function call to start of input sampling	—	82.2 μ s	—
t_{ret}	End of last valid bit to function return	max-period	max-period	—
t_{win}	Minimum input period width	—	93 μ s	—

NOTE: max-period is the timeout period passed to the function at the time of the call.

Figure 5-37. Infrared Input Object

5.6.1.4 ONTIME INPUT. A timer/counter may be configured to measure the time for which its input is asserted. Table 5-6 in Section 5.8 shows the resolution and maximum times for different I/O clock selections. Assertion may be defined as either logic high or logic low. This object may be used as a simple analog-to-digital converter with a voltage-to-time circuit, or for measuring velocity by timing motion past a position sensor. See Figures 5-34 and 5-38.



Reference Figure 5-34

Symbol	Description	Typ @ 10 MHz
t_{fin}	Function call to input sample	86 μ s
t_{ret}	Return from function	52/22 μ s*

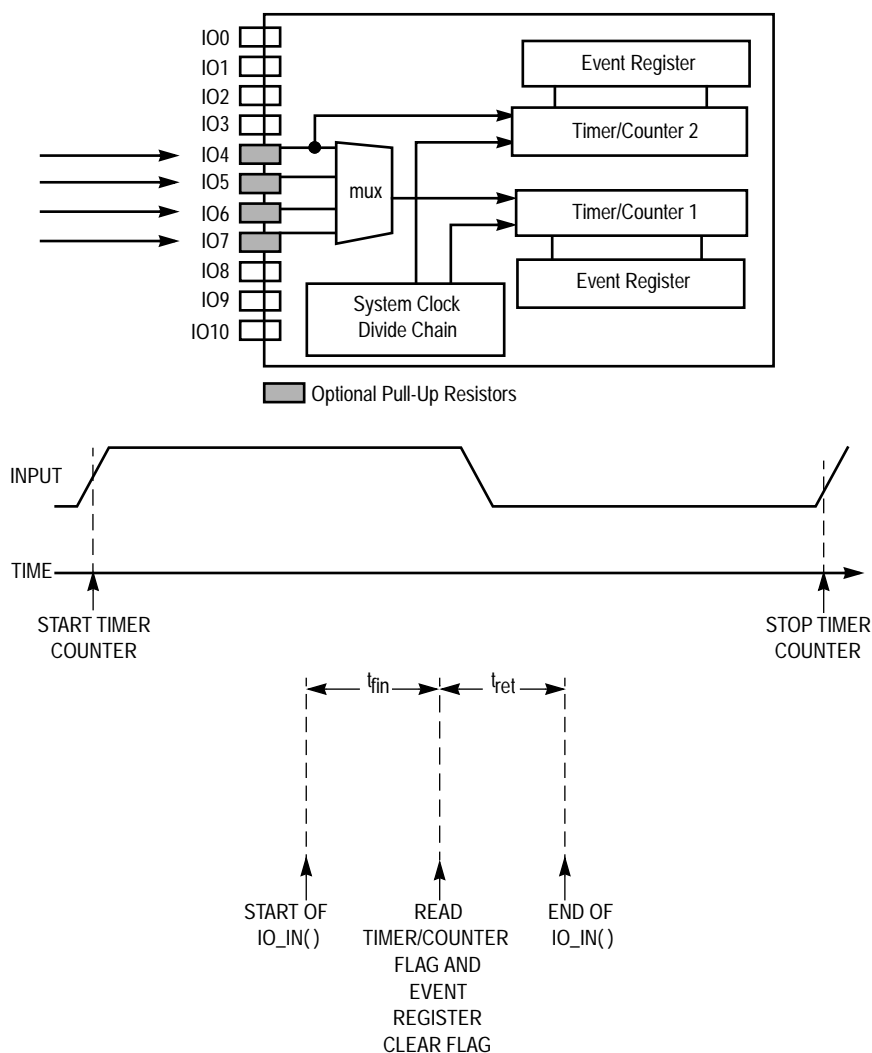
*If the measurement is new, t_{ret} = 52 μ s. If a new time is not being returned, t_{ret} = 22 μ s.

Figure 5-38. Ontime Latency Values

This is a level-sensitive function. The active level of the input signal gates the clock driving the internal counter in the Neuron Chip.

The actual active level of the input depends on whether or not the invert option was used in the declaration of the function block. The default is the high level.

5.6.1.5 PERIOD INPUT. A timer/counter may be configured to measure the period from one rising or falling edge to the next corresponding edge on the input. Table 5-6 in Section 5.8 shows the resolution and maximum time measured for various clock selections. This object is useful for instantaneous frequency or tachometer applications. Analog-to-digital conversion can be implemented using a voltage-to-frequency converter with this object. See Figure 5-39.



Reference Figure 5-34

Symbol	Description	Typ @ 10 MHz
t_{fin}	Function call to input sample	86 μ s
t_{ret}	Return from function	52/22 μ s*

*If the measurement is new, $t_{ret} = 52 \mu$ s. If a new time is not being returned, $t_{ret} = 22 \mu$ s.

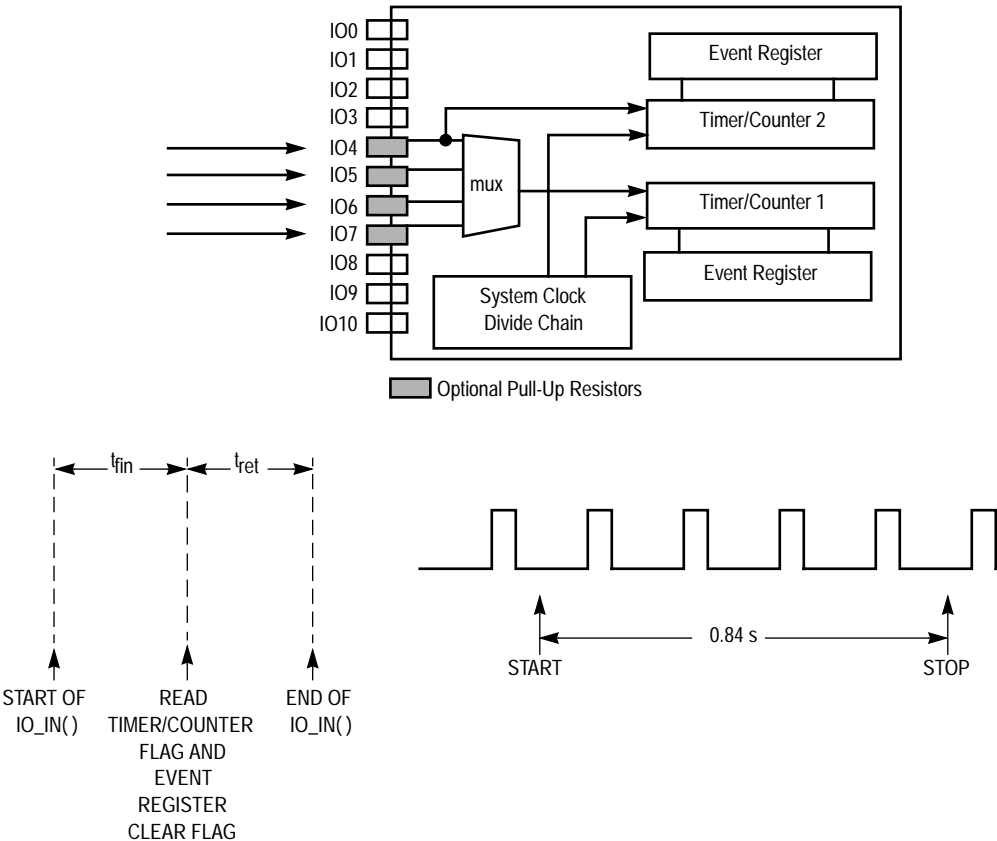
Figure 5-39. Period Input Latency Values

This is an edge-sensitive function. The clock driving the internal counter in the Neuron Chip is free running. The detection of active input edges stops and resets the counter each time.

The actual active edge of the input depends on whether or not the invert option was used in the declaration of the function block. The default is the negative edge.

Since the period function measures the delay between two consecutive active edges, the invert option has no effect on the returned value of the function for a repeating input waveform.

5.6.1.6 PULSECOUNT INPUT. A timer/counter may be configured to count the number of input edges (up to 65,535) in a fixed time (0.8388608 second) at all allowed input clock rates. Edges may be defined as rising or falling. This object is useful for average frequency measurements, or tachometer applications. See Figure 5-40.



Reference Figure 5-34

Symbol	Description	Typ @ 10 MHz
t_{fin}	Function call to input sample	86 μ s
t_{ret}	Return from function	52/22 μ s*

*If the measurement is new, $t_{ret} = 52 \mu$ s. If a new time is not being returned, $t_{ret} = 22 \mu$ s.

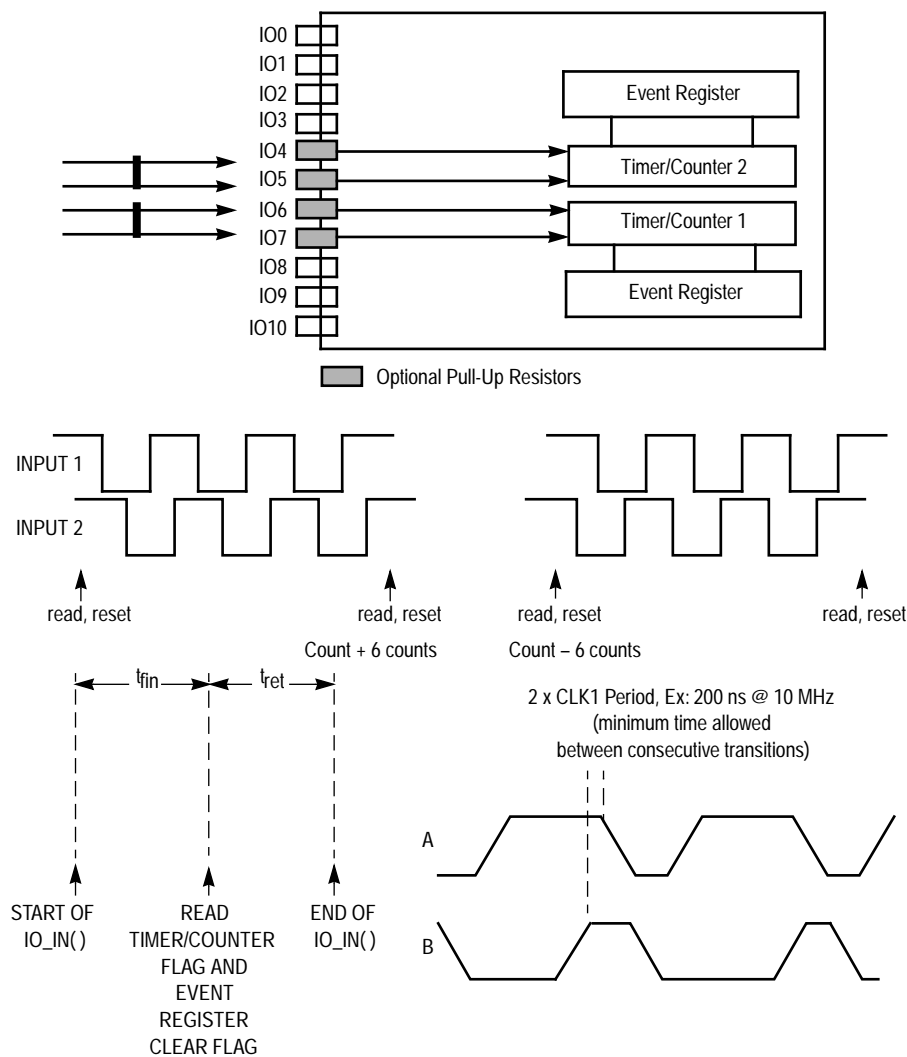
Figure 5-40. Pulse Count Input Latency Values

This is an edge-sensitive function. The clock driving the internal counter in the Neuron Chip is the actual input signal. The counter is reset automatically every 0.839 second.

The internal counter increments with every occurrence of an active input edge. Every 0.839 second, the content of the counter is saved and the counter is then reset to 0. This sequence is repeated indefinitely.

The actual active edge of the input depends on whether or not the invert option was used in the declaration of the function block. The default is the negative edge.

5.6.1.7 QUADRATURE INPUT. A timer/counter may be configured to count transitions of a binary Gray code input on two adjacent input pins. The Gray code is generated by devices such as shaft encoders and optical position sensors which generate the bit pattern (00,01,11,10,00, ...) for one direction of motion and (00,10,11,01,00, ...) for the opposite direction. Reading the value of a quadrature object gives the arithmetic net sum of the number of transitions since the last time it was read (– 16,384 to 16,383). The maximum frequency of the input is one-quarter of the input clock rate, for example 2.5 MHz at the maximum 10 MHz Neuron Chip input clock. Quadrature devices may be connected to timer/counter 1 via pins IO6 and IO7, and timer/counter 2 via pins IO4 and IO5. See Figure 5-41. If the second input transitions low while the first input is low and high while the first input is high, the counter counts up. Otherwise, the count is down.



Reference Figure 5-34

Symbol	Description	Typ @ 10 MHz
t_{fin}	Function call to input sample	90 μ s
t_{ret}	Return from function	88 μ s

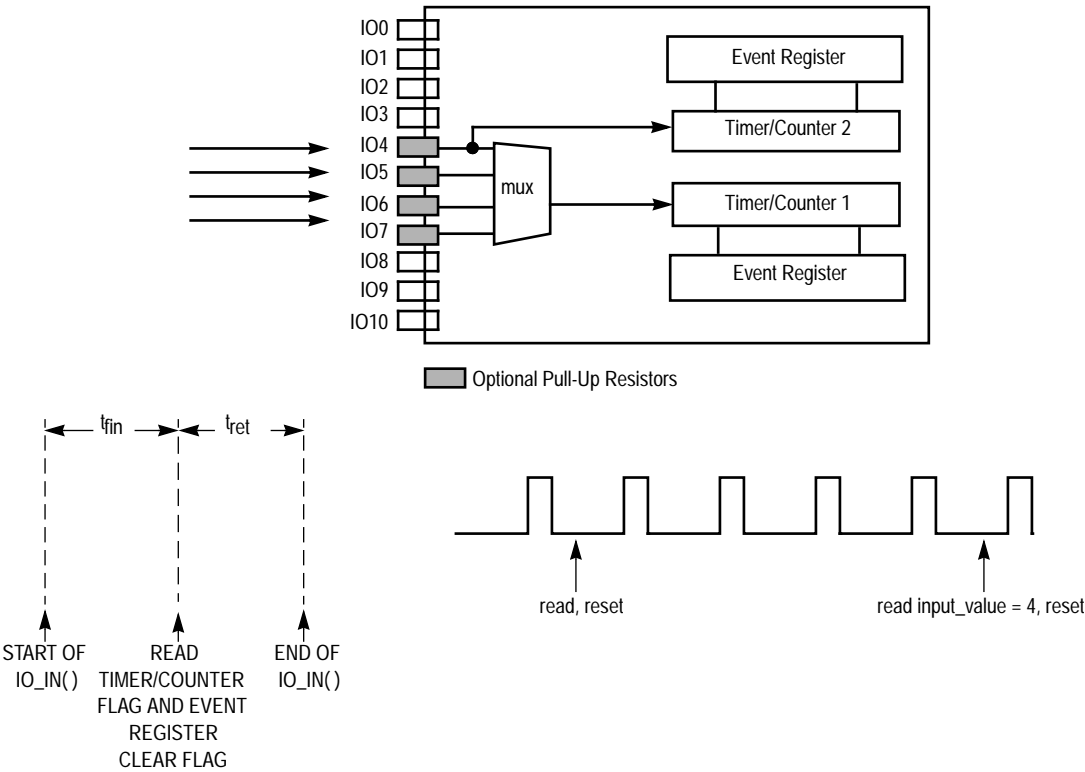
Figure 5-41. Quadrature Input Latency Values

A call to this function returns the current value of the quadrature count since the last read operation. The counter is then reset and ready for the next series of input transitions.

The count returned is a 16-bit signed binary number, capped at $\pm 16K$.

The number shown in the diagram above is the minimum time allowed between consecutive transitions at either input of the quadrature function block. For more information, see EB146/D, *Neuron Chip Quadrature Input Function Interface*.

5.6.1.8 TOTALCOUNT INPUT. A timer/counter may be configured to count input edges, either rising or falling, but not both. Reading the value of a totalcount object gives the number of transitions since the last time it was read (0 to 65,535). Maximum frequency of the input is one-quarter of the input clock rate, for example 2.5 MHz at the maximum 10 MHz Neuron Chip input clock. This object is useful for counting external events such as contact closures, where it is important to keep an accurate running total. See Figure 5-42.



Reference Figure 5-34

Symbol	Description	Typ @ 10 MHz
t_{fin}	Function call to input sample	92 μs
t_{ret}	Return from function	61 μs

Figure 5-42. Totalcount Input Latency Values

A call to this function returns the current value of the totalcount value corresponding to the total number of active clock edges since the last call. The counter is then reset and ready for the next series of input transitions.

The actual active edge of the input depends on whether or not the invert option was used in the declaration of the function block. The default is the negative edge.

5.6.2 Timer/Counter Output Objects (Edgedivide, Frequency, Oneshot, Pulsecount Output, Pulsewidth, Triac, Triggered Count)

5.6.2.1 EDGEDIVIDE OUTPUT. This output object acts as a frequency divider by providing an output frequency on either pin IO0 or IO1, which is a divided-down version of the input frequency applied on pins IO4 – IO7. The object is useful for any divide-by- n operation, where n is passed to the timer/counter object through the application program and can be from 1 to 65,535. The value of 0 forces the output to the off level and halts the timer/counter.

A new divide value will not take effect until after the output toggles, with two exceptions: if the output is initially disabled, the new (non-zero) output will start immediately after t_{fod} ; or, for a new divide value of 0, the output is disabled immediately.

Normally the negative edges of the input sync pulses are the active edge. Using the *invert* keyword in the object declaration makes the positive edge active.

The initial state of the output pin is logic 0 by default. This can also be changed to logic 1 through the object declaration.

Figure 5-43 shows the pinout and timing information for this output object.

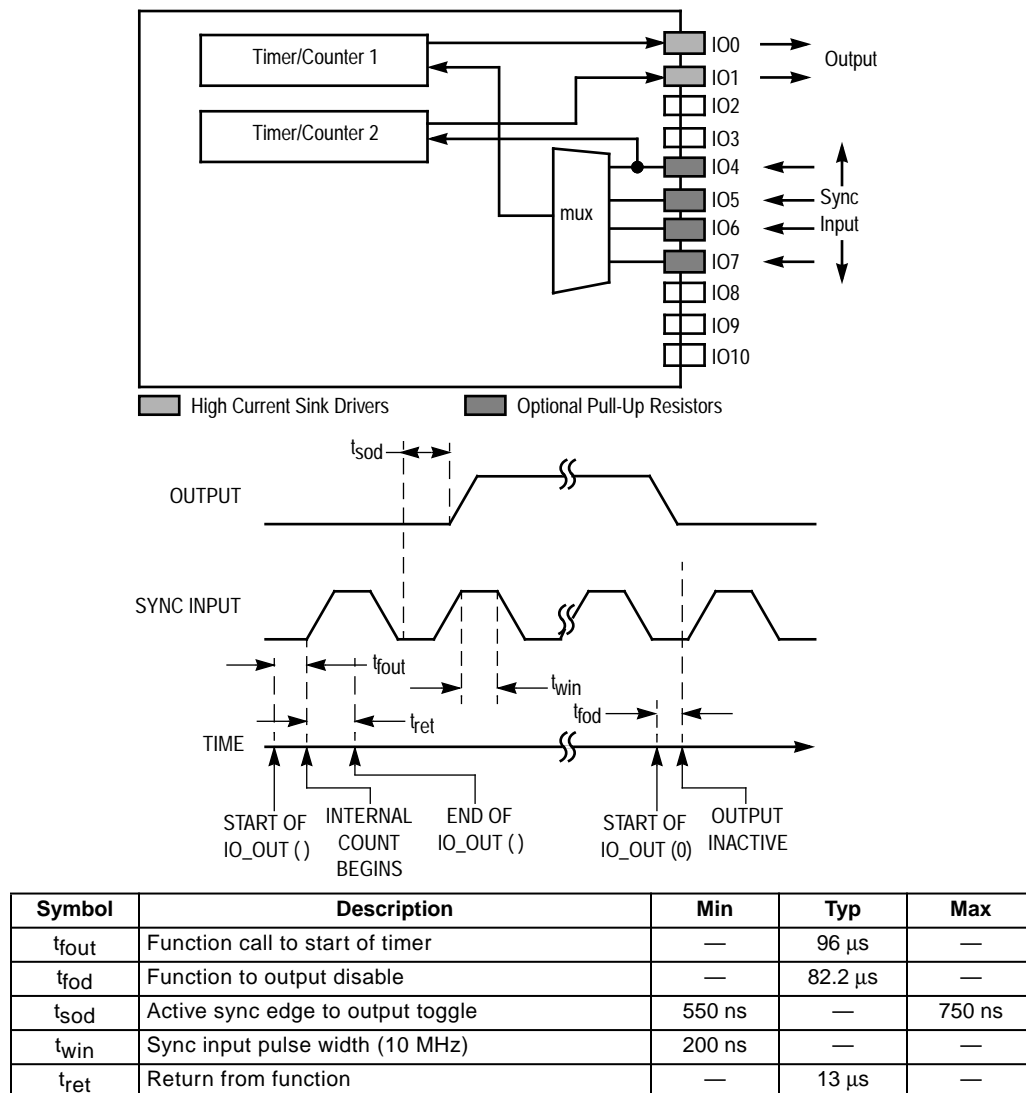


Figure 5-43. Edgedivide Output Object

5.6.2.2 FREQUENCY OUTPUT. A timer/counter may be configured to generate a continuous square wave of 50% duty cycle. Writing a new frequency value to the device takes effect at the end of the current cycle. This object is useful for frequency synthesis to drive an audio transducer, or to drive a frequency to voltage converter to generate an analog output. See Figure 5-44.

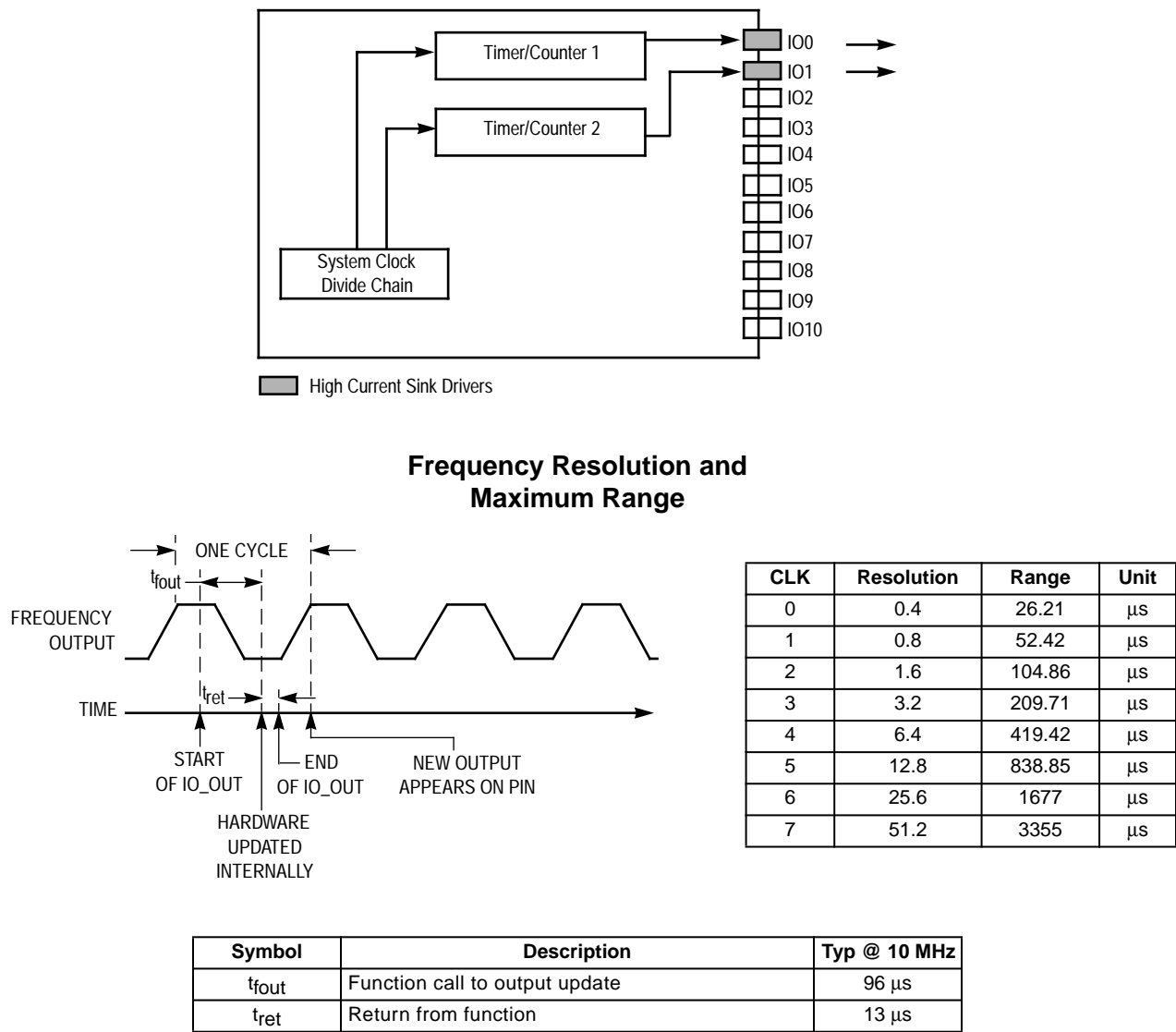
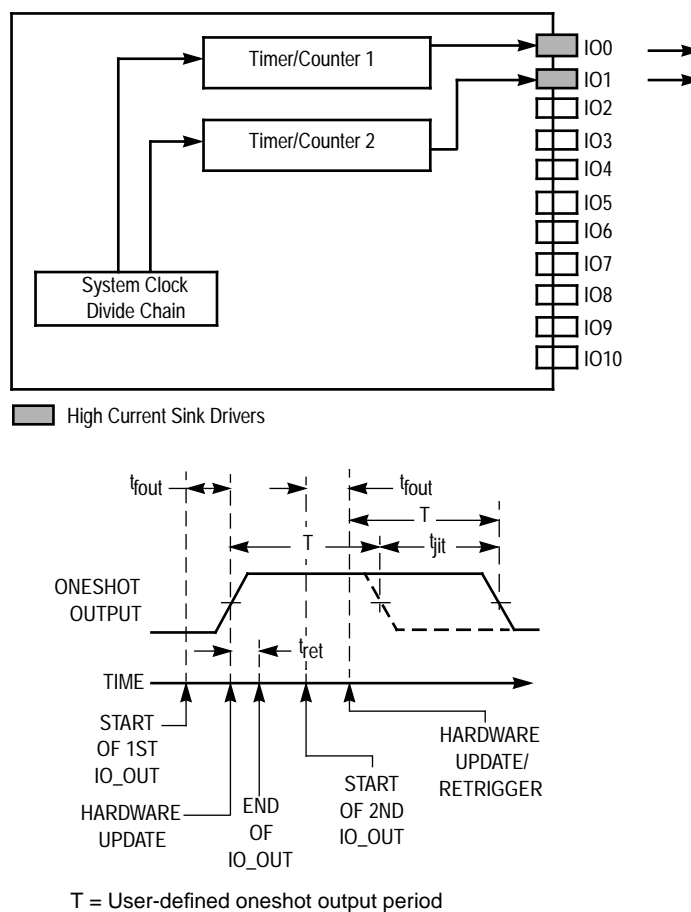


Figure 5-44. Frequency Output Latency Values

A new frequency output value will not take effect until the end of the current cycle. There are two exceptions to this rule. If the output is disabled, the new (non-zero) output will start immediately after t_{fout}. Also, for a new output value of 0, the output is disabled immediately and not at the end of the current cycle.

A disabled output is a logic 0 by default unless the *invert* keyword is used in the I/O object declaration.

5.6.2.3 ONSHOT OUTPUT. A timer/counter may be configured to generate a single pulse of programmable duration. The asserted state may be either logic high or logic low. Retriggering the oneshot before the end of the pulse causes it to continue for the new duration. Table 5-6 in Section 5.8 gives the resolution and maximum time of the pulse for various clock selections. This object is useful for generating a time delay without intervention of the application processor. See Figure 5-45.



Symbol	Description	Typ @ 10 MHz	Max
t_{fout}	Function call to output update	96 μ s	—
t_{ret}	Return from function	13 μ s	—
t_{jit}	Output duration jitter	—	1 timer/counter clock period*

*Timer/counter clock period = $2000 * 2^{(clock)/input\ clock\ (MHz)}$.

Figure 5-45. Oneshot Output Latency Values

While the output is still active, a subsequent call to this function will cause the update to take effect immediately, extending the current cycle. This is, therefore, a retriggerable oneshot function.

5.6.2.4 PULSECOUNT OUTPUT. A timer/counter may be configured to generate a series of pulses. The number of pulses output is in the range 0 to 65,535, and the output waveform is a square wave of 50% duty cycle. This function suspends application processing until the pulse train is complete. The frequency of the waveform may be one of eight values given by Table 5-7 in Section 5.8, with clock select values of 0 through 7. This object is useful for external counting devices that can accumulate pulse trains, such as stepper motors. See Figure 5-46.

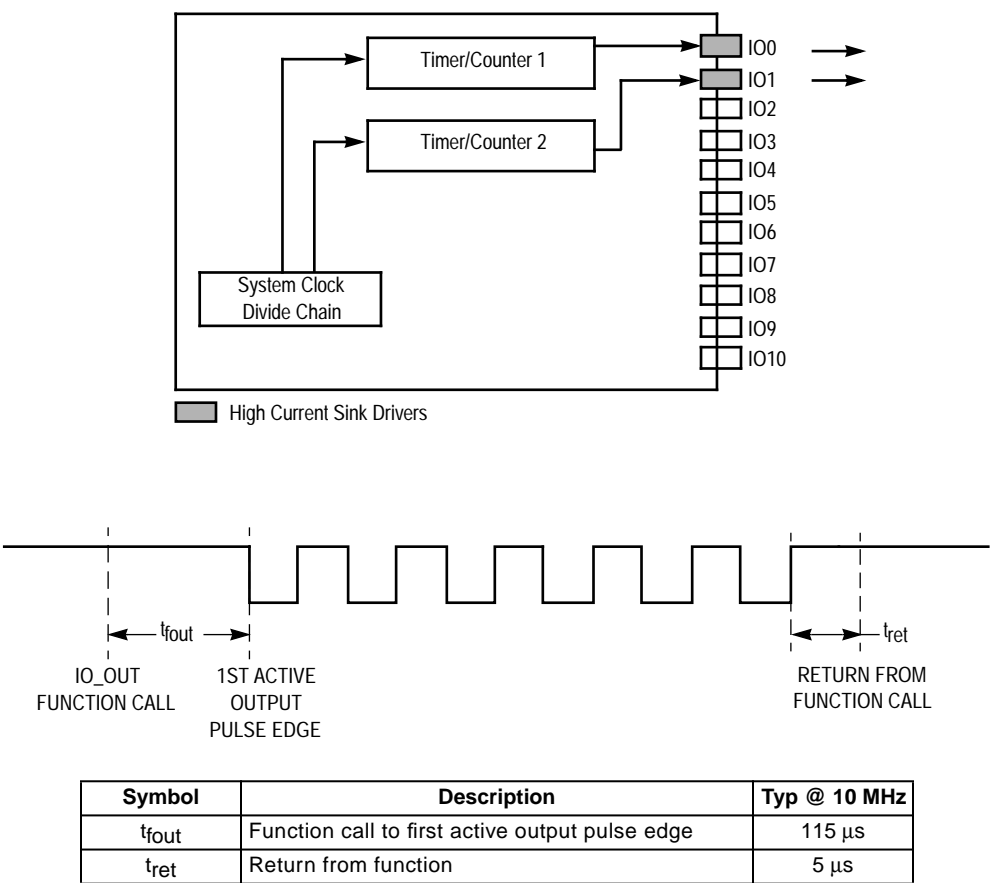


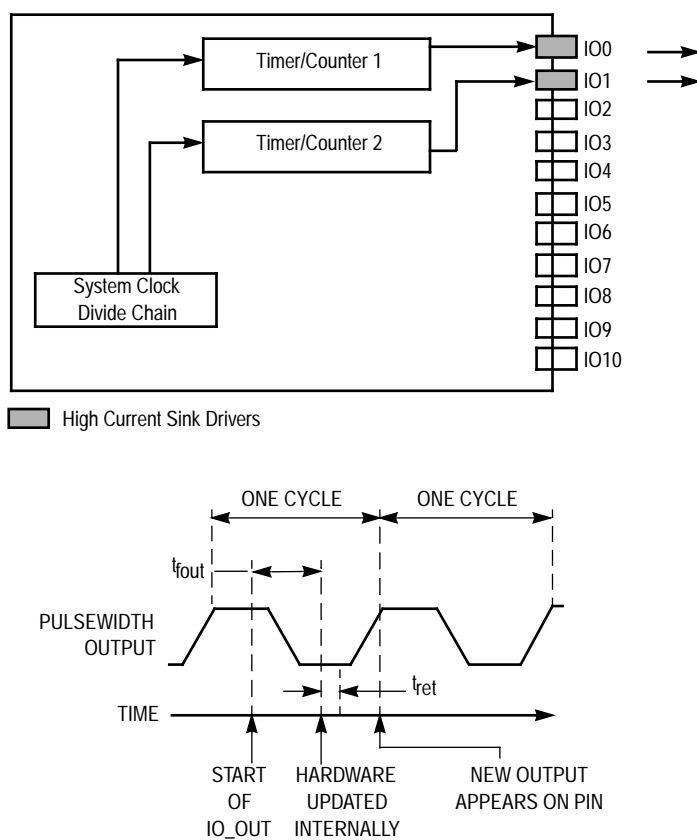
Figure 5-46. Pulsecount Output

The return from this function does not occur until all output pulses have been produced.

t_{fout} is the time from function call to first output pulse. Therefore, the calling of this function ties up the application processor for a period of $N \times (\text{pulse period}) + t_{fout} + t_{ret}$, where N is the number of specified output pulses.

The polarity of the output depends on whether or not the invert option was used in the declaration of the function block. The default is low with high pulses.

5.6.2.5 PULSEWIDTH OUTPUT. A timer/counter may be configured to generate a pulsewidth modulated repeating waveform. In pulsewidth short function, the duty cycle ranges from 0% to 100% (0/256 to 255/256) of a cycle in steps of about 0.4% (1/256). In pulsewidth short function, the frequency of the waveform may be one of eight values given by Table 5-7 in Section 5.8. In pulsewidth long function, the duty cycle ranges from 0% to almost 100% (0/65,536 to 65,535/65,536) of a cycle in steps of 15.25 ppm (1/65,536). In pulsewidth long function, the frequency of the waveform may be one of eight values given by Table 5-8 in Section 5.8. The asserted state of the waveform may be either logic high or logic low. Writing a new pulsewidth value to the device takes effect at the end of the current cycle. A pulsewidth modulated signal provides a simple means of digital-to-analog conversion. See Figure 5-47.



Symbol	Description	Typ @ 10 MHz
t_{fout}	Function call to output update	101 μ s
t_{ret}	Return from function	13 μ s

Figure 5-47. Pulsewidth Output Latency Values

The new output value will not take effect until the end of the current cycle. There are two exceptions to this rule. If the output is disabled, the new (non-zero) output will start immediately after t_{fout} . Also, for a new output value of 0, the output is disabled immediately and not at the end of the current cycle.

A disabled output is a logic 0 by default unless the *invert* keyword is used in the I/O object declaration.

5.6.2.6 TRIAC OUTPUT. On the MC143150 and MC143120, a timer/counter may be configured to control the delay of an output signal with respect to a synchronization input. This synchronization can occur on the rising edge, the falling edge, or both the rising and falling edges of the input signal. For control of ac circuits using a triac device, the sync input is typically a zero-crossing signal, and the pulse output is the triac trigger signal. Table 5-6 in Section 5.8 shows the resolution and maximum range of the delay. See Figure 5-48.

The output gate pulse is gated by an internal clock with a constant period of 25.6 μs (independent of the Neuron Chip input clock). Since the input trigger signal (zero crossing) is asynchronous relative to this internal clock, there is a jitter, t_{jit} , associated with the output gate pulse.

The output gate pulse may be configured to be either a pulse or a level. In the case of a pulse output, the gate signal is a 25 μs wide pulse, independent of the Neuron Chip input clock. In the case of a level output, the output gate signal remains in the active state until the next zero crossing. This, for example, can be useful when the triac is driving a highly inductive load and the gate signal must remain active for the duration of the half cycle. Table 5-6 in Section 5.8 shows the resolution and maximum range of delay.

The actual active edge of the sync input and the triac gate output can be set by using the clock edge or invert parameters, respectively.

Note: Level output does not initialize properly.

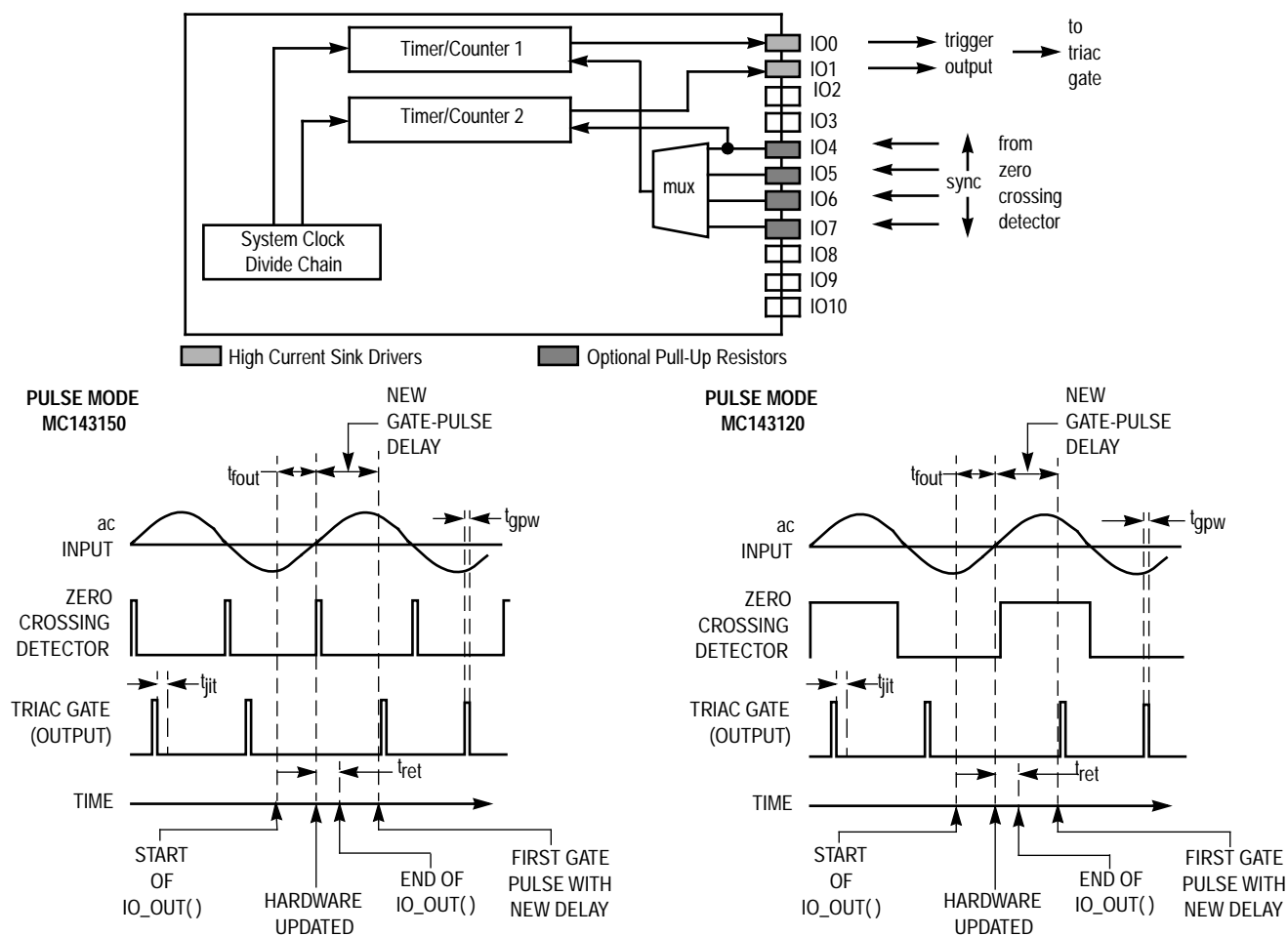
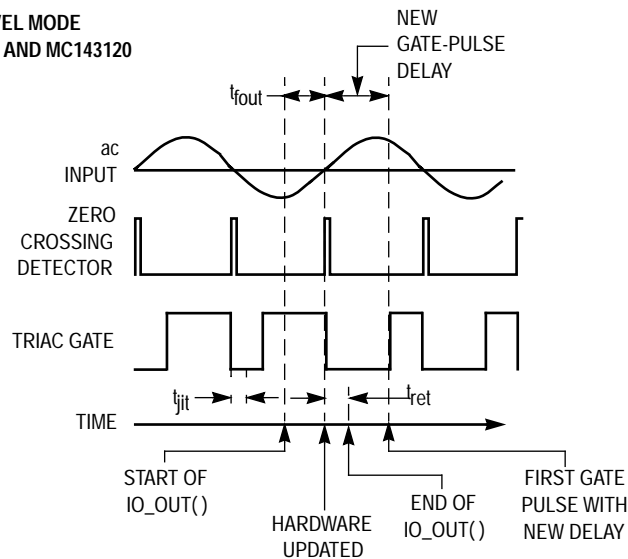


Figure 5-48. Triac Output Latency Values (Sheet 1 of 2)

LEVEL MODE
MC143150 AND MC143120



Symbol	Description	Min	Typ @ 10 MHz	Max
t_{fout}	Function call to first sync bit	—	185 μ s	—
	Min (first sample) Max (timeout)	—	10.2 ms	—
t_{ret}	Return from function	—	17 μ s	—
t_{gpw}	Trigger output pulsewidth	—	25.6 μ s	—
t_{jit}	Gate output jitter	0	—	26.5 μ s

Figure 5-48. Triac Output Latency Values (Sheet 2 of 2)

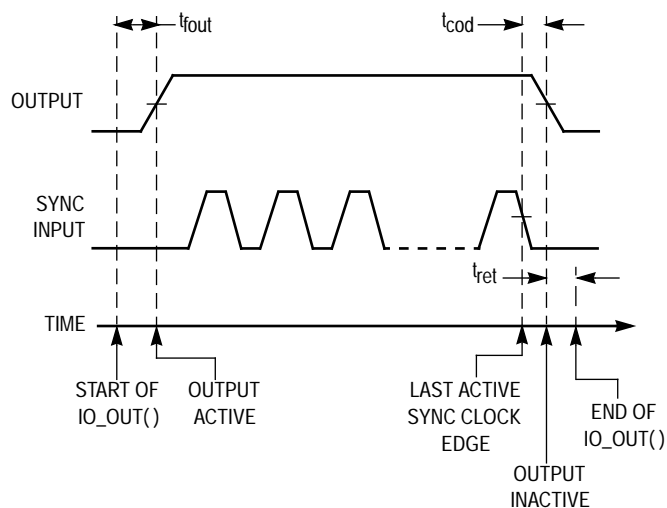
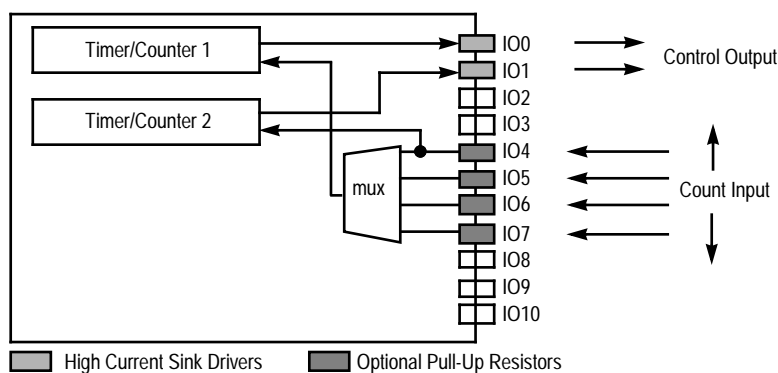
The actual hardware update does not happen until the occurrence of an external active sync clock edge. The internal timer is then enabled and a triac gate pulse is generated after the user-defined period has elapsed. This sequence is repeated indefinitely until another update is made to the triac gate pulse delay value.

t_{fout} (min) refers to the delay from the initiation of the function call to the first sampling of the sync input. In the absence of an active sync clock edge, the input is repeatedly sampled for 10 ms (1/2 wave of a 50 Hz line cycle time), t_{fout} (max), during which the application processor is suspended.

The output gate pulse is gated by an internal clock with a constant period of 25.6 μ s (independent of the Neuron Chip input clock). Since the input trigger signal (zero crossing) is asynchronous relative to this internal clock, there is a jitter, t_{jit} , associated with the output gate pulse.

The actual active edge of the sync input and the triac gate output can be set by using the clock edge or invert parameters, respectively.

5.6.2.7 TRIGGERED COUNT OUTPUT. A timer/counter may be configured to generate an output pulse that is asserted under program control, and de-asserted when a programmable number of input edges (up to 65,535) has been counted on an input pin (IO4 – IO0). Assertion may be either logic high or logic low. This object is useful for controlling stepper motors or positioning actuators which provide position feedback in the form of a pulse train. The drive to the external device is enabled until it has moved the required distance, and then the device is disabled. See Figure 5-49.



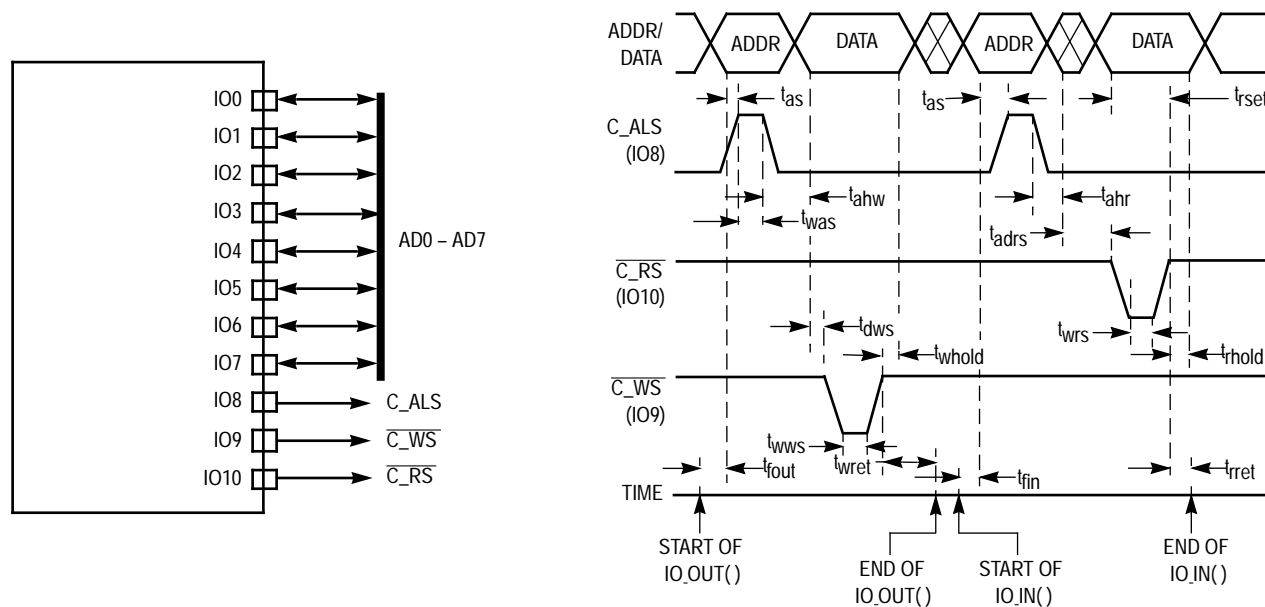
Symbol	Description	Typ @ 10 MHz
t_{fout}	Function call to output pulse	109 μ s
t_{cod}	Last negative sync Clock edge to output inactive	min 550 ns max 750 ns
t_{ret}	Return from function	7 μ s

Figure 5-49. Triggered Count Output Latency Values

The active output level depends on whether or not the invert option was used in the declaration of the function block. The default is high.

5.7 MUXBUS I/O

This I/O object provides another means of performing parallel I/O data transfers between the Neuron Chip and an attached peripheral device or processor (see Figure 5-50). Unlike the parallel I/O object, which makes use of a token-passing scheme for ensuring synchronization, the muxbus I/O enables the Neuron Chip to essentially be in control of all read and write operations at all times. This relieves the burden of protocol handling from the attached device and results in an easier-to-use interface at the expense of data throughput capacity. The data bus remains in the last state used.



NOTE: Data is latched 4.8 μ s after the falling edge of $\overline{C_RS}$.

Symbol	Description	Min	Typ	Max
t_{fout}	io_out() to valid address	—	26.4 μ s	—
t_{as}	Address valid to address strobe	—	10.8 μ s	—
t_{ahw}	Address hold for write	—	4.8 μ s	—
t_{ahr}	Address hold for read	—	6.6 μ s	—
t_{was}	Address strobe width	—	6.6 μ s	—
t_{wrs}	Read strobe width	—	10.8 μ s	—
t_{wws}	Write strobe width	—	10.8 μ s	—
t_{dws}	Data valid to write strobe	—	6.6 μ s	—
t_{rset}	Read setup time	4.8 μ s	—	—
t_{whold}	Write hold time	4.2 μ s	—	—
t_{rhold}	Read hold time	0 μ s	—	—
t_{adrs}	Address disable to read strobe	—	7.2 μ s	—
t_{fin}	io_in() to valid address	—	26.4 μ s	—
t_{rret}	Function return from read	—	4.2 μ s	—
t_{wret}	Function return from write	—	4.2 μ s	—

Figure 5-50. Muxbus I/O Object

5.8 NOTES

Various combinations of I/O pins may be configured as basic inputs or outputs. The application program may optionally specify the initial values of basic outputs. Pins configured as outputs may also be read as inputs, returning the value last written.

The gradient behavior of the timing numbers for different Neuron Chip pins for some of the IO objects is due to the shift-and-mask operation performed by the Neuron Chip firmware.

For dualslope input, edgelog input, ontime input, and period input, the timer/counter returns a value (or a table of values, in the case of edgelog input) in the range 0 to 65,535, representing elapsed times from 0 up to the maximum range given in Table 5-6.

For ontime input, period input, dualslope, edgelog, and infrared; the timer/counter returns a number in the range 0 to 65,535, representing elapsed times from 0 up to the maximum range given in Table 5-6.

For oneshot output, frequency output, and triac output; the timer/counter may be programmed with a number in the range 0 to 65,535. This number represents the waveform ontime for oneshot output, the waveform period for frequency output, and the control period from sync input to pulse/level output for the triac output. Table 5-6 gives the range and resolution for these timer/counter objects at 10 MHz. The clock select value is specified in the declaration of the I/O object in the Neuron C application program, and may be modified at runtime.

Table 5-6. Timer/Counter Resolution and Maximum Range

Clock Select	Oneshot and Triac Outputs; Dualslope, Edgelog, Ontime, and Period Inputs		Frequency Output	
	Resolution (μ s)	Maximum Range (ms)	Resolution (μ s)	Maximum Range (ms)
0	0.2	13.1	0.4	26.2
1	0.4	26.2	0.8	52.4
2	0.8	52.4	1.6	105
3	1.6	105	3.2	210
4	3.2	210	6.4	419
5	6.4	419	12.8	838
6	12.8	839	25.6	1,678
7	25.6	1,678	51.2	3,355

NOTE:

This table is for a 10 MHz input clock. Scale appropriately for other clock rates:

Resolution (μ s) = $2(\text{Clock Select} + n) / \text{Input Clock (MHz)}$

Maximum Range (μ s) = $65535 \times \text{Resolution} (\mu\text{s}) \times n$

$n = 1$ for oneshot and triac output, and dualslope, edgelog, ontime, and period input

$n = 2$ for frequency output.

For 20 MHz operation, the numbers in this table would be half the value shown.

For pulsewidth short output and pulsecount output, Table 5-7 gives the possible choices for pulsetrain repetition frequencies, except that pulsecount can not be used with clock select 0.

Table 5-7. Timer/Counter Square Wave Output

Clock Select (System Clock ÷)	Repetition Rate (Hz)	Repetition Period (μs)	Resolution of Pulse (μs)
0 (÷1) (5 MHz)	19,531	51.2	0.2
1 (÷ 2) (2.5 MHz)	9,766	102.4	0.4
2 (÷ 4) (1.25 MHz)	4,883	204.8	0.8
3 (÷ 8) (625 kHz)	2,441	409.6	1.6
4 (÷ 16) (312.5 kHz)	1,221	819.2	3.2
5 (÷ 32) (156.25 kHz)	610	1,638.4	6.4
6 (÷ 64) (78.125 kHz)	305	3,276.8	12.8
7 (÷ 128) (39.06 kHz)	153	6,553.6	25.6

NOTE:

This table is for 10 MHz input clock. Scale appropriately for other clock rates:

Period (μs) = $512 \times 2^{\text{Clock Select}} / \text{Input Clock (MHz)}$

Frequency (Hz) = $1,000,000 / \text{Period (μs)}$.

For 20 MHz operation, the numbers should be scaled accordingly.

For pulsewidth long output, Table 5-8 gives the possible choices for pulsetrain repetition frequencies.

Table 5-8. Timer/Counter Pulsetrain Output

Clock Select	Frequency (Hz)	Period (ms)
0	76.3	13.1
1	38.1	26.2
2	19.1	52.4
3	9.54	105
4	4.77	210
5	2.38	419
6	1.19	839
7	0.60	1,678

NOTE:

This table is for 10 MHz input clock. Scale appropriately for other clock rates:

Period (ms) = $131.072 \times 2^{\text{Clock Select}} / \text{Input Clock (MHz)}$

Frequency (Hz) = $1,000 / \text{Period (ms)}$

As with all CMOS devices, floating I/O pins can cause excessive current consumption. To avoid this, simply declare all unused I/O pins as bit output. Alternatively, unused I/O pins may be connected to +V_{DD} or GND.

Electrical and Mechanical Specifications

6

SECTION 6

Neuron CHIP ELECTRICAL AND MECHANICAL SPECIFICATIONS

6.1 INTRODUCTION

The Neuron Chip processor family consists of two device types: MC143150 and MC143120. Within these device types are different versions, each with its own electrical and mechanical specifications. When deciding at what speed to operate the MC143150, the cost of the external memory will be an important consideration. There is a cost difference between a 200 ns memory (EPROM access time required at 5 MHz) and a 120 ns memory (10 MHz access time) which is rated over the full industrial temperature range, particularly in surface mount packages.

For multiple memory configurations (external EPROM, EEPROM, flash, and/or SRAM), additional cost savings due to slower memory map decode logic, and lower cost of the memory, can be realized at 5 MHz operation. Power consumption will be 30% – 40% lower at 5 MHz than at 10 MHz. Issues such as required data rate and I/O response time must be considered.

Running the Neuron Chip at a higher clock rate decreases the processing time. Proper prioritizing of *when* statements in the application program and use of timer counter objects can help to minimize latency.

Refer to Section D.2 regarding handling precautions and moisture sensitivity of the various Neuron devices.

NOTE: The MC143150FU, MC143150FU1, and MC143150B1FU devices have been discontinued. **The MC143150B2 is recommended for new designs.**

6.2 ELECTRICAL SPECIFICATIONS

6.2.1 Absolute Maximum Ratings

Parameter	Symbol	Value	Unit
Supply Voltage Range (Referenced to V_{SS})	V_{DD}	– 0.3 to 7.0 V	V
Input Voltage Range (Referenced to V_{SS})	V_{in}	– 0.3 to $V_{DD} + 0.3$	V
Maximum Drain Current	I_{DD}	200	mA
Maximum Source Current	I_{SS}	300	mA
Maximum Power Dissipation	P_D	800	mW
Operating Temperature	T_A	– 40 to + 85	°C
Storage Temperature Range	T_{stg}	– 65 to + 150	°C

6.2.2 Recommended Operating Conditions

(Voltages referenced to V_{SS} , $T_A = -40$ to + 85°C)

Parameter	Symbol	Min	Max	Unit
Supply Voltage MC143120B1/E2/FE2, MC143150B1/B2 MC143120LE2	V_{DD}	4.5 2.8	5.5 3.3	V
TTL Low-Level Input Voltage	V_{IL}	V_{SS}	0.8	V
TTL High-Level Input Voltage	V_{IH}	2.0	V_{DD}	V
CMOS Low-Level Input Voltage	V_{IL}	V_{SS}	0.2 V_{DD}	V
CMOS High-Level Input Voltage	V_{IH}	0.7 V_{DD}	V_{DD}	V
Operating Free-Air Temperature	T_A	– 40	+ 85	°C

6.2.3 Electrical Characteristics

MC143120B1 ($V_{DD} = 4.5 - 5.5 \text{ V}$)

Parameter	Symbol	Min	Typ	Max	Unit
Input Low Voltage IO0 – IO10, D0 – D7, CP0, CP3, CP4, SERVICE CP0, CP1 (Differential) RESET	V_{IL}	— — —	— — —	0.8 Programmable 0.3 V_{DD}	V
Input High Voltage IO0 – IO10, D0 – D7, CP0, CP3, CP4, SERVICE CP0, CP1 (Differential) RESET	V_{IH}	2.0 Programmable $V_{DD} - 0.7$	— — —	— — —	V
Low-Level Output Voltage $I_{out} \leq 20 \mu\text{A}$ Standard Outputs ($I_{OL} = 1.4 \text{ mA}$) (Note 1) High Sink (IO0 – IO3), SERVICE, RESET ($I_{OL} = 20 \text{ mA}$) High Sink (IO0 – IO3), SERVICE, RESET ($I_{OL} = 10 \text{ mA}$) Maximum Sink (CP2, CP3) ($I_{OL} = 40 \text{ mA}$) Maximum Sink (CP2, CP3) ($I_{OL} = 15 \text{ mA}$)	V_{OL}	— — — — — —	— — — — — —	0.1 0.4 0.8 0.4 1.0 0.4	V
High-Level Output Voltage Standard Outputs ($I_{OH} = -1.4 \text{ mA}$) (Note 1) High Sink (IO0 – IO3), SERVICE ($I_{OH} = -1.4 \text{ mA}$) Maximum Source (CP2, CP3) ($I_{OH} = -40 \text{ mA}$) Maximum Source (CP2, CP3) ($I_{OH} = -15 \text{ mA}$)	V_{OH}	$V_{DD} - 0.4$ $V_{DD} - 0.4$ $V_{DD} - 1.0$ $V_{DD} - 0.4$	— — — —	— — — —	V
Hysteresis (Excluding CLK1, RESET)	V_{hys}	175	—	—	mV
Input Current (Excluding Pull-Ups) (V_{SS} to V_{DD}) (Note 2)	I_{in}	—	—	± 10	μA
Pull-up Source Current ($V_{out} = 0 \text{ V}$, Output = High-Z) (Note 2)	I_{pu}	60	—	260	μA
Operating Mode Supply Current (Notes 3, 4) 10 MHz Clock 5 MHz Clock 2.5 MHz Clock 1.25 MHz Clock 0.625 MHz Clock	I_{DD}	— — — — —	14 7.5 4.5 3.2 1.6	N/A	mA
Sleep Mode Supply Current (Notes 3, 4)	$I_{DDsleep}$	—	9	100	μA

NOTES:

- Standard outputs are A0 – A15, D0 – D7, IO4 – IO10, CP0, CP1, CP4, \bar{E} , and R/W. ($\overline{\text{RESET}}$ is a CMOS open drain input/output. CLK2 must have $\leq 15 \text{ pF}$.)
- IO4 – IO7 and $\overline{\text{SERVICE}}$ have configurable pull-ups. $\overline{\text{RESET}}$ has a permanent pull-up.
- Supply current measurement conditions: all outputs under no-load conditions, all inputs $\leq 0.2 \text{ V}$ or $\geq (V_{DD} - 0.2 \text{ V})$, configurable pull-ups off, crystal oscillator clock input, differential receiver disabled. The differential receiver adds approximately $200 \mu\text{A}$ typical and $600 \mu\text{A}$ maximum when enabled. It is enabled on either of the following conditions:
 - Neuron Chip in Operating mode **and** Comm Port in Differential mode.
 - Neuron Chip in Sleep mode **and** Comm Port in Differential mode **and** Comm Port Wake Up not masked.
- Typical values are at midpoint of supply voltage range and 25°C only.

MC143120E2 ($V_{DD} = 4.5 - 5.5 \text{ V}$)

Parameter	Symbol	Min	Typ	Max	Unit
Input Low Voltage IO0 – IO10, CP0, CP3, CP4, $\overline{\text{SERVICE}}$ CP0, CP1 (Differential) RESET	V_{IL}	— — —	— — —	0.8 Programmable 0.3 V_{DD}	V
Input High Voltage IO0 – IO10, CP0, CP3, CP4, $\overline{\text{SERVICE}}$ CP0, CP1 (Differential) RESET	V_{IH}	2.0 Programmable $V_{DD} - 0.7$	— — —	— — —	V
Low-Level Output Voltage $I_{out} \leq 20 \mu\text{A}$ Standard Outputs ($I_{OL} = 1.4 \text{ mA}$) (Note 1) High Sink (IO0 – IO3), $\overline{\text{SERVICE}}$, RESET ($I_{OL} = 20 \text{ mA}$) High Sink (IO0 – IO3), $\overline{\text{SERVICE}}$, RESET ($I_{OL} = 10 \text{ mA}$) Maximum Sink (CP2, CP3) ($I_{OL} = 40 \text{ mA}$) Maximum Sink (CP2, CP3) ($I_{OL} = 15 \text{ mA}$)	V_{OL}	— — — — — —	— — — — — —	0.1 0.4 0.8 0.4 1.0 0.4	V
High-Level Output Voltage $I_{out} \leq 20 \mu\text{A}$ Standard Outputs ($I_{OH} = -1.4 \text{ mA}$) (Note 1) High Sink (IO0 – IO3), $\overline{\text{SERVICE}}$ ($I_{OH} = -1.4 \text{ mA}$) Maximum Source (CP2, CP3) ($I_{OH} = -40 \text{ mA}$) Maximum Source (CP2, CP3) ($I_{OH} = -15 \text{ mA}$)	V_{OH}	$V_{DD} - 0.1$ $V_{DD} - 0.4$ $V_{DD} - 0.4$ $V_{DD} - 1.0$ $V_{DD} - 0.4$	— — — — —	— — — — —	V
Hysteresis (Excluding CLK1, RESET)	V_{hys}	175	—	—	mV
Input Current (Excluding Pull-Ups) (V_{SS} to V_{DD}) (Note 2)	I_{in}	—	—	± 10	μA
Pull-Up Source Current ($V_{out} = 0 \text{ V}$, Output = High-Z) (Note 2)	I_{pu}	60	—	260	μA
Operating Mode Supply Current (Notes 3, 4)	I_{DD}	— — — — —	16 8 5.0 3.5 1.8	N/A	mA
Sleep Mode Supply Current (Notes 3, 4)	$I_{DDsleep}$	—	9	100	μA

NOTES:

- Standard outputs are IO4 – IO10, CP0, CP1, and CP4. ($\overline{\text{RESET}}$ is an open drain input/output. CLK2 must have $\leq 15 \text{ pF}$.)
- IO4 – IO7 and $\overline{\text{SERVICE}}$ have configurable pull-ups. RESET has a permanent pull-up.
- Supply current measurement conditions: all outputs under no-load conditions, all inputs $\leq 0.2 \text{ V}$ or $\geq (V_{DD} - 0.2 \text{ V})$, configurable pull-ups off, crystal oscillator clock input, differential receiver disabled. The differential receiver adds approximately $200 \mu\text{A}$ typical and $600 \mu\text{A}$ maximum when enabled. It is enabled on either of the following conditions:
 - Neuron Chip in Operating mode **and** Comm Port in Differential mode.
 - Neuron Chip in Sleep mode **and** Comm Port in Differential mode **and** Comm Port Wake Up not masked.
- Typical values are at midpoint of supply voltage range and 25°C only.

MC143120FE2 ($V_{DD} = 4.5 - 5.5 \text{ V}$)

Parameter	Symbol	Min	Typ	Max	Unit
Input Low Voltage IO0 – IO10, CP0, CP3, CP4, $\overline{\text{SERVICE}}$ CP0, CP1 (Differential) RESET	V_{IL}	— — —	— — —	0.2 V_{DD} Programmable 0.3 V_{DD}	V
Input High Voltage IO0 – IO10, CP0, CP3, CP4, $\overline{\text{SERVICE}}$ CP0, CP1 (Differential) RESET	V_{IH}	0.7 V_{DD} Programmable $V_{DD} - 0.7$	— — —	— — —	V
Low-Level Output Voltage $I_{out} \leq 20 \mu\text{A}$ Standard Outputs ($I_{OL} = 1.4 \text{ mA}$) (Note 1) High Sink (IO0 – IO3), $\overline{\text{SERVICE}}$, RESET ($I_{OL} = 20 \text{ mA}$) High Sink (IO0 – IO3), $\overline{\text{SERVICE}}$, RESET ($I_{OL} = 10 \text{ mA}$) Maximum Sink (CP2, CP3) ($I_{OL} = 40 \text{ mA}$) Maximum Sink (CP2, CP3) ($I_{OL} = 15 \text{ mA}$)	V_{OL}	— — — — — —	— — — — — —	0.1 0.4 0.8 0.4 1.0 0.4	V
High-Level Output Voltage $I_{out} \leq 20 \mu\text{A}$ Standard Outputs ($I_{OH} = -1.4 \text{ mA}$) (Note 1) High Sink (IO0 – IO3), $\overline{\text{SERVICE}}$ ($I_{OH} = -1.4 \text{ mA}$) Maximum Source (CP2, CP3) ($I_{OH} = -40 \text{ mA}$) Maximum Source (CP2, CP3) ($I_{OH} = -15 \text{ mA}$)	V_{OH}	$V_{DD} - 0.1$ $V_{DD} - 0.4$ $V_{DD} - 0.4$ $V_{DD} - 1.0$ $V_{DD} - 0.4$	— — — — —	— — — — —	V
Hysteresis (Excluding CLK1, RESET)	V_{hys}	175	—	—	mV
Input Current (Excluding Pull-Ups) (V_{SS} to V_{DD}) (Note 2)	I_{in}	—	—	± 10	μA
Pull-Up Source Current ($V_{out} = 0 \text{ V}$, Output = High-Z) (Note 2)	I_{pu}	60	—	260	μA
Operating Mode Supply Current (Notes 3, 4)	I_{DD}	— — — — — —	32 16 8 5.0 3.5 1.8	N/A	mA
Sleep Mode Supply Current (Notes 3, 4)	$I_{DDsleep}$	—	9	100	μA

NOTES:

- Standard outputs are IO4 – IO10, CP0, CP1, and CP4. ($\overline{\text{RESET}}$ is an open drain input/output. CLK2 must have $\leq 15 \text{ pF}$.)
- IO4 – IO7 and $\overline{\text{SERVICE}}$ have configurable pull-ups. $\overline{\text{RESET}}$ has a permanent pull-up.
- Supply current measurement conditions: all outputs under no-load conditions, all inputs $\leq 0.2 \text{ V}$ or $\geq (V_{DD} - 0.2 \text{ V})$, configurable pull-ups off, crystal oscillator clock input, differential receiver disabled. The differential receiver adds approximately 200 μA typical and 600 μA maximum when enabled. It is enabled on either of the following conditions:
 - Neuron Chip in Operating mode **and** Comm Port in Differential mode.
 - Neuron Chip in Sleep mode **and** Comm Port in Differential mode **and** Comm Port Wake Up not masked.
- Typical values are at midpoint of supply voltage range and 25°C only.

MC143120LE2 ($V_{DD} = 2.8 - 3.3 \text{ V}$)

Parameter	Symbol	Min	Typ	Max	Unit
Input Low Voltage IO0 – IO10, CP0, CP3, CP4, $\overline{\text{SERVICE}}$ CP0, CP1 (Differential) RESET	V_{IL}	— — —	— — —	0.2 V_{DD} Programmable 0.3 V_{DD}	V
Input High Voltage IO0 – IO10, CP0, CP3, CP4, $\overline{\text{SERVICE}}$ CP0, CP1 (Differential) RESET	V_{IH}	0.7 V_{DD} Programmable $V_{DD} - 0.7$	— — —	— — —	V
Low-Level Output Voltage $I_{out} \leq 20 \mu\text{A}$ Standard Outputs ($I_{OL} = 1.4 \text{ mA}$) (Note 1) High Sink (IO0 – IO3), $\overline{\text{SERVICE}}$, RESET ($I_{OL} = 12 \text{ mA}$) High Sink (IO0 – IO3), $\overline{\text{SERVICE}}$, RESET ($I_{OL} = 6 \text{ mA}$) Maximum Sink (CP2, CP3) ($I_{OL} = 17 \text{ mA}$) Maximum Sink (CP2, CP3) ($I_{OL} = 12 \text{ mA}$)	V_{OL}	— — — — — —	— — — — — —	0.1 0.4 0.8 0.4 0.6 0.4	V
High-Level Output Voltage $I_{out} \leq 20 \mu\text{A}$ Standard Outputs ($I_{OH} = -1.4 \text{ mA}$) (Note 1) High Sink (IO0 – IO3), $\overline{\text{SERVICE}}$ ($I_{OH} = -1.4 \text{ mA}$) Maximum Source (CP2, CP3) ($I_{OH} = -17 \text{ mA}$) Maximum Source (CP2, CP3) ($I_{OH} = -12 \text{ mA}$)	V_{OH}	$V_{DD} - 0.1$ $V_{DD} - 0.4$ $V_{DD} - 0.4$ $V_{DD} - 0.6$ $V_{DD} - 0.4$	— — — — —	— — — — —	V
Hysteresis (Excluding CLK1, RESET)	V_{hys}	300	—	—	mV
Input Current (Excluding Pull-Ups) (V_{SS} to V_{DD}) (Note 2)	I_{in}	—	—	± 10	μA
Pull-Up Source Current ($V_{out} = 0 \text{ V}$, Output = High-Z) (Note 2)	I_{pu}	10	—	100	μA
Operating Mode Supply Current (Notes 3, 4)	I_{DD}	— — — —	4 2 1 0.8	N/A	mA
Sleep Mode Supply Current (Notes 3, 4)	$I_{DDsleep}$	—	4	TBD	μA

NOTES:

- Standard outputs are IO4 – IO10, CP0, CP1, and CP4. ($\overline{\text{RESET}}$ is an open drain input/output. CLK2 must have $\leq 15 \text{ pF}$.)
- IO4 – IO7 and $\overline{\text{SERVICE}}$ have configurable pull-ups. RESET has a permanent pull-up.
- Supply current measurement conditions: all outputs under no-load conditions, all inputs $\leq 0.2 \text{ V}$ or $\geq (V_{DD} - 0.2 \text{ V})$, configurable pull-ups off, crystal oscillator clock input, differential receiver disabled. The differential receiver adds approximately $120 \mu\text{A}$ typical and $450 \mu\text{A}$ maximum when enabled. It is enabled on either of the following conditions:
 - Neuron Chip in Operating mode **and** Comm Port in Differential mode.
 - Neuron Chip in Sleep mode **and** Comm Port in Differential mode **and** Comm Port Wake Up not masked.
- Typical values are at midpoint of supply voltage range and 25°C only.

MC143150B1 ($V_{DD} = 4.5 - 5.5 \text{ V}$)

Parameter	Symbol	Min	Typ	Max	Unit
Input Low Voltage IO0 – IO10, D0 – D7, CP0, CP3, CP4, $\overline{\text{SERVICE}}$ CP0, CP1 (Differential) RESET	V_{IL}	— — —	— — —	0.8 Programmable $0.3 V_{DD}$	V
Input High Voltage IO0 – IO10, D0 – D7, CP0, CP3, CP4, $\overline{\text{SERVICE}}$ CP0, CP1 (Differential) RESET	V_{IH}	2.0 Programmable $V_{DD} - 0.7$	— — —	— — —	V
Low-Level Output Voltage $I_{out} \leq 20 \mu\text{A}$ Standard Outputs ($I_{OL} = 1.4 \text{ mA}$) (Note 1) High Sink (IO0 – IO3), $\overline{\text{SERVICE}}$, RESET ($I_{OL} = 20 \text{ mA}$) High Sink (IO0 – IO3), $\overline{\text{SERVICE}}$, RESET ($I_{OL} = 10 \text{ mA}$) Maximum Sink (CP2, CP3) ($I_{OL} = 40 \text{ mA}$) Maximum Sink (CP2, CP3) ($I_{OL} = 15 \text{ mA}$)	V_{OL}	— — — — — —	— — — — — —	0.1 0.4 0.8 0.4 1.0 0.4	V
High-Level Output Voltage Standard Outputs ($I_{OH} = -1.4 \text{ mA}$) (Note 1) High Sink (IO0 – IO3), $\overline{\text{SERVICE}}$ ($I_{OH} = -1.4 \text{ mA}$) Maximum Source (CP2, CP3) ($I_{OH} = -40 \text{ mA}$) Maximum Source (CP2, CP3) ($I_{OH} = -15 \text{ mA}$)	V_{OH}	$V_{DD} - 0.4$ $V_{DD} - 0.4$ $V_{DD} - 1.0$ $V_{DD} - 0.4$	— — — —	— — — —	V
Hysteresis (Excluding CLK1, RESET)	V_{hys}	175	—	—	mV
Input Current (Excluding Pull-Ups) (V_{SS} to V_{DD}) (Note 2)	I_{in}	—	—	± 10	μA
Pull-up Source Current ($V_{out} = 0 \text{ V}$, Output = High-Z) (Note 2)	I_{pu}	60	—	260	μA
Operating Mode Supply Current (Notes 3, 4) 10 MHz Clock 5 MHz Clock 2.5 MHz Clock 1.25 MHz Clock 0.625 MHz Clock	I_{DD}	— — — — —	15 8 4.5 2.4 1.5	25 13 7 4.2 2.5	mA
Sleep Mode Supply Current (Notes 3, 4)	$I_{DDsleep}$	—	15	100	μA

NOTES:

- Standard outputs are A0 – A15, D0 – D7, IO4 – IO10, CP0, CP1, CP4, $\overline{\text{E}}$, and R/W. ($\overline{\text{RESET}}$ is a CMOS open drain input/output. CLK2 must have $\leq 15 \text{ pF}$.)
- IO4 – IO7 and $\overline{\text{SERVICE}}$ have configurable pull-ups. RESET has a permanent pull-up.
- Supply current measurement conditions: all outputs under no-load conditions, all inputs $\leq 0.2 \text{ V}$ or $\geq (V_{DD} - 0.2 \text{ V})$, configurable pull-ups off, crystal oscillator clock input, differential receiver disabled. The differential receiver adds approximately $200 \mu\text{A}$ typical and $600 \mu\text{A}$ maximum when enabled. It is enabled on either of the following conditions:
 - Neuron Chip in Operating mode **and** Comm Port in Differential mode.
 - Neuron Chip in Sleep mode **and** Comm Port in Differential mode **and** Comm Port Wake Up not masked.
- Typical values are at midpoint of supply voltage range and 25°C only.

MC143150B2 ($V_{DD} = 4.5 - 5.5 \text{ V}$)

Parameter	Symbol	Min	Typ	Max	Unit
Input Low Voltage IO0 – IO10, D0 – D7, CP0, CP3, CP4, $\overline{\text{SERVICE}}$ CP0, CP1 (Differential) RESET	V_{IL}	— — —	— — 1.8	0.8 Programmable $0.3 V_{DD}$	V
Input High Voltage IO0 – IO10, D0 – D7, CP0, CP3, CP4, $\overline{\text{SERVICE}}$ CP0, CP1 (Differential) RESET	V_{IH}	2.0 Programmable $V_{DD} - 0.7$	— — —	— — —	V
Low-Level Output Voltage $I_{out} \leq 20 \mu\text{A}$ Standard Outputs ($I_{OL} = 1.4 \text{ mA}$) (Note 1) High Sink (IO0 – IO3), $\overline{\text{SERVICE}}$, RESET ($I_{OL} = 20 \text{ mA}$) High Sink (IO0 – IO3), $\overline{\text{SERVICE}}$, RESET ($I_{OL} = 10 \text{ mA}$) Maximum Sink (CP2, CP3) ($I_{OL} = 40 \text{ mA}$) Maximum Sink (CP2, CP3) ($I_{OL} = 15 \text{ mA}$)	V_{OL}	— — — — — —	— — — — — —	0.1 0.4 0.8 0.4 1.0 0.4	V
High-Level Output Voltage $I_{out} \leq 20 \mu\text{A}$ Standard Outputs ($I_{OH} = -1.4 \text{ mA}$) (Note 1) High Sink (IO0 – IO3), $\overline{\text{SERVICE}}$ ($I_{OH} = -1.4 \text{ mA}$) Maximum Source (CP2, CP3) ($I_{OH} = -40 \text{ mA}$) Maximum Source (CP2, CP3) ($I_{OH} = -15 \text{ mA}$)	V_{OH}	$V_{DD} - 0.1$ $V_{DD} - 0.4$ $V_{DD} - 0.4$ $V_{DD} - 1.0$ $V_{DD} - 0.4$	— — — — —	— — — — —	V
Hysteresis (Excluding CLK1, RESET)	V_{hys}	175	—	—	mV
Input Current (Excluding Pull-Ups) (V_{SS} to V_{DD}) (Note 2)	I_{in}	—	—	± 10	μA
Pull-up Source Current ($V_{out} = 0 \text{ V}$, Output = High-Z) (Note 2)	I_{pu}	60	—	260	μA
Operating Mode Supply Current (Notes 3, 4)	I_{DD}	— — — — —	15 8 4.5 2.4 1.5	25 13 7 4.2 2.5	mA
Sleep Mode Supply Current (Notes 3, 4)	$I_{DDsleep}$	—	15	100	μA

NOTES:

- Standard outputs are A0 – A15, D0 – D7, IO4 – IO10, CP0, CP1, CP4, $\overline{\text{E}}$, and $\text{R}/\overline{\text{W}}$. ($\overline{\text{RESET}}$ is a CMOS open drain input/output. CLK2 must have $\leq 15 \text{ pF}$ load.)
- IO4 – IO7 and $\overline{\text{SERVICE}}$ have configurable pull-ups. $\overline{\text{RESET}}$ has a permanent pull-up.
- Supply current measurement conditions: all outputs under no-load conditions, all inputs $\leq 0.2 \text{ V}$ or $\geq (V_{DD} - 0.2 \text{ V})$, configurable pull-ups off, crystal oscillator clock input, differential receiver disabled. The differential receiver adds approximately $200 \mu\text{A}$ typical and $600 \mu\text{A}$ maximum when enabled. It is enabled on either of the following conditions:
 - Neuron Chip in Operating mode **and** Comm Port in Differential mode.
 - Neuron Chip in Sleep mode **and** Comm Port in Differential mode **and** Comm Port Wake Up not masked.
- Typical values are at midpoint of supply voltage range and 25°C only.

LVI Trip Point (V_{DD}) (See Note)

Part Number	Min	Typ	Max	Unit
MC143120LE2 (3 V Operation)	2.4	2.55	2.7	V
MC143120E2, MC143120FE2, and MC143150B2	3.8	4.1	4.4	V

NOTE: The MC143120B1 and MC143150B1 require external LVI.

6.2.4 External Memory Interface Timing — MC143150B1/B2, $V_{DD} \pm 10\%$

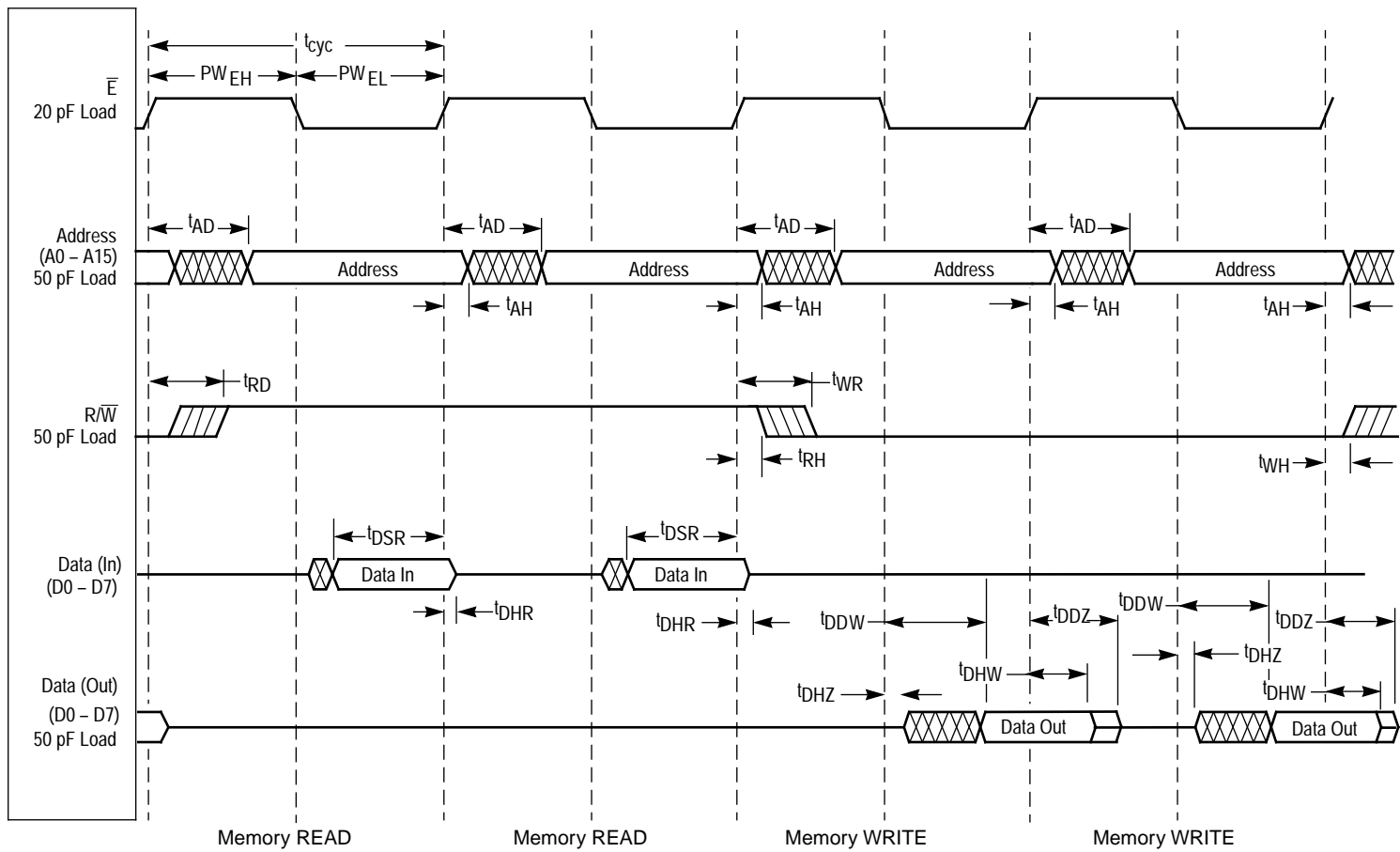
($V_{DD} = 4.5$ to 5.5 V, $T_A = -40$ to $+85^\circ\text{C}$)

Symbol	Parameter	Min	Max	Unit
t_{cyc}	Memory Cycle Time (System Clock Period) (Note 1)	200	3200	ns
PW_{EH}	Pulse Width, \bar{E} High (Note 2)	$t_{cyc}/2 - 5$	$t_{cyc}/2 + 5$	ns
PW_{EL}	Pulse Width, \bar{E} Low	$t_{cyc}/2 - 5$	$t_{cyc}/2 + 5$	ns
t_{AD}	Delay, \bar{E} High to Address Valid MC143150B1 MC143150B2	— —	50 45	ns
t_{AH}	Address Hold Time After \bar{E} High	10	—	ns
t_{RD}	Delay, \bar{E} High to R/\bar{W} Valid Read	—	45	ns
t_{RH}	R/\bar{W} Hold Time Read After \bar{E} High	5	—	ns
t_{WR}	Delay, \bar{E} High to R/\bar{W} Valid Write	—	45	ns
t_{WH}	R/\bar{W} Hold Time Write After \bar{E} High	5	—	ns
t_{DSR}	Read Data Setup Time to \bar{E} High MC143150B1 MC143150B2	20 25	— —	ns
t_{DHR}	Data Hold Time Read After \bar{E} High	0	—	ns
t_{DHW}	Data Hold Time Write After \bar{E} High (Notes 3, 4)	20	—	ns
t_{DDW}	Delay, \bar{E} Low to Data Valid	—	60	ns
t_{DHZ}	Data Three State Hold Time After \bar{E} Low (Note 5)	0	—	ns
t_{DDZ}	Delay, \bar{E} High to Data Three-State (Note 4)	—	60	ns
t_{acc}	External Memory Access Time ($t_{acc} = t_{cyc} - t_{AD} - t_{DSR}$)	130	—	ns

NOTES:

1. $t_{cyc} = 2 \cdot 1/f$, where f is the input clock (CLK1) frequency (10, 5, 2.5, 1.25, or 0.625 MHz). Refer to Clocking System for more details on the CLK1 input clock, including the accuracy requirements (in ppm) and duty cycle requirements.
2. Refer to Figure 6-3 for detailed measurement information.
3. The data hold parameter, t_{DHW} , is measured to the disable levels shown in Figure 6-7, rather than to the traditional data invalid levels.
4. Refer to Figures 6-6 and 6-7 for detailed measurement information.
5. The three-state condition is when the device is not actively driving data. Refer to Figures 6-2 and 6-5 for detailed measurement information.

Figure 6-1. External Memory Interface Timing Diagram



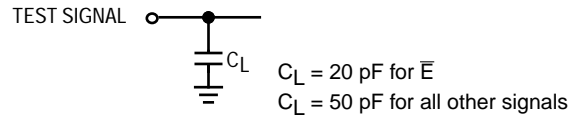


Figure 6-2. Signal Loading for Timing Specifications Unless Otherwise Specified

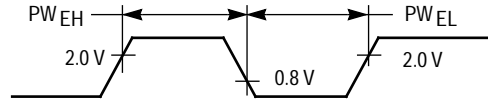
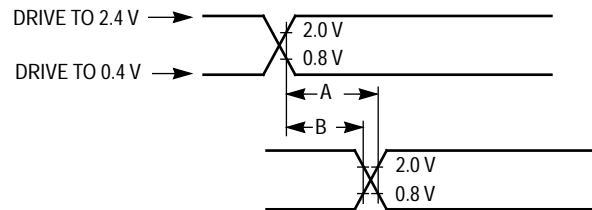


Figure 6-3. Test Point Levels for \bar{E} Pulse Width Measurements



A — Signal valid-to-signal valid specification (maximum or minimum)
 B — Signal valid-to-signal invalid specification (maximum or minimum)

Figure 6-4. Drive Levels and Test Point Levels for Timing Specifications Unless Otherwise Specified

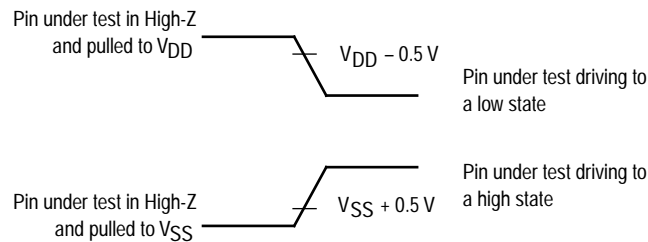


Figure 6-5. Test Point Levels for Three-State-to-Driven Time Measurements

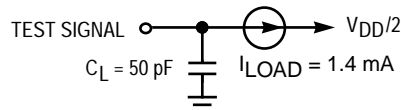
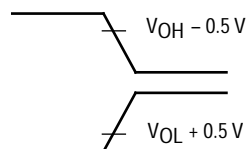


Figure 6-6. Signal Loading for Driven-to-Three-State Time Measurements



V_{OH} — Measured high output drive level
 V_{OL} — Measured low output drive level

Figure 6-7. Test Point Levels for Driven-to-Three-State Time Measurements

6.2.5 Communications Port Programmable Hysteresis Values

(Expressed as differential peak-to-peak voltages in terms of V_{DD})

Hysteresis*	V_{hys} Min	V_{hys} Typ	V_{hys} Max
0	$0.019 V_{DD}$	$0.027 V_{DD}$	$0.035 V_{DD}$
1	$0.040 V_{DD}$	$0.054 V_{DD}$	$0.068 V_{DD}$
2	$0.061 V_{DD}$	$0.081 V_{DD}$	$0.101 V_{DD}$
3	$0.081 V_{DD}$	$0.108 V_{DD}$	$0.135 V_{DD}$
4	$0.101 V_{DD}$	$0.135 V_{DD}$	$0.169 V_{DD}$
5	$0.121 V_{DD}$	$0.162 V_{DD}$	$0.203 V_{DD}$
6	$0.142 V_{DD}$	$0.189 V_{DD}$	$0.236 V_{DD}$
7	$0.162 V_{DD}$	$0.216 V_{DD}$	$0.270 V_{DD}$

*Hysteresis values are under the conditions that the input signal swing is 200 mV greater than the programmed value.

6.2.6 Communications Port Programmable Glitch Filter Values*

[Receiver (end-to-end) filter values expressed as transient pulse suppression times]

Filter (F)	Min	Typ	Max	Unit
0	10	75	140	ns
1	120	410	700	ns
2	240	800	1350	ns
3	480	1500	2600	ns

*Must be disabled if data rate is 1.25 Mbps.

6.2.7 Receiver* (End-to-End) Absolute Asymmetry

(Worst case across hysteresis)

Filter (F)	Max ($ t_{PLH} - t_{PHL} $)	Unit
0	35	ns
1	150	ns
2	250	ns
3	400	ns

*Receiver input, $V_D = V_{CP0} - V_{CP1}$, at least 200 mV greater than hysteresis levels. See Figure 6-8.

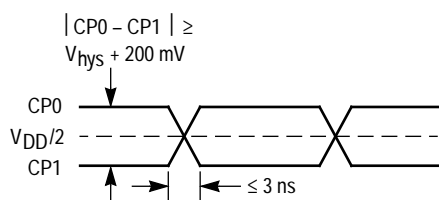


Figure 6-8. Receiver Input Waveform

6.2.8 Differential Receiver (End-to-End) Absolute Symmetry

Filter (F)	Hysteresis (H)	Max ($ t_{PLH} - t_{PHL} $)	Unit
0	0	24	ns

NOTES:

1. CP0 and CP1 inputs each 0.60 V_{p-p}, 1.25 MHz sine wave 180° out of phase with each other as shown in Figure 6-9.

V_{DD} = 5.00 V ± 5%.

2. t_{PLH}: Time from input switching states from low to high to output switching states.

t_{PHL}: Time from input switching states from high to low to output switching states.

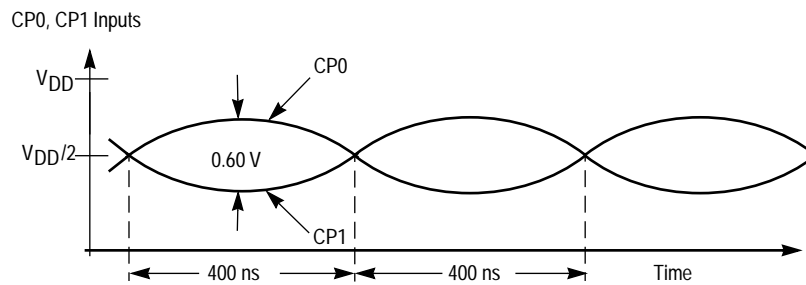


Figure 6-9. Communications Port Signal Input for Table 6.2.8

6.2.9 Differential Transceiver Electrical Characteristics

Characteristic	Min	Max	Unit
Receiver Common Mode Voltage Range to maintain hysteresis as specified in Table 4-4*	1.2	V _{DD} - 2.2	V
Receiver Common Mode Range to operate with unspecified hysteresis	0.9	V _{DD} - 1.75	V
Input Offset Voltage	- 0.05 V _{hys} - 35	0.05 V _{hys} + 35	mV
Propagation Delay (F = 0, V _{ID} = V _{hys} /2 + 200 mV)	—	230 ns	ns
Input Resistance	5	—	MΩ
Wake-Up Time	—	10	μs

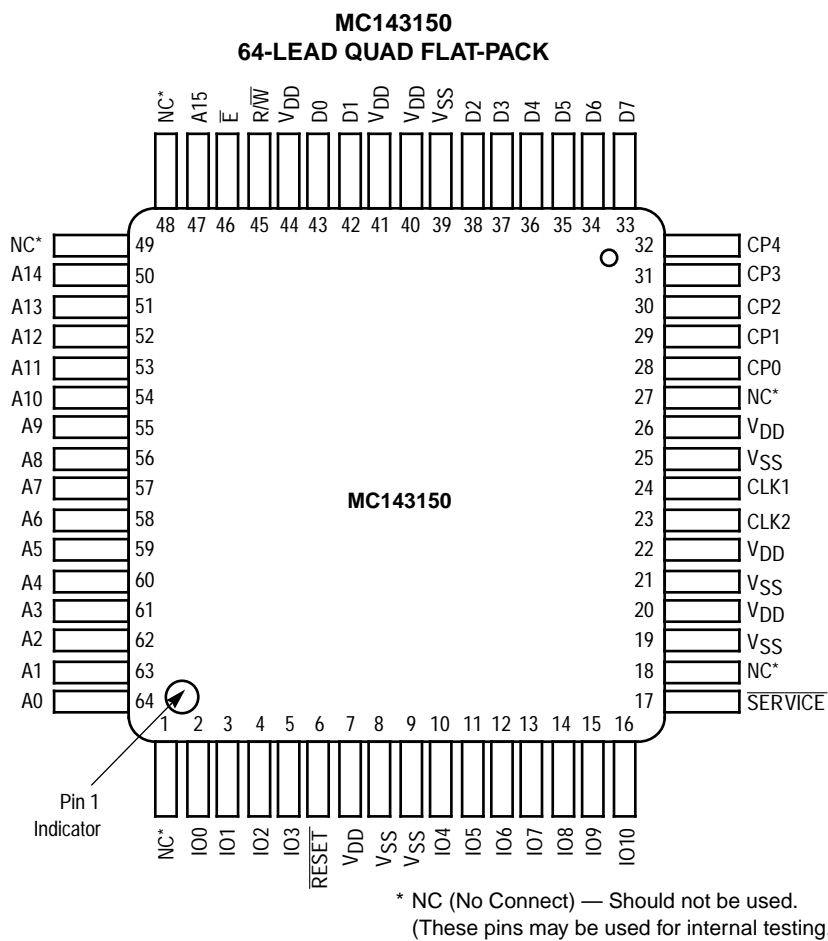
*Common mode voltage is defined as the average value of the waveform at each input at the time switching occurs.

6.3 MECHANICAL SPECIFICATIONS

6.3.1 Pin Descriptions

Pin Name	I/O	Pin Function	MC143150B1/B2 Pin No.	MC143120xx DW Suffix Pin No.	MC143120xx FB Suffix Pin No.
CLK1	Input	Oscillator connection or external clock input.	24	15	15
CLK2	Output	Oscillator connection. Leave open when external clock is input to CLK1. One Load.	23	14	14
RESET	I/O (Built-In Pull-up)	Reset pin (active low).	6	1	40
SERVICE	I/O (Built-In Configurable Pull-up)	Service pin. Indicator output during operation.	17	8	5
IO0 – IO3	I/O	Large current-sink capacity (20 mA). General I/O port.	2, 3, 4, 5	7, 6, 5, 4	4, 3, 2, 43
IO4 – IO7	I/O (Built-In Configurable Pull-up)	General I/O port. One of IO4 – IO7 can be specified as No. 1 timer/counter input with IO0 as output. IO4 can be used as the No. 2 timer/counter input with IO1 as output.	10, 11, 12, 13	3, 30, 29, 28	42, 36, 35, 32
IO8 – IO10	I/O	General I/O port. Can be used for serial communication with other devices.	14, 15, 16	27, 26, 24	31, 30, 27
D0 – D7	I/O	Memory data bus.	43, 42, 38, 37, 36, 35, 34, 33	N/A	N/A
R/W	Output	Read/write control output port for external memory.	45	N/A	N/A
\overline{E}	Output	Control output port for external memory.	46	N/A	N/A
A15 – A0	Output	Address output port.	47, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64	N/A	N/A
V _{DD}	Input	Power input (5 V nom). All V _{DD} pins must be connected together externally.	7, 20, 22, 26, 40, 41, 44	2, 11, 12, 18, 25, 32	9, 10, 19, 29, 38, 41
V _{SS}	Input	Power input (0 V, GND). All V _{SS} pins must be connected together externally.	8, 9, 19, 21, 25, 39	9, 10, 13, 16, 23, 31	7, 8, 13, 16, 26, 37
CP0 – CP4	Communication Network Interface	Bidirectional port that supports communications protocols by specifying mode.	28, 29, 30, 31, 32	19, 20, 17, 21, 22	20, 21, 18, 24, 25
NC	—	No connect. These pins may be used for internal testing.	1, 18, 27, 48, 49	N/A	1, 6, 11, 12, 17, 22, 23, 28, 33, 34, 39, 44

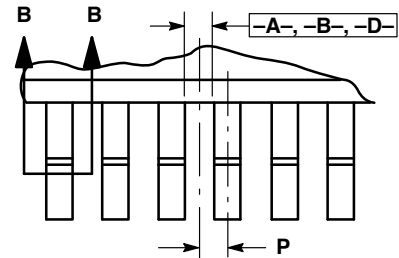
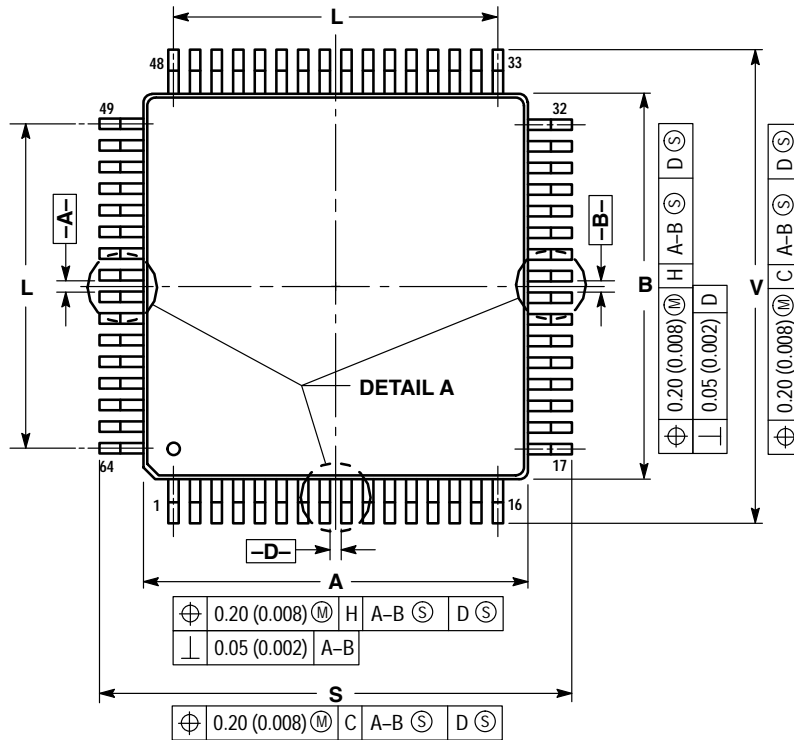
6.3.2 MC143150 Pin Assignment



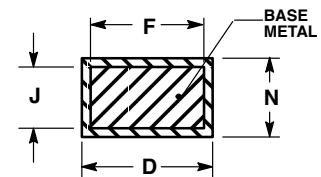
The larger dimple at the bottom left of the marking indicates pin 1.

6.3.3 MC143150 Package Dimensions

64-LEAD PQFP CASE 840C-04



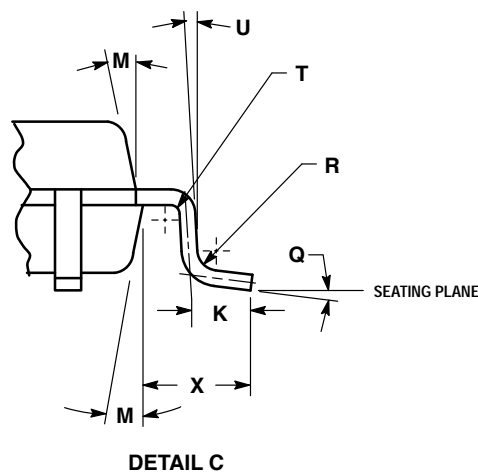
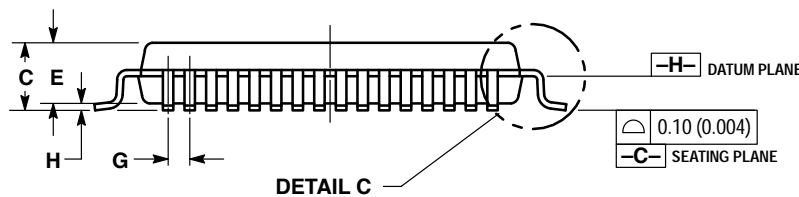
DETAIL A



SECTION B-B

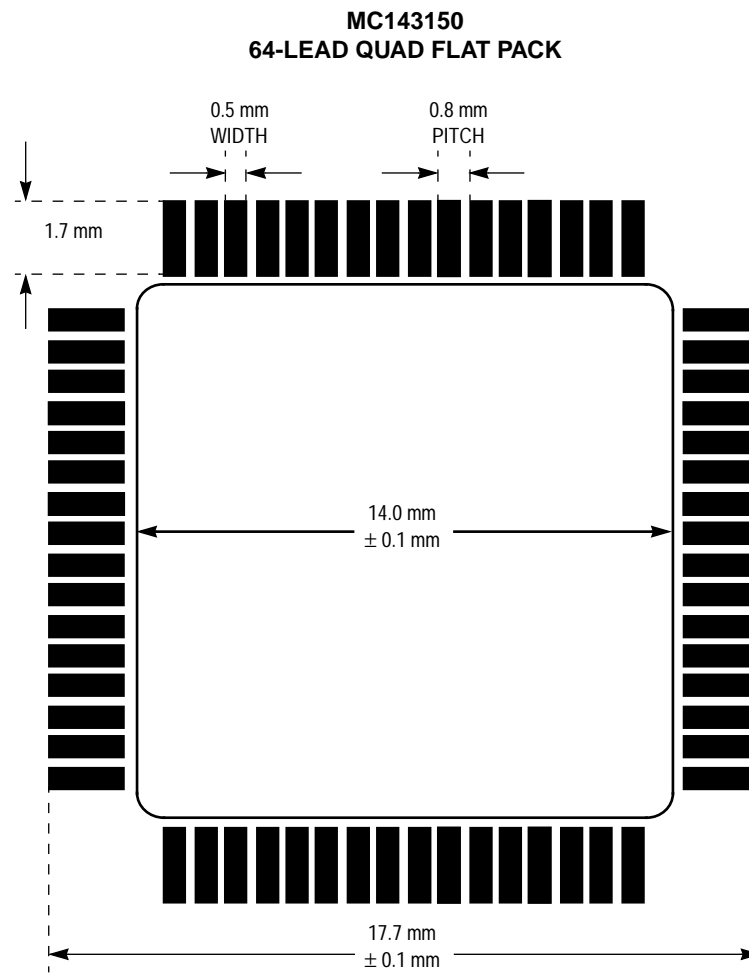
NOTES:

1. DIMENSIONING AND TOLERANCING PER ANSI Y14.5M, 1982.
2. CONTROLLING DIMENSION: MILLIMETER.
3. DATUM PLANE -H- IS LOCATED AT BOTTOM OF LEAD AND IS COINCIDENT WITH THE LEAD WHERE THE LEAD EXITS THE PLASTIC BODY AT THE BOTTOM OF THE PARTING LINE.
4. DATUMS A-B AND -D- TO BE DETERMINED AT DATUM PLANE -H-.
5. DIMENSIONS S AND V TO BE DETERMINED AT SEATING PLANE -C-.
6. DIMENSIONS A AND B DO NOT INCLUDE MOLD PROTRUSION. ALLOWABLE PROTRUSION IS 0.25 (0.010) PER SIDE. DIMENSIONS A AND B DO INCLUDE MOLD MISMATCH AND ARE DETERMINED AT DATUM PLANE -H-.
7. DIMENSION D DOES NOT INCLUDE DAMBAR PROTRUSION. DAMBAR PROTRUSION SHALL NOT CAUSE THE D DIMENSION TO EXCEED 0.53 (0.021). DAMBAR CANNOT BE LOCATED ON THE LOWER RADIUS OR THE FOOT.
8. DIMENSION K IS TO BE MEASURED FROM THE THEORETICAL INTERSECTION OF LEAD FOOT AND LEG CENTERLINES.

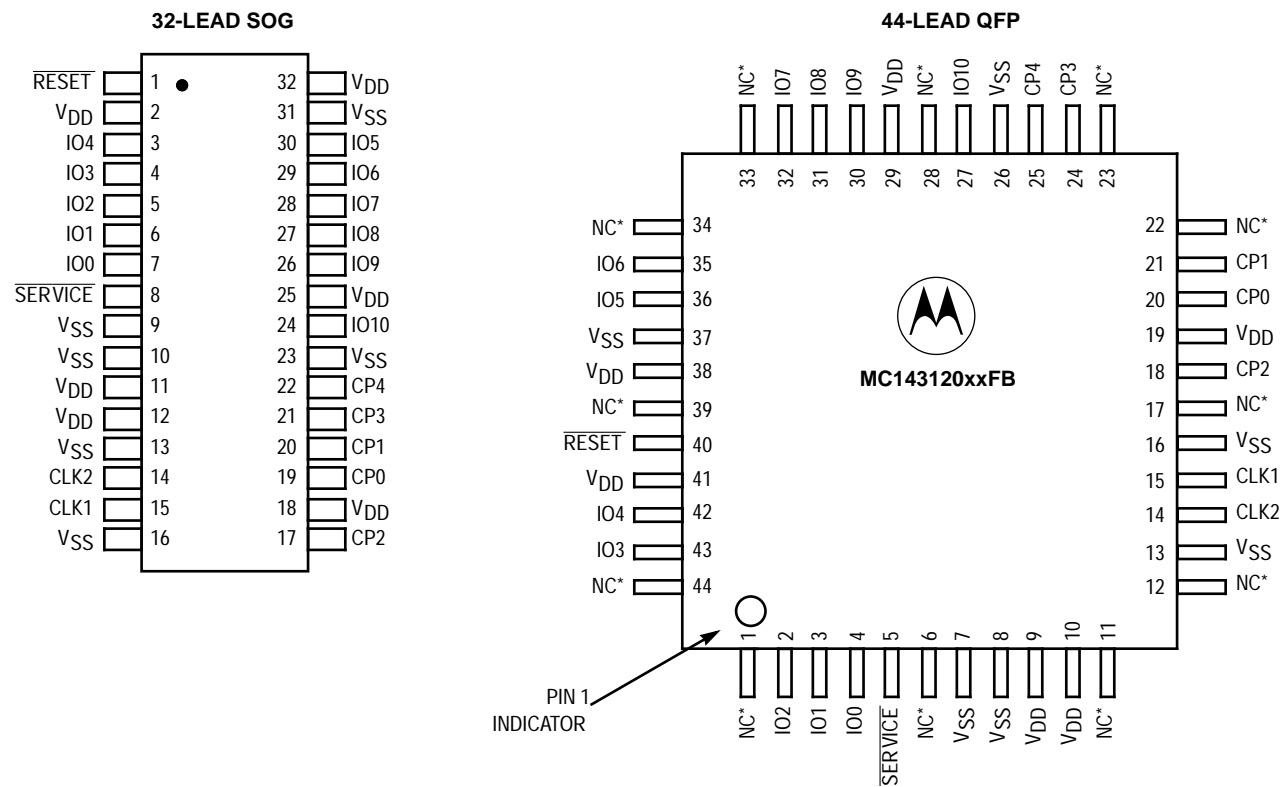


DIM	MILLIMETERS		INCHES	
	MIN	MAX	MIN	MAX
A	13.90	14.10	0.547	0.555
B	13.90	14.10	0.547	0.555
C	2.07	2.46	0.081	0.097
D	0.30	0.45	0.012	0.018
E	2.00	2.40	0.079	0.094
F	0.30	---	0.012	---
G	0.80 BSC	---	0.031 BSC	---
H	0.067	0.250	0.003	0.010
J	0.130	0.230	0.005	0.090
K	0.50	0.66	0.020	0.026
L	12.00 REF	---	0.472 REF	---
M	5 °	10 °	5 °	10 °
N	0.130	0.170	0.005	0.007
P	0.40 BSC	---	0.016 BSC	---
Q	2 °	8 °	2 °	8 °
R	0.13	0.30	0.005	0.012
S	16.20	16.60	0.638	0.654
T	0.20 REF	---	0.008 REF	---
U	0 °	---	0 °	---
V	16.20	16.60	0.638	0.654
X	1.10	1.30	0.043	0.051

6.3.4 MC143150 Pad Layout



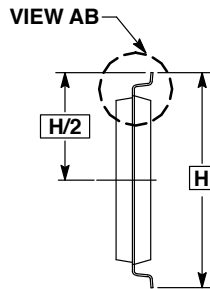
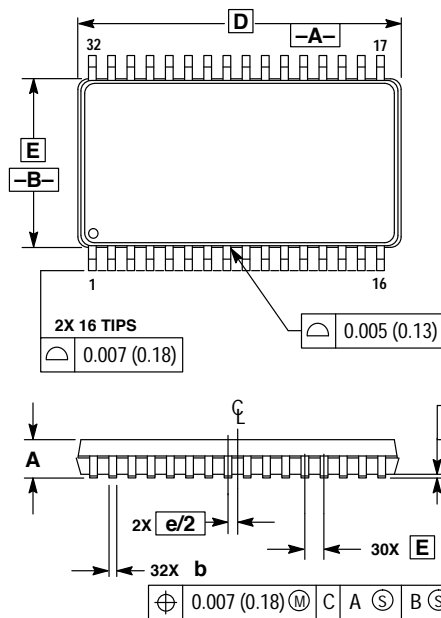
6.3.5 MC143120 Pin Assignments



* NC (No Connect) — Should not be used.
These pins may be used for internal testing.

6.3.6 MC143120 Package Dimensions

DW SUFFIX SOG PACKAGE CASE 1116-01



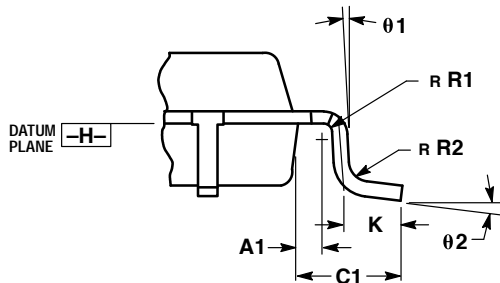
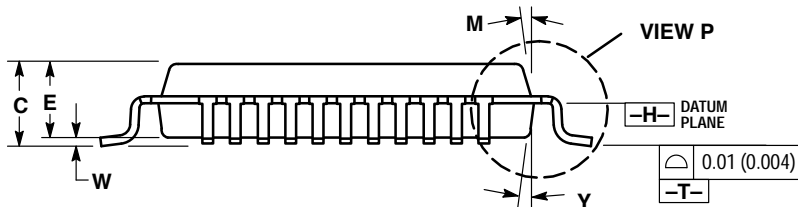
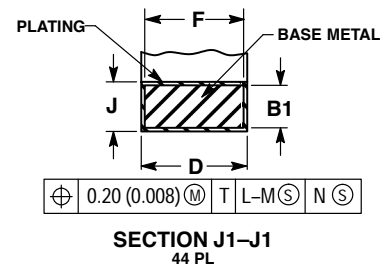
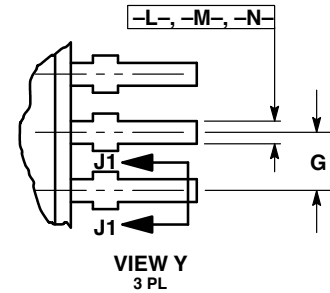
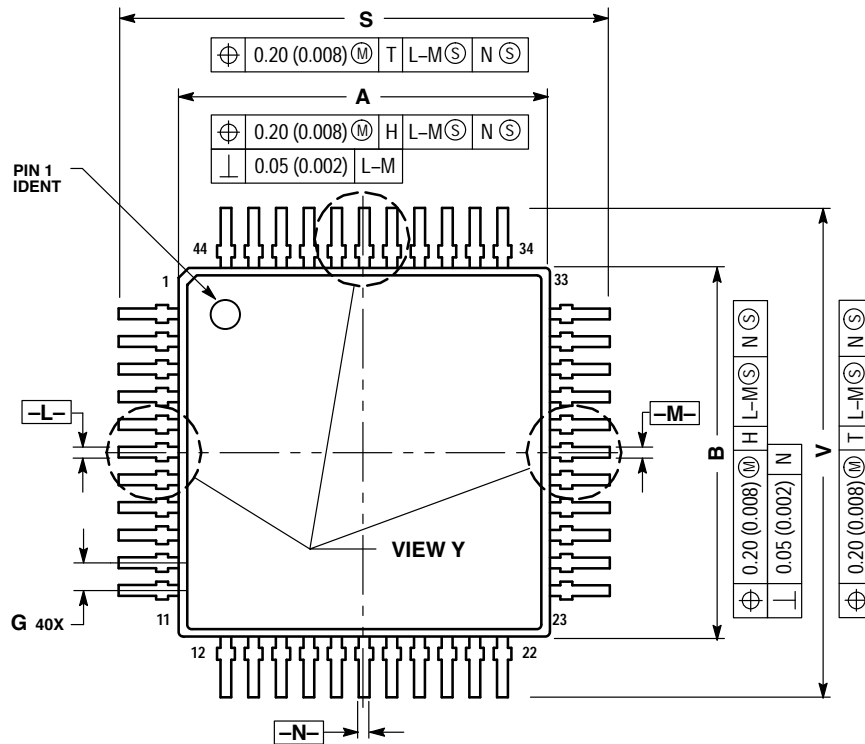
NOTES:

1. DIMENSIONING AND TOLERANCING PER ANSI Y14.5M, 1982.
2. CONTROLLING DIMENSION: INCHES.
3. DIMENSIONS D AND E DO NOT INCLUDE MOLD PROTRUSION. MAXIMUM MOLD PROTRUSIONS SHALL NOT EXCEED 0.006 (0.15) PER SIDE.
4. DIMENSION b DOES NOT INCLUDE DAMBAR PROTRUSION. DAMBAR PROTRUSION SHALL NOT CAUSE THE LEAD WIDTH TO EXCEED 0.026 (0.65).

DIM	INCHES		MILLIMETERS	
	MIN	MAX	MIN	MAX
A	0.090	0.100	2.29	2.54
A1	0.004	0.010	0.10	0.25
A2	0.086	0.090	2.18	2.29
b	0.014	0.020	0.35	0.51
C	0.004	0.009	0.10	0.22
D	0.825 BSC		20.96 BSC	
E	0.430 BSC		10.92 BSC	
e	0.050 BSC		1.27 BSC	
H	0.560 BSC		14.22 BSC	
L	0.021	0.041	0.33	1.04
L1	0.120 REF		3.048 REF	
θ	0°	8°	0°	8°

VIEW AB

FB SUFFIX
PQFP PACKAGE
CASE 824E-02



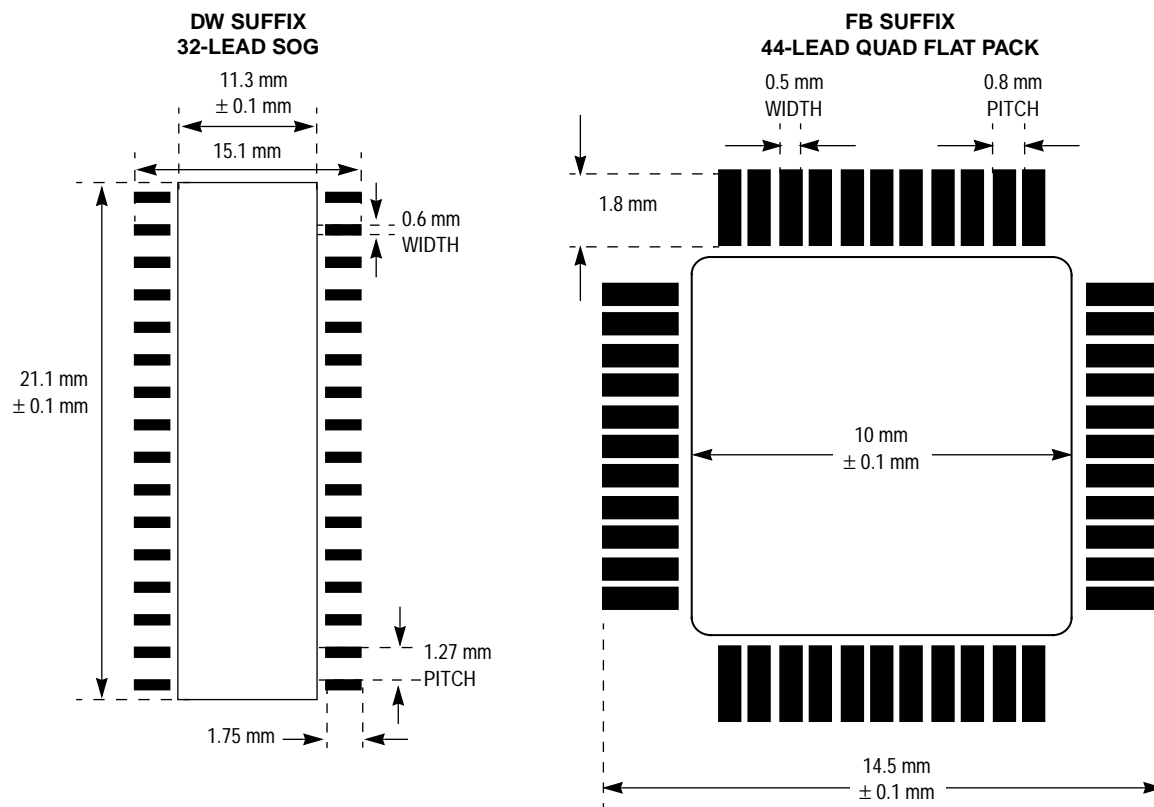
NOTES:

1. DIMENSIONING AND TOLERANCING PER ANSI Y14.5M, 1982.
2. CONTROLLING DIMENSION: MILLIMETER.
3. DATUM PLANE -H- IS LOCATED AT BOTTOM OF LEAD AND IS COINCIDENT WITH THE LEAD WHERE THE LEAD EXITS THE PLASTIC BODY AT THE BOTTOM OF THE PARTING LINE.
4. DATUMS -L-, -M- AND -N- TO BE DETERMINED AT DATUM PLANE -H-.
5. DIMENSIONS S AND V TO BE DETERMINED AT SEATING PLANE -T-.
6. DIMENSIONS A AND B DO NOT INCLUDE MOLD PROTRUSION. ALLOWABLE PROTRUSION IS 0.25 (0.010) PER SIDE. DIMENSIONS A AND B DO INCLUDE MOLD MISMATCH AND ARE DETERMINED AT DATUM PLANE -H-.
7. DIMENSION D DOES NOT INCLUDE DAMBAR PROTRUSION. DAMBAR PROTRUSION SHALL NOT CAUSE THE D DIMENSION TO EXCEED 0.530 (0.021).

DIM	MILLIMETERS		INCHES	
	MIN	MAX	MIN	MAX
A	9.90	10.10	0.390	0.398
B	9.90	10.10	0.390	0.398
C	2.00	2.21	0.079	0.087
D	0.30	0.45	0.0118	0.0177
E	2.00	2.10	0.079	0.083
F	0.30	0.40	0.012	0.016
G	0.80 BSC		0.031 BSC	
J	0.13	0.23	0.005	0.009
K	0.65	0.95	0.026	0.037
M	5°	10°	5°	10°
S	12.95	13.45	0.510	0.530
V	12.95	13.45	0.510	0.530
W	0.000	0.210	0.000	0.008
Y	5°	10°	5°	10°
A1	0.450 REF		0.018 REF	
B1	0.130	0.170	0.005	0.007
C1	1.600 REF		0.063 REF	
R1	0.130	0.300	0.005	0.012
R2	0.130	0.300	0.005	0.012
θ_1	5°	10°	5°	10°
θ_2	0°	7°	0°	7°

6.3.7 MC143120 Pad Layout

MC143120 PAD LAYOUTS



6.3.8 Sockets for Neuron Chips

NOTE: Motorola can not recommend one supplier over another and in no way suggests that these are the only suppliers.

Integrated Circuit	Manufacturer	Part Number
MC143120 32-Pin SOG (DW Suffix)	Yamaichi	IC51-0322-667-2
MC143120 44-Pin PQFP (FB Suffix)	Enplas	FPQ-44-0.8-16
MC143150B1FU1/MC143150B2 64-Pin QFP	Enplas	FPQ-64-0.8-02

6.3.9 Test Clips

Below is a listing of Pamona IC test clips and their part numbers.

Device	Type	Pamona Part Number
MC143120 32-Pin SOG (DW Suffix)	SOIC	6107
MC143120 44-Pin PQFP (FB Suffix)	PLCC	5281
MC143150B1FU1/MC143150B2 64-Pin QFP	QFP	5888-2

Programming Model **7**

SECTION 7

LONWORKS PROGRAMMING MODEL

The primary programming language used to write applications for the Neuron Chip is a derivative of the C programming language called Neuron C. Neuron C is based on ANSI C, enhanced to support I/O, event processing, message passing, and distributed data objects. Several major differences between Neuron C and ANSI C are in the area of supported data types.

The numeric data types supported by Neuron C are:

char	8 bits	signed or unsigned
short	8 bits	signed or unsigned
int	8 bits	signed or unsigned
long	16 bits	signed or unsigned
boolean	8 bits	

Neuron C also supports *typedefs*, *enums*, arrays, pointers, *structs*, and *unions*. Unlike ANSI C, Neuron C does not include a standard runtime library supporting file I/O and other features common to larger target processors, such as floating point. However, Neuron C has a special runtime library and language syntax extensions supporting intelligent distributed control applications using Neuron Chips. Neuron C extensions include software timers, network variables, explicit messages, a multitasking scheduler, EEPROM variables, and miscellaneous functions. For further details on the Neuron C language, see the *Neuron C Programmer's Guide*.

7.1 TIMERS

The application program can use up to 15 timers that decrement either every second or every millisecond, and optionally repeat. These timers are implemented in software running on the Network Processor, and are independent of the Neuron Chip input clock rate. The expiration of a timer is an event that may cause the execution of a user-written task. This event is called *timer_expires*. The value of a timer variable is an unsigned long (0 – 65,535). Timers can be set to any value at any time by the application program.

7.2 NETWORK VARIABLES

The application program can declare a special class of static objects called network variables, which may be of class *input* or *output*. Assignment of a value to a network output variable causes propagation of that value to all nodes declaring an input variable that is connected to the network output variable. For example, a node that contains a temperature sensing device could declare an output network variable which contains the current temperature sensed by the node. Every time the node measures a new value for the temperature, it updates the output network variable. Another node (or nodes) needing to know the current temperature, such as a heating control node, can then declare an input network variable for current temperature. Whenever the heating control node wants to use the value of the current temperature, it simply refers to the input network variable which will always contain the last temperature measured by the sensing node. At installation time, the output network variable on the sensing node is connected to the input network variable on the controlling node.

Binding is the process of connecting network variables from different nodes together, and is typically performed by a network management device. The LonBuilder Developer's Workbench and the LonManager Applications Programming Interface (API) include such a binding capability. The binding is physically implemented by sending network management messages containing the necessary addressing information to the nodes to be connected. Nodes may also update their own binding information for simple networks, without network management devices. Tables containing binding information are in the Neuron Chip's EEPROM, which may be updated.

Nodes declaring an input network variable need only refer to that variable to determine the latest value propagated across the network. A node declaring an input network variable may also call the library routine *poll* to cause the latest value to be propagated. The node being polled must declare the output network variable as a polled variable. The library routine *is_bound* may be used to check if a network variable is associated with (bound to) a network variable on another node.

Note that declaring network variables within a node's code occurs at compile time. The binding of a node's network variable output to the input of another node occurs at a later time, either before, during, or after the node is installed in a network.

The network variable concept greatly simplifies the programming of complex distributed applications. Network variables provide a very flexible view of distributed data to be operated on by the nodes in a system. The programmer does not need to deal with message buffers, node addressing, request/response/retry processing, and other low-level details.

A node running a Neuron C application program may declare up to 62 network variables, including array elements. In most cases, this is not a significant limitation, since a single input network variable can receive data of the same type from an unlimited number of nodes, and a single output network variable can send data of the same type to an unlimited number of nodes. Additional network variables can be accessed from Neuron C by explicitly building a network variable update or fetch as described in Section B.3. A network variable may be a Neuron C variable or structure up to 31 bytes in length. Arrays of up to 31 bytes may be embedded in a structure and propagated as a single network variable. If more than 31 bytes of data are needed in a single message, explicit messaging can be used as described in Section 7.3. A network variable may also be an array of elements, each of which may be individually connected to network variables on other nodes. An output network variable on a node may also be connected to an input network variable on the same node (turnaround network variable). Nodes may be implemented with all applications and communications processing running on a Neuron Chip. Such nodes are called *Neuron Chip-hosted nodes* and are programmed using the Neuron C programming language.

Nodes may also be implemented using the Neuron Chip as a communications processor and a second processor as the host for the applications processing. Such nodes are called *host-based nodes* and are programmed using the native programming tools of the host processor. The host may be a microcontroller, microprocessor, PC, workstation, or any other computer. The host communicates with a LONWORKS network via a *LONWORKS network interface*.

A LONWORKS network interface implements layers 1 through 6 of the LonTalk protocol, and moves layer 6 and layer 7 application support to the host processor. The LonTalk protocol includes the specification of a standard protocol between the host and network interface, and is called the LONWORKS network interface protocol.

Turn-key LONWORKS network interfaces are available, such as Echelon's PCLTA (PC LonTalk Adapter) and SLTA/2 (Serial LonTalk Adapter), to provide a ready-made network interface for any host. A custom LONWORKS network interface can also be built using the LonBuilder Microprocessor Interface Program (MIP). The MIP is firmware for the Neuron Chip that transforms the Neuron Chip into a high-performance communications processor for an attached host.

When using a LONWORKS network interface, network variables are moved to the host processor. Network variable configuration management may be performed entirely by the host (called *host selection*), or may be split between the host and network interface (called *network interface selection*). When host selection is used, the host application can implement up to 4096 bound network variables versus the 62 bound network variables for Neuron Chip-hosted nodes. For further details, see the *Host Application Programmer's Guide* and the *MIP User's Guide*.

The network variable updates may be sent with four classes of service:

<i>ACKD</i>	Acknowledged service with retries
<i>UNACKD</i>	Unacknowledged service
<i>UNACKD_RPT</i>	Unacknowledged repeated service (message sent multiple times)
<i>REQUEST</i>	Request/response service, used for polling network variables

Network variables may be specified as authenticated, meaning that authenticated messages are used to transmit their values (see Section 8.6). Network variables may also be specified to have priority, meaning that a priority timeslot is used to transmit their values (see Section 8.7). Finally, network variables may be specified as synchronous, in which case all values assigned to the network variable are propagated. Normally network variables propagate only the most recent value when network variable updates are generated faster than they can be propagated. Intermediate values may be discarded.

The following built-in events may be checked by the scheduler to allow the asynchronous processing of network variables:

<i>nv_update_occurs</i>	A new value has been received for an input network variable
<i>nv_update_fails</i>	Propagation of the value of an output network variable has failed
<i>nv_update_succeeds</i>	Propagation of the value of an output network variable has succeeded
<i>nv_update_completes</i>	Propagation of the value of an output network variable has completed, either successfully or unsuccessfully

Completion status is binary. A network variable update either succeeded or it did not. With acknowledged service, a network variable update succeeds only if acknowledgments are received from all the recipients. With unacknowledged or repeated service, a network variable update succeeds only if the update message was transmitted onto the network. Additional information about the source of failures can be derived by examining node statistics using the query status network management message.

If the destination node application wishes to know the source address of an incoming network variable update message, it can refer to the built-in variable *nv_in_addr*. The definition of this variable is in the *ECHELON.H* include file and is reproduced here for reference.

```
const struct {
    unsigned domain      : 1; // domain table reference
    unsigned flex_domain: 1; // received on flexible domain
    unsigned format      : 6; // 0 = broadcast, 1 = group,
                               // 2 = subnet/node, 3 = Neuron ID
                               // 4 = turnaround
    struct {
        unsigned subnet;
        unsigned      : 1;
        unsigned node : 7;
    } src_addr; // source address
    struct {
        unsigned group // group destination address
    } dest_addr;
} nv_in_addr;
```

7.3 NETWORK VARIABLE ALIASES

To reduce the constraints on connections between network variables, a special class of network variables is defined called *alias network variables*, or *aliases*. An alias is an abstraction for a network variable that is managed by the Neuron Chip firmware. Each alias can be associated with a non-alias network variable, which is called the *primary network variable*. The alias inherits the attributes of the primary network variable including data value, direction, and type. The primary network variable's value is shared with its aliases, so network variable update messages received by a primary input network variable or any of its aliases update the value of the primary network variable and generate an update event for the primary network variable. Poll requests to a primary output network variable or its aliases generate a response based on the primary network variable's value. When an application updates a primary output network variable, the Neuron Chip firmware sends a network variable update message for the primary network variable and each associated alias on the sending node. When an application initiates a poll for a primary input network variable, the Neuron Chip firmware sends a network variable poll request for the primary network variable and each associated alias on the polling node.

7.4 EXPLICIT MESSAGES

For most applications, network variables allow the most compact and simple implementation. However, for applications where data objects larger than 31 bytes need to be transmitted, request/response service is desired, or the network variable model is not suitable, then the application can use explicit messaging.

The application program can construct messages containing up to 229 bytes of data. These messages can be addressed to other nodes or groups of nodes via implicit address connections called message tags. Alternatively, messages may be explicitly addressed to other nodes using subnet/node, group, broadcast, or unique ID addressing. The messages may be sent with four classes of service:

<i>ACKD</i>	Acknowledged service with retry
<i>UNACKD</i>	Unacknowledged service
<i>UNACKD_RPT</i>	Unacknowledged repeated service (message sent multiple times)
<i>REQUEST</i>	Request/response service

Request/response service allows the node receiving a message to respond with data to the response message, as distinct from an acknowledgment, which contains no application-level data. Messages may be specified as authenticated (see Section 5.6). Messages may also be specified to have priority, meaning that they use a priority timeslot on the channel if the node has a priority slot allocated to it. Priority messages are always sent by the node before non-priority messages (see Section 5.7). The application program may explicitly assign the destination addresses, or use the default address associated with the message tag.

Messages are exchanged between nodes by calling runtime library support routines:

<i>msg_alloc()</i>	Allocate a message buffer
<i>msg_alloc_priority()</i>	Allocate a priority message buffer
<i>msg_cancel()</i>	Cancel a message being built for sending
<i>msg_free()</i>	Free a message buffer
<i>msg_receive()</i>	Receive a message at this node
<i>msg_send()</i>	Send a message to another node
<i>resp_alloc()</i>	Allocate a buffer for a response to a request message
<i>resp_cancel()</i>	Cancel a response being built
<i>resp_free()</i>	Free a response buffer
<i>resp_receive()</i>	Receive a response to a request message
<i>resp_send()</i>	Send a response to a request message

The following built-in events may be checked by the scheduler to allow asynchronous transmission and reception of messages:

<i>msg_arrives()</i>	A message has arrived at this node
<i>msg_completes()</i>	Outgoing message has been handled, either successfully or unsuccessfully
<i>msg_succeeds()</i>	Outgoing message has been successfully sent
<i>msg_fails()</i>	Some acknowledgments have not been received for an outgoing message
<i>resp_arrives()</i>	A response to a request has been received

The following data objects are defined for use by the application program:

<i>msg_in()</i>	Currently received message
<i>msg_out()</i>	Message to be sent
<i>resp_in()</i>	Currently received response to a previous message
<i>resp_out()</i>	Response to be sent to a previous request

The declarations of these structures are built into the Neuron C compiler. They are reproduced here for reference.

```
typedef enum {
    ACKD,                // acknowledged
    UNACKD_RPT,          // unacknowledged repeated
    UNACKD,              // unacknowledged
    REQUEST              // request/response
} service_type;
struct {
    int      code;        // message code
    int      len;         // length of message data
    int      data [ ];    // message data
    boolean  authenticated; // TRUE if message was successfully
                          // authenticated
    service_type service; // service type
    msg_in_addr addr;     // optional field if explicit
                          // addressing used
} msg_in;
struct {
    boolean  priority_on; // TRUE if message is to be sent with
                          // priority
    msg_tag; tag;         // destination message tag
    int      code;        // message code
    int      data [ ];    // message data
    boolean  authenticated; // TRUE if message is to be
                          // authenticated
    service_type service; // service type
    msg_out_addr dest_addr; // optional field if explicit
                          // addressing used
} msg_out;
struct {
    int      code;        // response message code
    int      len;         // length of response message data
    int      data [ ];    // message data
    resp_in_addr addr;     // optional field if explicit
                          // addressing used
}
```

```

} resp_in;

struct {
    int      code;           // response message code
    int      data [ ];      // response message data
} resp_out;

```

If an explicit address is used to address an outgoing message, the source node application creates an addr data structure in the msg_out buffer. If the destination node wishes to know the source address of an incoming message, it can refer to the addr data structure in the msg_in buffer. If a requesting node wishes to know the source address of an incoming response, it can refer to the addr data structure in the resp_in buffer. These definitions are in the include file \LB\INCLUDE\MSG_ADDR.H, and are reproduced here for reference. For definitions of the data types: addr_type, group_struct, snode_struct, nrnid_struct, and bcast_struct, see Section A.3.

```

typedef union {
    addr_type      no_address;    // unbound
    group_struct   group;        // group
    snode_struct   snode;        // subnet/node
    nrnid_struct   nrnid;        // Neuron ID
    bcast_struct   bcast;        // broadcast
} msg_out_addr;

typedef struct {
    unsigned domain      : 1; // domain table reference
    unsigned flex_domain: 1; // received on flexible domain
    unsigned format      : 6; // 0 = broadcast, 1 = group,
                                // 2 = subnet/node; 3 = Neuron ID

    struct {
        unsigned subnet;
        unsigned      : 1;
        unsigned node  : 7;
    } src_addr;           //source address
    union {
        unsigned bcast_subnet; // broadcast destination address
        unsigned group;        // group destination address
        struct {
            unsigned subnet;
            unsigned      : 1;
            unsigned node  : 7;
        } snode;           // subnet/node destination address
        struct {
            unsigned subnet;
            unsigned nid[ Neuron_ID_LEN ];
        } nrnid;           // Neuron ID destination address
    } dest_addr;          // destination address
} msg_in_addr;

typedef struct {
    unsigned domain      : 1; // domain table reference
    unsigned flex_domain: 1; // received on flexible domain
    struct {
        unsigned subnet;
        unsigned is_snode : 1; // 0 = group response,
                                // 1 = subnet/node response
    }

```

```

        unsigned node      : 7;
    } src_addr;           // source address
union {
    struct {
        unsigned subnet;
        unsigned      :1;
        unsigned node  :7;
    } snode;             // Subnet/node destination address
    struct {
        unsigned subnet;
        unsigned      :1;
        unsigned node  :7;
        unsigned group;
        unsigned      :2;
        unsigned member:6;
    } group;             // group destination address
    } dest_addr;         // destination address
} resp_in_addr;

```

Preemption Mode

The use of buffer allocation functions (`msg_alloc` and `msg_alloc_priority`) is optional. If they are not used and buffers are not available when message construction commences, the node enters the preemption mode. The preemption mode suspends the application program except for completion event (e.g., msg completes) and incoming message event processing. If no buffer becomes available within a configurable number of seconds known as the maximum free buffer wait time, the node resets. This is known as the preemption mode timeout. Preemption mode can also be initiated when synchronous network variables are updated or when the *flush_wait* function is used.

The use of the `preempt_safe` keyword before a *when* clause causes the evaluation of the *when* clause, even if the node is in preemption mode. See the *Neuron C Programmer's Guide* for more information.

7.5 SCHEDULER

The scheduler executes user-written tasks in response to events or conditions specified in *when* clauses by the application program. When a specified event or condition becomes true, the associated task code is executed. The user has the capability of specifying one or more *when* clauses as having priority, and the scheduler checks priority *when* clauses in order of their appearance in the Neuron C program, followed by one non-priority *when* clause. Each of the remaining non-priority *when* clauses is checked during successive scheduler cycles, at one clause per cycle. The task scheduler can handle up to 80 *when* clauses, depending on type.

Events fall into five classes:

System Wide Events:

<i>reset</i>	Node has been reset
<i>offline</i>	Node has been set offline
<i>online</i>	Node has been set online
<i>flush_completes</i>	Node has completed preparations to enter sleep mode
<i>wink</i>	Node has received "wink" network management message with "wink" command option

Input/Output Events:

<i>io_changes</i>	Value read from an I/O device has changed since last reading. Changes may be unconditional, or specified as <i>to</i> a specified value, or <i>by</i> a specified amount.
<i>io_update_occurs</i>	Value read from a timer/counter input object device has been updated
<i>io_in_ready</i>	Parallel I/O device is ready to receive data from external processor
<i>io_out_ready</i>	Parallel I/O device is ready to transmit data to external processor

Timer Events:

<i>timer_expires</i>	Software timer value has decremented to 0
----------------------	---

Message and Network Variable Events:

The following events are associated with the transmission of network variables and messages (also discussed in Sections 7.2 and 7.3):

nv_update_occurs
nv_update_fails
nv_update_succeeds
nv_update_completes
msg_arrives
msg_completes
msg_succeeds
msg_fails
resp_arrives

User-Specified Events:

<*boolean expression*> User-specified expression, evaluating to true or false

7.6 ADDITIONAL LIBRARY FUNCTIONS

The following miscellaneous functions are available from the application program. These functions are built into the Neuron C Compiler, are part of the Neuron Chip system image, or are linked into the application image from a system library. Each function has a specified type from the following list:

1. Built-in function included with the Neuron C Compiler.
2. Function included with Neuron 3120, 3120E2, and 3150 Chip system images.
3. Function included with Neuron 3150 Chip system image and linked into the application image from a system library for Neuron 3120 and 3120E2 Chips.
4. Function linked into the application image from a system library for all Neuron Chips.
5. Function included with Neuron 3150 Chip system image and linked into the application image from a system library for Neuron 3120E2 Chips. Not available on Neuron 3120 Chips.
6. Function included with Neuron 3120 and 3150 Chip system images and linked into the application image from a system library for Neuron 3120E2 Chips.

Function	Type	Description
Execution Control:		
<i>delay()</i>	2	Delay processing for a time independent of input clock rate
<i>flush_cancel()</i>	2	Cancel a flush in process
<i>flush_wait()</i>	3	Wait for outgoing messages and updates to be sent before going offline
<i>go_offline()</i>	2	Cease execution of the application program
<i>post_events()</i>	2	Defines a critical section for network variable and message processing
<i>power_up()</i>	3	Determine whether last processor reset was due to power up
<i>scaled_delay</i>	2	Delay processing for a time that depends on the input clock rate
<i>watchdog_update</i>	2	Tickle the watchdog timer to prevent node reset
Network Management Control:		
<i>access_address</i>	3	Read node's address table
<i>access_alias</i>	5	Read node's alias table
<i>access_domain</i>	3	Read node's domain table
<i>access_nv</i>	3	Read node's network variable configuration table
<i>addr_table_index</i>	1	Determine address table index of message tag
<i>go_unconfigured</i>	3	Reset this node to an uninstalled state
<i>nv_table_index</i>	1	Determine index of a network variable
<i>offline_confirm</i>	2	Inform network management tool that this node is going offline
<i>update_address</i>	3	Write node's address table
<i>update_alias</i>	5	Write node's alias table
<i>update_clone_domain</i>	3	Write node's domain table with clone entry
<i>update_domain</i>	3	Write node's domain table with normal entry
<i>update_nv</i>	3	Write node's network variable configuration table
<i>update_config_data</i>	3	Write node's configuration data structure
Error Handling:		
<i>application_restart</i>	2	Begin application program over again
<i>clear_status</i>	4	Clear error statistics accumulators and error log
<i>error_log</i>	4	Record software-detected error
<i>node_reset</i>	2	Activate Neuron Chip $\overline{\text{RESET}}$ pin, and reset all processors
<i>retrieve_status</i>	3	Read error statistics from protocol processor
Sleep Mode:		
<i>flush</i>	2	Flush all outgoing messages and network variable updates
<i>sleep</i>	2	Enter low-power mode by disabling system clock
<i>timers_off</i>	2	Turn off all software timers

Function	Type	Description
Integer Math:		
<i>abs()</i>	2	Arithmetic absolute value
<i>bcd2bin()</i>	3	Convert binary coded decimal data to binary
<i>bin2bcd()</i>	3	Convert binary data to binary coded decimal
<i>high_byte()</i>	1	Extract the high byte of a 16-bit number
<i>low_byte()</i>	1	Extract the low byte of a 16-bit number
<i>make_long()</i>	1	Create a 16-bit number from two 8-bit numbers
<i>max()</i>	2	Arithmetic maximum of two values
<i>min()</i>	2	Arithmetic minimum of two values
<i>muldiv()</i>	3	Multiply/divide with 32-bit intermediate result — unsigned
<i>muldivs()</i>	3	Multiply/divide with 32-bit intermediate result — signed
<i>random()</i>	2	Generate 8-bit random number
<i>reverse()</i>	3	Reverse the order of bits in an 8-bit number
<i>rotate_long_left()</i>	4	Rotate left a 16-bit number
<i>rotate_long_right()</i>	4	Rotate right a 16-bit number
<i>rotate_short_left()</i>	4	Rotate left an 8-bit number
<i>rotate_short_right()</i>	4	Rotate right an 8-bit number
<i>s32_abs()</i>	4	Take the absolute value of a signed 32-bit number
<i>s32_add()</i>	4	Add two signed 32-bit numbers
<i>s32_cmp()</i>	4	Compare two 32-bit signed numbers
<i>s32_copy()</i>	4	Copy a 32-bit signed number
<i>s32_dec()</i>	4	Decrement a 32-bit signed number
<i>s32_div()</i>	4	Divide two signed 32-bit numbers
<i>s32_div2()</i>	4	Divide a 32-bit signed number by 2
<i>s32_eq()</i>	4	Return TRUE if two signed 32-bit numbers are equal
<i>s32_from_ascii()</i>	4	Convert an ASCII string into 32-bit signed
<i>s32_from_slong()</i>	4	Convert a signed long number into 32-bit signed
<i>s32_from_ulong()</i>	4	Convert an unsigned long number into 32-bit signed
<i>s32_ge()</i>	4	Return TRUE if first argument is \geq second argument
<i>s32_gt()</i>	4	Return TRUE if first argument is $>$ the second argument
<i>s32_inc()</i>	4	Increment a 32-bit signed number
<i>s32_le()</i>	4	Return TRUE if first argument is \leq second argument
<i>s32_lt()</i>	4	Return TRUE if first argument is $<$ the second argument
<i>s32_max()</i>	4	Take the maximum of two signed 32-bit numbers
<i>s32_min()</i>	4	Take the minimum of two signed 32-bit numbers
<i>s32_mul()</i>	4	Multiply two signed 32-bit numbers
<i>s32_mul2()</i>	4	Multiply a 32-bit signed number by 2
<i>s32_ne()</i>	4	Return TRUE if two signed 32-bit numbers are not equal
<i>s32_neg()</i>	4	Take the negative of a signed 32-bit number
<i>s32_rand()</i>	4	Return a random 32-bit signed number
<i>s32_rem()</i>	4	Take the remainder of a division of two signed 32-bit numbers
<i>s32_sign()</i>	4	Return the sign of a 32-bit signed number
<i>s32_sub()</i>	4	Subtract two signed 32-bit numbers
<i>s32_to_ascii()</i>	4	Convert a 32-bit signed number into an ASCII string
<i>s32_to_slong()</i>	4	Convert a 32-bit signed number into signed long
<i>s32_to_ulong()</i>	4	Convert a 32-bit signed number into unsigned long
<i>swap_bytes()</i>	1	Exchange the two bytes of a 16-bit number

Function	Type	Description
Floating Point Math:		
<i>fl_abs()</i>	4	Take the absolute value of a floating point number
<i>fl_add()</i>	4	Add two floating point numbers
<i>fl_ceil()</i>	4	Return the ceiling of a floating point number
<i>fl_cmp()</i>	4	Compare two floating point numbers
<i>fl_div()</i>	4	Divide two floating point numbers
<i>fl_div2()</i>	4	Divide a floating point number by 2
<i>fl_eq()</i>	4	Return TRUE if two floating point numbers are equal
<i>fl_floor()</i>	4	Return the floor of a floating point number
<i>fl_from_ascii()</i>	4	Convert an ASCII string to floating point
<i>fl_from_s32()</i>	4	Convert a signed 32-bit number to floating point
<i>fl_from_slong()</i>	4	Convert a signed long number into a floating point number
<i>fl_from_ulong()</i>	4	Convert an unsigned long number to floating point
<i>fl_ge()</i>	4	Return TRUE if first argument \geq second argument
<i>fl_gt()</i>	4	Return TRUE if first argument $>$ second argument
<i>fl_le()</i>	4	Return TRUE if first argument \leq second argument
<i>fl_lt()</i>	4	Return TRUE if first argument $<$ second argument
<i>fl_max()</i>	4	Find the maximum of two floating point numbers
<i>fl_min()</i>	4	Find the minimum of two floating point numbers
<i>fl_mul()</i>	4	Multiply two floating point numbers
<i>fl_mul2()</i>	4	Multiply a floating point number by 2
<i>fl_ne()</i>	4	Return TRUE if two floating point numbers are not equal
<i>fl_neg()</i>	4	Take the negative of a floating point number
<i>fl_rand()</i>	4	Return a random floating point number
<i>fl_round()</i>	4	Round a floating point number to the nearest whole number
<i>fl_sign()</i>	4	Return the sign of a floating point number
<i>fl_sqrt()</i>	4	Take the square root of a floating point number
<i>fl_sub()</i>	4	Subtract two floating point numbers
<i>fl_to_ascii()</i>	4	Convert a floating point number to an ASCII string
<i>fl_to_ascii_fmt()</i>	4	Convert a floating point number to a formatted ASCII string
<i>fl_to_s32()</i>	4	Convert a floating point number to signed 32-bit
<i>fl_to_slong()</i>	4	Convert a floating point number to signed long
<i>fl_to_ulong()</i>	4	Convert a floating point number to unsigned long
<i>fl_trunc()</i>	4	Take the whole number part of a floating point number

Function	Type	Description
Strings:		
<i>strcat()</i>	4	Append a copy of a string at the end of another
<i>strchr()</i>	4	Scan a string for a specific character
<i>strcmp()</i>	4	Compare two strings
<i>strcpy()</i>	4	Copy one string into another
<i>strlen()</i>	4	Return the length of a string
<i>strncat()</i>	4	Append a copy of a string at the end of another
<i>strncmp()</i>	4	Compare two strings
<i>strncpy()</i>	4	Copy one string into another
<i>strrchr()</i>	4	Scan a string for a specific character
Utilities:		
<i>abs()</i>	2	Arithmetic absolute value
<i>ansi_memcpy()</i>	4	Copy a block of memory with ANSI return value
<i>ansi_memset()</i>	4	Set a block of memory to a specified value with ANSI return value
<i>bcd2bin()</i>	3	Convert binary coded decimal data to binary
<i>bin2bcd()</i>	3	Convert binary data to binary coded decimal
<i>clr_bit()</i>	4	Clear a bit in a bit array
<i>max()</i>	2	Arithmetic maximum of two values
<i>memcpy()</i>	4	Copy a block of memory
<i>memchr()</i>	4	Search a block of memory
<i>memcmp()</i>	4	Compare a block of memory
<i>memcpy()</i>		Copy a block of memory:
	3	• From <i>msg_in.data</i> and <i>resp_in.data</i>
	3	• To <i>resp_out.data</i>
	3	• Length ≥ 256 bytes
	2	• Others
<i>memset()</i>		Set a block of memory to a specified value:
	3	• Length ≥ 256 bytes
	2	• Others
<i>min()</i>	2	Arithmetic minimum of two values
<i>muldiv(s)</i>	3	Multiply/divide with 32-bit intermediate result — (un)signed
<i>random()</i>	2	Generate 8-bit random number
<i>refresh_memory()</i>	2	Rewrite contents of EEPROM memory
<i>retrieve_status()</i>	3	Read statistics from protocol processor
<i>retrieve_xcvr_status()</i>	4	Read transceiver status register
<i>reverse()</i>		Reverse the order of bits in an 8-bit number
<i>set_bit()</i>	4	Set a bit in a bit array
<i>set_eeprom_lock()</i>	2	Set the state of the checksummed EEPROM lock
<i>tst_bit()</i>	4	Return TRUE if bit tested was set

Function	Type	Description
Input/Output (see Section 8 for details):		
<i>io_change_init()</i>	2	Initialize reference value for <i>io_changes</i> event
<i>io_edgelog_preload()</i>	3	Define maximum value for edgelog period measurements
<i>io_in()</i>		Input data from I/O object:
	3	• Dualslope input
	3	• Edgelog input
	3	• Infrared input
	4	• JIScard input
	3	• Magcard input
	3	• Neurowire I/O slave mode
	4	• Neurowire I/O with invert option
	6	• Serial input
	4	• Touch I/O
	4	• Wiegand input
	2	• Others
<i>io_in_ready()</i>	2	Event function which evaluates to TRUE when a block of data is available from the parallel I/O object
<i>io_in_request()</i>	3	Start dualslope A/D conversion
<i>io_out()</i>		Output data to I/O object:
	6	• Bitshift output
	3	• Neurowire I/O slave mode
	4	• Neurowire I/O with invert option
	6	• Serial output
	4	• Touch I/O
	2	• Others
<i>io_out_ready()</i>	2	Event function which evaluates to TRUE when a block of data is available from the parallel I/O object
<i>io_out_request()</i>	2	Request ready indication from parallel I/O object
<i>io_preserve_input()</i>	3	Preserve first timer/counter value after reset or <i>io_select()</i>
<i>io_select()</i>	2	Set timer/counter multiplexer
<i>io_set_clock()</i>	2	Set timer/counter clock rate
<i>io_set_direction()</i>	2	Change direction of I/O pins

7.7 BUILT-IN VARIABLES

<i>activate_service_led</i>	Control service LED state
<i>config_data</i>	Read-only copy of the node's configuration data
<i>input_value</i>	Data read by last explicit <i>io_in()</i> call or by last input/output event (e.g., <i>io_changes</i>)
<i>input_is_new</i>	TRUE if last input from a timer/counter object read an updated value
<i>msg_tag_index</i>	Message tag index for the most recent explicit message response or completion event received
<i>nv_array_index</i>	Index of element in network variable array with updated value
<i>nv_in_addr</i>	Source address of the last network variable update
<i>nv_in_index</i>	Network variable index for the most recent network variable update

<i>read_only_data</i>	Copy of node's read only data structure
<i>read_only_data_2</i>	Copy of node's read only data structure extension

7.8 MC143120 AND MC143120E2 FIRMWARE EXTENSIONS

On the MC143120, all application code is placed in on-chip EEPROM. System library functions specified in Section 7.5 are linked with the application and are also placed in on-chip EEPROM. In addition, when any of the following Neuron C features are used, object code is brought in from a system library and placed in on-chip EEPROM. The sizes and number of functions are firmware dependent.

Function	No.Bytes	Description
<code>access_address()</code>	5	Returns a pointer to the address table
<code>access_domain()</code>	5	Returns a pointer to the domain table
<code>access_nv()</code>	17	Returns a pointer to the network variable configuration table
<code>addr_table_index()</code>		Determines the address table index of a message tag (0 – 14)
<code>bcd2bin()</code>	24	Converts a binary coded decimal (BCD) structure to a binary number
<code>bin2bcd()</code>	47	Converts a binary number to a BCD structure
<code>flush_wait()</code>	6	Causes an application to enter preemption mode: pending events processed
<code>go_unconfigured()</code>	14	Puts node into an unconfigured state, overwriting all domain information, and destroys authentication
<code>muldiv()</code>	37	(A*B)/C: Makes a 32-bit intermediate product out of (A*B) then divides it by C; A, B, and C are all 16-bit unsigned long numbers
<code>muldivs()</code>	25	Same as muldiv except A, B, and C are signed long numbers
<code>nv_table_index()</code>		Determines the index of a network variable (0 – 61)
<code>power_up()</code>	5	Returns TRUE if the last reset resulted from a power_up; this can be used to tell if it was first reset from power up
<code>retrieve_status()</code>	11	Returns diagnostics status information
<code>reverse()</code>	11	Reverses bits (unsigned int), ex: 0xE3 becomes 0xc7
<code>update_address()</code>	6	Self-configuration function to modify own configuration data
<code>update_clone_domain()</code>	15	Self-configuration function to modify own configuration data
<code>update_config_data()</code>		
<code>update_domain()</code>		
<code>update_nv()</code>	6	Self-configuration function to modify own configuration data

The following are not functions but are automatically called by the compiler and stored in EEPROM if the Neuron C application needs them.

<code>_bitf_sign_ext</code>	7	Use of a signed bitfield
<code>_memcpy16</code>	19	Memcpy or structure assignment with length ≥ 256
<code>_memset16</code>	18	Memset with length ≥ 256
<code>_msg_data_blockget</code>	17	Memcpy out of msg_in.data
<code>_msg_in_addr_ptr</code>	6	Any use of msg_in.addr

<code>_msg_out_addr_ptr</code>	15	Any use of <code>msg_out.dest_addr</code>
<code>_Neurowire_slave</code>	59	Neurowire slave only; any <code>io_in()</code> or <code>io_out()</code> for a Neurowire slave object
<code>_resp_data_blockset</code>	7	Memcpy into <code>resp_out.data</code>

The ability to send explicitly addressed messages is also stored in EEPROM if the Neuron C application needs them.

I/O Objects

- Dualslope input
- Edgelog input
- Infrared input
- Magcard input
- Neurowire I/O slave mode

Miscellaneous

- Explicitly addressed messages
- Structure assignment (length \geq 256 bytes)
- Use of a signed bit field

LonTalk Protocol 8

SECTION 8

LonTalk PROTOCOL

The Neuron Chip implements a complete networking protocol using two of the three on-chip processors (Processor-1 and Processor-2). This networking protocol follows the ISO OSI reference model for network protocols; it allows application code running on Processor-3 to communicate with applications running on other Neuron Chip nodes elsewhere on the same network. See Sections 7.2 and 7.3 and the *Neuron C Programmer's Guide* for details of how the protocol is used by application-level objects called network variables and message tags. Table 8-1 shows the mapping of LonTalk services onto the 7-layer OSI reference model.

Table 8-1. LonTalk Protocol Layering

OSI Layer		Purpose	Services Provided	Processor
7	Application	Application compatibility	Standard network variable types	Application
6	Presentation	Data interpretation	Network variables, foreign frame transmission	Network
5	Session	Remote actions	Request/response, authentication, network management	Network
4	Transport	End-to-end reliability	Acknowledged and unacknowledged, unicast and multicast, authentication, common ordering, duplicate detection	Network
3	Network	Destination addressing	Addressing, routers	Network
2	Link	Media access and framing	Framing, data encoding, CRC error checking, predictive carrier sense multiple access (CSMA), collision avoidance, priority, collision detection	MAC
1	Physical	Electrical interconnect	Media-specific interfaces and modulation schemes	MAC, XCVR

The main features of the LonTalk protocol are described in the following sections.

8.1 MULTIPLE MEDIA SUPPORT

The protocol processing on the Neuron Chip is media-independent. This allows the Neuron Chip to support a wide variety of communications media, including twisted-pair, powerline, RF, infrared, coaxial cable, and fiber optics.

8.2 SUPPORT FOR MULTIPLE COMMUNICATION CHANNELS

A channel is a physical transport medium for packets and can contain up to 32,385 nodes, maximum. A network may be composed of one or more channels. In order for packets to be transferred from one channel to another, a device called a router is used to connect the two channels. Routers typically consist of two Neuron Chips connected together via their I/O pins. Each of the Neuron Chips uses its own transceiver to communicate with its channel. The protocol supports such devices so that multiple media networks may be constructed, and network loading can be optimized by localizing traffic.

8.3 COMMUNICATIONS RATES

Channels may be configured for different bit rates to allow trade-offs of distance, throughput, and power consumption. The allowable bit rates are 0.6, 1.2, 2.4, 4.9, 9.8, 19.5, 39.1, 78.1, 156.3, 312.5, 625, and 1250 kbps. Channel throughput depends on bit rate, oscillator frequencies and accuracy, transceiver characteristics, average size of a packet, and the use of acknowledgments, priority, and authentication. An average packet is in the range of 10 to 16 bytes long, depending on the length of the domain identifier, the addressing mode, and the size of the data field for a network variable update or an explicit message. The maximum packet size is 255 bytes including data, addressing, and protocol overhead.

8.4 LONTALK ADDRESSING LIMITS

The first (top) level of the addressing hierarchy is a domain. For example, if different network applications are implemented on a shared communications medium such as RF, different domain identifiers can be used to keep the applications completely separate. The domain identifier is selectable to be 0, 1, 3, or 6 bytes long. A single node can be a member of up to two domains.

The second level of addressing is the subnet. There may be up to 255 subnets per domain. A subnet is a logical grouping of nodes from one or more channels. An intelligent router operates at the subnet level. It determines which subnets lie on which side of it, and forwards packets accordingly.

The third level of addressing is the node. There may be up to 127 nodes per subnet. Thus, a maximum of $255 \times 127 = 32,385$ nodes may be in a single domain. Any node may be a member of one or two domains, allowing a node to serve as an inter-domain gateway. This also allows, for example, a single sensor node to transmit its outputs into two different domains.

The channel (transmission medium) does not affect the way a node is addressed. Domains can contain several channels. Subnets and groups may also span several channels.

Nodes may also be grouped. Groups of nodes may span several subnets within a domain. Groups may also span channels. Up to 256 groups may be specified within a domain, and up to 64 nodes may be in a group for acknowledged service. For unacknowledged service within a domain, an unlimited number of nodes may belong to a group. A single node may be a member of up to 15 groups for receiving messages. Group addressing reduces the number of bytes of address information transmitted with each message, and also allows many nodes to receive information using a single message on the network.

In addition, each node carries a unique 48-bit Neuron Chip ID assigned during manufacture. This ID is typically used as a network address only during installation and configuration. It may also be read and used by application programs as a unique product serial number.

The channel of a node does not affect the way a node is addressed. Domains can contain many channels. Subnets and groups may also span channels.

Nodes are addressed using one of five addressing formats:

Address Data Specified	Nodes Addressed
Domain, Subnet = 0	All nodes in the domain
Domain, Subnet	All nodes in the subnet
Domain, Subnet, Node	Specific logical node
Domain, Group	All nodes in the group
Unique-ID	Specific physical node

Note that domain and subnet information are required for Neuron ID addressing for routing purposes only.

8.5 MESSAGE SERVICES

The LonTalk protocol offers four basic types of message service. The first two service types are end-to-end acknowledged. They are:

ACKD, or end-to-end acknowledged service, where a message is sent to a node or group of nodes, and individual acknowledgments are expected from each receiver. If the acknowledgments are not all received, the sender times out and retries the transaction. The number of retries and timeout are both selectable. The acknowledgments are generated by the network processor without intervention of the application. Transaction IDs keep track of messages and acknowledgments, so that an application does not receive duplicate messages.

REQUEST, or request/response service, where a message is sent to a node or group of nodes, and individual responses are expected from each receiver. The incoming message is processed by the application on the receiving side before a response is generated. The same retry and timeout options are available as with ACKD service. Responses may include data, so that this service is particularly suitable for remote procedure call, or client/server applications.

The other two service types are unacknowledged. They are:

UNACKD_RPT (unacknowledged repeated), where a message is sent to a node or group of nodes multiple times, and no response is expected. This service is typically used when broadcasting to large groups of nodes, to avoid network overload caused by all node responses.

UNACKD (unacknowledged), where a message is sent once to a node or group of nodes, and no response is expected. This is typically used when highest transmission rate is required, or when large amounts of data are to be transferred. When using this service, the application must not be sensitive to the loss of a message.

The LonTalk protocol provides duplicate message detection, and normally delivers a message to the destination application only once, even when the message has been duplicated. Duplicate packets can occur when acknowledgments and responses are lost, when packets are unintentionally overheard on open media such as RF and powerline, and when using unacknowledged repeated service. Only in the case of request/response service, where the response includes data other than the message code, is a message delivered more than once to the destination application. Duplicate detection capability is provided by a received transaction database in each node.

8.6 AUTHENTICATION

The protocol supports authenticated messages, which allow the receivers of a message to determine if the sender is authorized to send that message. Authentication prevents unauthorized access to nodes and is implemented by distributing 48-bit keys to the nodes at installation time. For an authenticated message to be accepted by the receiver, both sender and receiver must possess the same key. The key is distinct from the node's unique ID.

When an authenticated message is sent, the receiver challenges the sender to provide authentication, using a different random challenge (8 bytes) every time. The sender then responds with a transformation performed on the challenge, using the authentication key. The receiver compares the reply to the challenge with its own transformation on the challenge. If the transformations match, the transaction goes forward. The transformation used is designed so that it is extremely difficult to deduce what the key is, even if the challenge and the response are both known. The use of authentication is configurable individually for each network variable connection. In addition, network management transactions may be optionally authenticated.

8.7 PRIORITY

The LonTalk protocol optionally offers a priority mechanism to improve the response time of critical packets. The protocol permits the user to specify priority timeslots on a channel dedicated to priority nodes. Each priority timeslot on a channel adds time to the transmission of every message, but now dedicated bandwidth is available at the end of each packet for priority access without any contention for the channel.

8.8 COLLISION AVOIDANCE

The LonTalk protocol uses a unique collision avoidance algorithm which has the property that under conditions of overload, the channel can still carry its maximum capacity, rather than have its throughput degrade due to excess collisions.

8.9 COLLISION DETECTION

On communications media that support hardware collision detection (for example, twisted-pair), the LonTalk protocol can optionally cancel transmission of a packet as soon as a collision is detected by the transceiver. This allows the node to immediately retransmit any packet that has been damaged by a collision. Retransmission after a collision has been detected occurs for all classes of service. Without collision detection, the node would have to wait for the retry time before retransmitting the packet, assuming acknowledged or request/response service.

In direct mode (i.e., differential or single-ended), collisions are detected as early as 25% into the preamble up through the end of the packet. Detection of a collision does not normally terminate packet transmission. However, optionally, packets can be terminated at the end of the preamble. To reliably terminate at the end of the preamble requires that all nodes involved in a collision detect that collision during the preamble.

In special-purpose mode, collisions can be detected at any point in the packet. Packets are always terminated immediately upon report of a collision from the transceiver to the Neuron Chip. If the transceiver supports collision resolution (i.e., collision is detected early in the preamble and all colliders but one stop transmitting), the Neuron Chip is able to detect the collision and turn around to receive the victorious packet.

8.10 DATA INTERPRETATION

A foreign range of data up to 228 bytes long may be embedded in a message packet and transmitted like any other message. Although the LonTalk protocol applies no special processing for foreign frames, a special range of message codes is reserved for foreign frame transmission. These are treated as a simple array of bytes. The application program may interpret the data in any way it wishes.

8.11 NETWORK MANAGEMENT AND DIAGNOSTIC SERVICES

In addition to application message services, the LonTalk protocol provides network management services for installation and configuration of nodes, downloading of software, and diagnosis of the network. Refer to Appendix B for more detail on these services.

	Page
Appendix A Neuron Chip Configuration Data Structures	9-3
Appendix B Network Management and Diagnostic Services	9-30
Appendix C External Memory Interfacing	9-57
Appendix D Design and Handling Guidelines	9-63
Appendix E Neuron Memory Maps	9-87
Appendix F Support Tools	9-95
Appendix G Customer Alerts	9-132
Appendix H Neurowire (SPI Interface) Compatible Devices	9-136
Appendix I Echelon Trademark Usage	9-139

Appendices 9

APPENDIX A

Neuron CHIP DATA STRUCTURES

This appendix contains information on the Neuron Chip data structures and related firmware data. The bit-field ordering in all data structures presented here are from high-order to low-order bit within a byte. The byte ordering is high-order to low-order byte within a field.

The software in the Neuron Chip may be divided into three main sections: system image, application image, and network image.

THE SYSTEM IMAGE

This contains the LonTalk protocol, the Neuron C runtime library, and the task scheduler. In the MC143120 devices, this software is in the on-chip 10K ROM. In the MC143150 devices, this software is in an external storage device. For the MC143150, this software is provided as part of the LonBuilder and NodeBuilder software. Using LonBuilder or NodeBuilder software, the user can produce Intel Hex or Motorola S-record files containing the system image so that EPROM or flash devices may be programmed. Flash memory support started with version 6 of the MC143150.

In the 3120 devices, the following Neuron C runtime library functions and I/O objects are not in the on-chip ROM, but may be loaded into EEPROM along with the application program:

- access_address
- access_domain
- access_nv
- bcd2bin
- bin2bcd
- flush_wait
- go_unconfigured
- muldiv
- muldivs
- power_up
- retrieve_status
- reverse
- update_address
- update_clone_domain
- update_config_data
- update_domain
- update_nv
- use of a signed bit field
- memcpy or structure assignment (length \geq 256 bytes)
- memset (length \geq 256 bytes)
- memcpy from msg_in.data and resp_in.data
- memcpy to resp_out.data
- ability to send explicitly addressed messages

nv_in_add
dualslope input
edgelong input
infrared input
magcard input
Neurowire I/O slave mode

THE APPLICATION IMAGE

This contains the object code generated by the Neuron C compiler from the user's application program, along with other application-specific parameters. These parameters may be queried by a network management node. They include:

- Network variable fixed and self-identification data
- Network variable external interface data (XIF file)
- Program ID string
- Optional self-identification and self-documentation data
- Number of address table entries
- Number of domain table entries
- Number and size of network buffers
- Number and size of application buffers
- Number of receive transaction records
- Input clock speed of target Neuron Chip
- Transceiver type and bit rate

In the MC143150 devices, the application image is typically programmed into an external ROM, or downloaded over the network to EEPROM or flash memory. In the MC143120 devices, the application image is downloaded into the on-chip EEPROM memory. LonBuilder and NodeBuilder software supports the creation of application images.

The application image data structures described here are:

- A fixed read-only structure, whose size is independent of the application on the node
- A network variable fixed table, with entries for every network variable defined by this node
- Optional self-identification and self-documentation information describing this node and its network variables

THE NETWORK IMAGE

This contains the address assignments of the LONWORKS node, the binding information connecting network variables and message tags between the nodes in the network, parameters of the LonTalk protocol that may be set at installation time, and configuration variables of the application program. A network management node typically downloads the network image over the network into on-chip EEPROM memory when the node is installed. For simple networks, a node can update its own network image.

This section is intended for application programmers who need to understand the internal data structures of the Neuron Chip that are used for address assignment, binding, and configuration. The Neuron C application program running on the Neuron Chip may access these data using runtime library calls and declarations for the purpose of node self-installation. In the LonBuilder or NodeBuilder Neuron C development environment, function prototypes and declarations are to be found in the files `... \INCLUDE\ACCESS.H` and `... \INCLUDE\ADDRDEFS.H`.

These data structures may also be accessed by network management messages received over the network from a network management node (see Appendix B). For network management nodes running on host computers, the LonManager application programmer's interface API provides convenient high-level access to these data structures from a host-based application program. See the *LonManager API Developer's Guide* for more details. For network management tools running on any host, the LonManager NSS-10 Network Services module provides similar services. See the *LonManager API Programmer's Guide* and the *NSS-10 Developer's Guide* for more details.

The network image data structures described here are:

- A domain table, with an entry for every domain to which this node belongs
- An address table, with an entry for every network address referenced by this node
- A network variable configuration table, with entries for every network variable defined by this node
- A channel configuration structure, defining the transceiver interface of the node

ON-CHIP EEPROM LAYOUT

For reference, this section defines the relative layout of the various data structures in on-chip EEPROM. Structures in on-chip EEPROM should be accessed only from within a Neuron C program using the access routines defined in `ACCESS.H`. They may be accessed over the network using the specific network management messages defined in `NETMGMT.H`. The application program should access these structures using the built-in access functions because:

- The locations of these structures vary depending on the configuration of the application
- Writing to these structures without the necessary safeguards provided by the firmware access routines could cause the node to crash, possibly irreparably
- Future versions of the Neuron Chip may have different layouts, but the access routines will be aware of the differences

Table A-1. Data Structure and Memory Location

Structure	Location	
	MC143120B1	MC143120E1/E2/FE2/LE2 MC143150B1/B2
Fixed read-only data structure	0xF000	0xF000
Configuration data structure	0xF024	0xF029
Boot ID	0xF1FE (Neuron 3150 Chip only)	0xF1FE (Neuron 3150 Chip only)
Domain table	0xF03D	0xF042
Address table	0xF03D + 15 (decimal) bytes per domain	0xF042 + 15 (decimal) bytes per domain
Network variable configuration table	Address table + 5 bytes per address	Address table + 5 bytes per address

NOTES:

1. MC143120B1 contains version 4 firmware. MC143120 contains version 3 firmware. MC143120E2/FE2/LE2 contains version 6 firmware. MC143150 firmware version depends on the software which exported it.
2. Refer to the Reset Process and Timing section for more information on the Boot ID.

The network variable fixed table, SNVT descriptor table, and various other parts of the application image are located by the LonBuilder linker at addresses which depend on the memory map of the node. For an MC143150-based node, it is possible to specify that the off-chip ROM is just 16K bytes (64 pages), which will force the whole application image into internal EEPROM.

RAM LAYOUT

The RAM usage of an application is determined by the Neuron C linker based on the various resources used in the application, and the memory map of the node. These resource counts are also present in the application image in the node's on-chip EEPROM. On node reset, the firmware uses the resource counts to allocate structures in on-chip and off-chip RAM (if present). If the network manager wishes to reallocate RAM usage in a node and the external interface file .XIF for the node is available, then it can use the information in that file to reallocate the available RAM, as desired. If the external interface file is not available, then there is no direct way to determine the amount of available RAM. In this case, a safe strategy is to read the current RAM usage from the node, and then reallocate the resources to use no more than the current usage. All modifications to node RAM usage should be made with the node in the applicationless state. The node should be reset afterwards so that the new RAM allocation can take effect. RAM is allocated to the following data structures:

- *System Base RAM.* This memory contains stacks and system control variables. The amount of this fixed memory is a function of the Neuron Chip model; i.e., MC143120 and MC143150 and firmware version. For versions 4 and 6, the values are 454 and 636 bytes, respectively. The portion of this memory allocated to the application data and return stacks are 114 and 232 bytes, respectively.
- *Application Timers.* The number of application software timers times 4 bytes per timer.
- *Receive Transactions.* The number of receive transactions times 13 bytes per transaction record.
- *Transmit Transactions.* The number of transmit transactions times 18 bytes per transaction record. The number of transmit transactions is two if priority buffers are present, one otherwise.
- *Buffers.* Buffers are allocated in the following order:
 - Application input buffers
 - Application output buffers
 - Application output priority buffers
 - Network input buffers
 - Network output buffers
 - Network output priority buffers

The space required for each set is determined by multiplying the size of each buffer by the appropriate count.

- *I/O Changes Array.* The number of I/O changes events in the program times 3 bytes per event.
- *Application Data.* The application data is allocated from the end of RAM downward towards the end of system RAM. Note that the firmware is not aware of where the application data resides. It relies on the linker to correctly partition the RAM. Thus, increasing system RAM requirements after linking has occurred must be done with caution. To aid such an effort, a node's external interface file contains the amount of total RAM available for system usage.

The system base RAM, application timers, receive transactions, and transmit transactions must reside in on-chip RAM. The buffers and I/O changes array will move to available off-chip RAM, if there is insufficient space on-chip. Note that some fragmentation may occur as neither individual buffers nor the I/O changes array can cross the on-chip/off-chip boundary.

A.1 FIXED READ-ONLY DATA STRUCTURE

This structure defines the node identification, as well as some of the application image parameters.

Declarations from ACCESS.H and ADDRDEFS.H

```
#define NEURON_ID_LEN 6
#define ID_STR_LEN 8
typedef struct {
    unsigned    neuron_id[ NEURON_ID_LEN ];           // offset 0x00
```

```

        unsigned    model_num;                                // offset 0x06
        unsigned                                : 4;
        unsigned    minor_model_num                        : 4;    // offset 0x07
const    nv_fixed_struct * nv_fixed;                        // offset 0x08
        unsigned    read_write_protect                    : 1;    // offset 0x0A
        unsigned                                : 1;
        unsigned    nv_count                              : 6;
const    snvt_struct * snvt;                                // offset 0x0B
        unsigned    id_string[ ID_STR_LEN ];                // offset 0x0D
        unsigned    NV_processing_off                      : 1;    // offset 0x15
        unsigned    two_domains                          : 1;
        unsigned    explicit_addr                        : 1;
        unsigned                                : 5;
        unsigned    address_count                        : 4;    // offset 0x16
        unsigned                                : 4;
        unsigned                                : 4;    // offset 0x17
        unsigned    receive_trans_count                  : 4;
        unsigned    app_buf_out_size                     : 4;    // offset 0x18
        unsigned    app_buf_in_size                     : 4;
        unsigned    net_buf_out_size                     : 4;    // offset 0x19
        unsigned    net_buf_in_size                     : 4;
        unsigned    net_buf_out_priority_count           : 4;    // offset 0x1A
        unsigned    app_buf_out_priority_count           : 4;
        unsigned    app_buf_out_count                    : 4;    // offset 0x1B
        unsigned    app_buf_in_count                    : 4;
        unsigned    net_buf_out_count                    : 4;    // offset 0x1C
        unsigned    net_buf_in_count                    : 4;
        int         reserved1 [6];                          // offset 0x1D
        unsigned                                : 6;    // offset 0x23
        unsigned    tx_by_address                        : 1;
        unsigned    idempotent_duplicate                  : 1;
} read_only_data_struct;
    // The following addendum to the read-only data
    // structure is available only in the 3150 firmware
    // version 6 and later (3120 version 4)
typedef struct {
        unsigned                                : 2;    // offset 0x24
        unsigned    alias_count                        : 6;    // offset 0x25
        unsigned    msg_tag_count                      : 4;
        unsigned                                : 4;
        int         reserved2[3];
} read_only_data_struct_2;

const read_only_data_struct read_only_data;
const read_only_data_struct_2 read_only_data_2;

```

The application program may read from, but not write to these structures, using the global declaration `read_only_data` and `read_only_data_2`. The structure may be read and mostly written (except for the first 8 bytes) over the network using the *Read Memory* and *Write Memory* network management messages with `address_mode=1`. It is written during the process of downloading a new application image into the node.

Read-Only Structure Field Descriptions

```
unsigned    neuron_id[NEURON_ID_LEN];    // offset 0x00
```

This field is a 6-byte ID assigned by the manufacturer of the Neuron Chip which is unique to each Neuron Chip manufactured. Hardware prevents this field from being written after manufacture. It may be read over the network using the Query ID network management message. It is also part of the unsolicited SERVICE pin network management message.

```
unsigned    model_num;                    // offset 0x06
unsigned    minor_model_num : 4;          // offset 0x07
```

These are two fields that specify the model of the Neuron Chip. The encoding of the model number field is: MC143150 = 0, MC143120B1 = 8, 3120E1 = 9, MC143120E2 = 0x0A.

```
const nv_fixed_struct * nv_fixed;        // offset 0x08
```

This field is a pointer to the Network Variable fixed data table (see Section A.4). If there are no network variables on the node, or this node is a LONWORKS network interface (ex: MIP) using host network variable selection, then this pointer is not useful.

```
unsigned    read_write_protect: 1;        // offset 0x0A
```

This bit specifies that parts of the Neuron Chip memory may not be read or written over the network with the *Read Memory* and *Write Memory* network management messages (Section B.1.5). The application program may set this bit with the Neuron C compiler directive `#pragma read_write_protect`. If this bit is set, only the Read-Only Structure (Section A.1), the SNVT Structures (Section A.5), the Configuration Structure (Section A.6), and the application data area may be read and only the Configuration Structure may be written. The write-protected data includes the `read_write_protect` bit itself, so that once set, the bit may not be reset over the network.

```
unsigned    nv_count          : 6;
```

This field specifies the number of network variables declared in the application program running on this node (0 – 62). Each element of a network variable array is counted separately. If this node is a LONWORKS network interface (ex: MIP) using host network variable selection, this field is 0.

```
const snvt_struct * snvt;                // offset 0x0B
```

This field is a pointer to the data structure that gives self-identification information for the network variables (see Section A.5). If the self-identification information is not present, this is a null (0) pointer. If the Neuron Chip is a LONWORKS network interface (ex: MIP) using host network variable selection, this pointer is 0xFFFF. The self-identification information may be suppressed with the Neuron C compiler directive `#pragma disable_snvt_si`.

```
unsigned    id_string[ ID_STR_LEN ];      // offset 0x0D
```

This field contains an 8-byte program identifying information as specified in either of the Neuron C compiler directives:

```
#pragma set_id_string "ssssssss"
```

or

```
#pragma set_std_prog_id fm:mm:mm:cc:cc:ss:ss:nn
```

The second format is reserved for nodes that conform to the LONMARK interoperability guidelines. A program ID conforming to this format can also be generated using the program ID field in the NodeBuilder device definition window. The bit assignment for this format is as follows:

- The first 4 bits are the format code (0x8 – 0xF). Format 8 is reserved for devices passing the LONMARK conformance review and format 9 would be used by devices with standard program IDs that have not yet passed the conformance review. Formats 0xA – 0xF are reserved for future use.
- The next 20 bits are the manufacturer code — assigned to manufacturers when they become members of the LONMARK program.
- The next 16 bits are the device class — drawn from a registry of predefined class definitions defined by the LONMARK program.
- The next 16 bits are the device subclass — drawn from a registry of predefined subclass definitions defined by the LONMARK program.
- The last 8 bits are the manufacturer-assigned model number.

If the program ID string is not specified by either of the compiler directives, then it contains the name of the Neuron C source file. The program ID string may be read over the network using the Query ID network management message. It is also part of the unsolicited SERVICE pin network management message.

```
unsigned    NV_processing_off : 1;          // offset 0x15
```

This bit specifies that network variable processing is being performed off-chip in a host-based node using a LONWORKS network interface (ex: using a MIP).

```
unsigned    two_domains      : 1;
```

This bit specifies that the domain table has two entries (see Section A.2). If this bit is 0, the domain table has only one entry. This bit is set unless the Neuron C compiler directive `#pragma one_domain` was specified in the application program.

```
unsigned    explicit_addr    : 1;
```

This bit specifies that the node uses explicit message addressing in its application. If this bit is set, the application buffers contain an 11-byte explicit address field. This is set for any application using explicit addressing or the `nv_in_addr` structure.

```
unsigned    address_count    : 4;          // offset 0x16
```

This field specifies the number of entries (0 – 15) in the address table (see Section A.3).

```
unsigned    receive_trans_count : 4; // offset 0x17
```

This field specifies the number of receive transactions, as defined by the Neuron C compiler directive `#pragma receive_trans_count`. The number of receive transactions is one more than the number specified in this field. Each receive transaction uses 13 bytes of RAM.

```

unsigned  app_buf_out_size : 4;          // offset 0x18
unsigned  app_buf_in_size  : 4;
unsigned  net_buf_out_size : 4;          // offset 0x19
unsigned  net_buf_in_size  : 4;

```

These fields specify the sizes of the application and network buffers, as defined by the corresponding Neuron C compiler directives. The fields are encoded as follows:

Field Value	Buffer Size
2	20
3	21
4	22
5	24
6	26
7	30
8	34
9	42
10	50
11	66
12	82
13	114
14	146
15	210
0	255

```

unsigned  net_buf_out_priority_count : 4;    // offset 0x1A
unsigned  app_buf_out_priority_count : 4;
unsigned  app_buf_out_count          : 4;    // offset 0x1B
unsigned  app_buf_in_count           : 4;
unsigned  net_buf_out_count          : 4;    // offset 0x1C
unsigned  net_buf_in_count           : 4;

```

These fields specify the number of application and network buffers, as defined by the corresponding Neuron C compiler directives (from 0 to 191 buffers). Note that if one of the priority output buffer counts is 0, then both of them must be 0. The fields are encoded as follows:

Field Value	Buffer Count
0	0
2	1
3	2
4	3
5	5
6	7
7	11
8	15
9	23
10	31
11	47
12	63
13	95
14	127
15	191


```
unsigned tx_by_addr : 1;
```

This bit specifies that the node is maintaining a separate outgoing transaction space for each unique destination address in the address table.

```
unsigned idempotent_duplicate : 1;
```

This bit specifies that the Neuron Chip sets the idempotent retry bit in the application buffer when a request retry is sent up to the application.

```
unsigned alias_count : 6;
```

This field specifies the number of entries in the network variable alias table. If this node is a LONWORKS network interface using host network variable selection, this field is 0. Maximum value for this field is 62.

```
unsigned msg_tag_count : 4;
```

This field specifies the number of bindable message tags used by the node. The range is 0 to 15.

A.2 THE DOMAIN TABLE

This table defines the domains to which this node belongs. It is located in EEPROM, and is part of the network image written during node installation.

Declarations from ACCESS.H and ADDRDEFS.H

```
#define AUTH_KEY_LEN 6
#define DOMAIN_ID_LEN 6
typedef struct {
    unsigned id[ DOMAIN_ID_LEN ];           // offset 0x00
    unsigned subnet;                        // offset 0x06
    unsigned : 1;                           // clone_domain_bit
    unsigned node : 7;                      // offset 0x07
    unsigned len;                           // offset 0x08
    unsigned key[ AUTH_KEY_LEN ];           // offset 0x09
} domain_struct;

const domain_struct *access_domain( int index );
void update_domain( domain_struct *domain, int index );
void update_clone_domain( const domain_struct *domain, int index );
```

The application program may read or write any entry in this table using the access routines `access_domain()`, `update_domain()`, and `update_clone_domain()`. Normally, the function `update_domain()` should be used to write into the node's domain table. The function `update_clone_domain()` can be used for self-installation applications where it is not necessary to assign unique subnet and node IDs to each node. It will allow the node to receive messages from another node with the same subnet and node ID, but it will prevent the node from receiving any message addressed with subnet/node addressing mode in that domain. If `update_clone_domain()` is used, the node can only receive messages addressed with Neuron ID, group unacknowledged, or broadcast addressing modes.

The domain table consists of up to two entries, each 15 bytes in length. The default number of entries is two, which may be overridden by the Neuron C compiler directive `#pragma one_domain`. The entries in this table may be written and read over the network with the Update Domain, Leave Domain, and Query Domain network management messages.

All packets out on the network have the `clone_domain_bit` set. When cloning domains, this bit is cleared in the domain table guaranteeing that a packet, sent by a node with the same address, will not match the node's address.

Domain Table Field Descriptions

```
unsigned    id[ DOMAIN_ID_LEN ];
```

Each domain in a LONWORKS network has a unique ID of 0, 1, 3, or 6 bytes in length. If the ID is shorter than 6 bytes, it is left justified in this field. In the LonBuilder software, the user specifies the size and value of the domain ID when creating the domain object. In the NodeBuilder software, the subnet is assigned automatically when the device is created.

```
unsigned    subnet;
```

This field specifies the ID of the subnet within this domain to which this node belongs. A subnet ID may be in the range 1 – 255 for each domain. In the development environment, this is assigned automatically when the subnet object is created. A 0 is an invalid subnet ID.

```
unsigned          : 1;
```

If no clone domain is used, then the `clone_domain_bit` is set. A bit in every outgoing network packet has this bit reset. When cloning domains, this bit is cleared in the domain table guaranteeing it will not match the incoming packets, making the address different from the receiving node.

```
unsigned    node : 7;
```

This field specifies the ID of the node within this subnet (1 – 127). In the development environment, this is assigned automatically when the node specification object is created. The value 0 means that this domain table entry is not in use.

```
unsigned    len;
```

This field specifies the length of the domain ID in bytes (0, 1, 3, or 6). The value 0xFF (255) means that this domain table entry is not in use.

```
unsigned    key[ AUTH_KEY_LEN ];
```

This field specifies the 6-byte authentication key to be used in this domain for authenticated transactions. This key must match the key of all the other nodes on this domain that participate in authenticated transactions with this node. In the development environment, the user specifies the key in the node specification screen. NodeBuilder software uses a fixed key that can be changed by a network management tool. The authentication key may be incremented over the network using the *Update Key* network management message.

A.3 THE ADDRESS TABLE

This table defines the network addresses to which this node may send implicitly-addressed messages and network variables. It also defines the groups to which this node belongs. It is located in EEPROM, and is part of the network image written during node installation.

Declarations from `ACCESS.H` and `ADDRDEFS.H`

```
typedef enum { UNBOUND, SUBNET_NODE, NEURON_ID, BROADCAST } addr_type;
typedef union {
    group_struct      gp;
    snode_struct      sn;
    bcast_struct      bc;
    turnaround_struct ta;
```

```

    } address_struct;
    const address_struct *access_address( int index );
    update_address( const address_struct *address_entry, int index );
    int addr_table_index( message_tag_name );

```

The application program may read and write any entry in this table using the access routines `access_address()` and `update_address()`. The index of the address table entry corresponding to any message tag declared in the program may be determined with the function `addr_table_index`. The address table consists of up to 15 entries, each 5 bytes in length. The default number of entries is 15, which may be overridden by the Neuron C compiler directive `#pragma num_addr_table_entries nn`.

Each entry may be in one of five formats: group address, subnet/node address, broadcast address, turnaround address, or not in use. A group address is used for multicast addressing, when a network variable or message tag is used in a connection having more than two members. A subnet/node address is used for unicast addressing, when an output network variable or message tag is used in a connection with one other node. Broadcast address is not used by the LonBuilder, LonMaker, LonManager API, or NSS-10 binders when they create network variable or message tag connections. However, broadcast addressing is supported in the protocol, and may be used with explicit addressing. A turnaround address is used for network variables that are only bound to other network variables in the same node, and not to any network variables on other nodes. Destination addresses in the unique 48-bit Neuron Chip ID format are never used in the address table, but they may be used as destination addresses in explicitly addressed messages. For completeness, this format is described below. The declaration of this format is in the Neuron C include file `MSG_ADDR.H`.

In the development environment, entries in the address table are created when the network manager loads the network image into the node. The entries in this table may be read and written over the network with the Query Address and Update Address Table network management messages. An entry using group address format may be updated over the network with the Update Group Address Data network management message.

The first byte in an address table entry specifies the format of the entry:

0	not in use/turnaround format
1	(subnet, node) format
3	broadcast format
128 – 255	group format

Any bindable message tags declared in the application program are assigned to the first entries in the address table in order of declaration. This is followed by address table entries used for network variables — in the development environment, the binder assigns these entries.

The repeat timer, retry count, receive timer, and transaction timer are common to several of these address formats, and are described below in Section A.3.11.

A.3.1 Declaration of Group Address Format

```

typedef struct {
    unsigned type      : 1;           // offset 0x00
    unsigned size      : 7;
    unsigned domain    : 1;           // offset 0x01
    unsigned member    : 7;
    unsigned rpt_timer : 4;           // offset 0x02
    unsigned retry     : 4;
    unsigned rcv_timer : 4;           // offset 0x03
    unsigned tx_timer  : 4;

```

```

        unsigned   group       : 8;           // offset 0x04
    } group_struct;

```

A.3.2 Group Address Field Descriptions

```

    unsigned   type       : 1;

```

This bit is 1 for a group address, 0 for any of the other formats.

```

    unsigned   size       : 7;

```

This field specifies the size of the group (2 – 64). The size of the group includes the sender of the message. If this field is 0, then the group is of unlimited size, and unacknowledged or unacknowledged-repeated service must be used.

```

    unsigned   domain     : 1;

```

This field specifies the index into the domain table for this address (0 or 1).

```

    unsigned   member     : 7;

```

This field specifies the member ID of this node within this group (0 – 63). A group of unlimited size has 0 in this field. The member ID is used in acknowledgments to allow the sender of an acknowledged multicast message to keep track of which nodes have responded.

```

    unsigned   group      : 8;

```

This field specifies the ID of this group within this domain. A group ID may be in the range of 0 – 255. In the development environment, the group ID is allocated by the binder.

A.3.3 Declaration of Subnet/Node Address Format

```

typedef struct {
    addr_type   type;           // offset 0x00
    unsigned    domain         : 1;       // offset 0x01
    unsigned    node          : 7;
    unsigned    rpt_timer     : 4;       // offset 0x02
    unsigned    retry         : 4;
    unsigned    rcv_timer     : 4;       // offset 0x03
    unsigned    tx_timer      : 4;
    unsigned    subnet        : 8;       // offset 0x04
} snode_struct;

```

A.3.4 Subnet/Node Address Field Descriptions

```

    addr_type   type;

```

This field contains the value SUBNET_NODE (1).

```

    unsigned    domain     : 1;

```

This field specifies the index into the domain table for this address (0 or 1).

```

    unsigned    node       : 7;

```

This field specifies the node ID (1 – 127) within the specified subnet and domain. A 0 is not a valid node ID.

```

    unsigned    subnet     : 8;

```

This field specifies the subnet ID (1 – 255) within the specified domain. A 0 is not a valid subnet ID.

A.3.5 Declaration of Broadcast Address Format

```
typedef struct {
    addr_type  type;                // offset 0x00
    unsigned   domain      : 1;    // offset 0x01
    unsigned   : 1;
    unsigned   backlog     : 6;    // offset 0x01
    unsigned   rpt_timer   : 4;    // offset 0x02
    unsigned   retry       : 4;
    unsigned   : 4;                // offset 0x03
    unsigned   tx_timer    : 4;
    unsigned   subnet      : 8;    // offset 0x04
} bcast_struct;
```

A.3.6 Broadcast Address Field Descriptions

addr_type type;

This field contains the value BROADCAST (3).

unsigned domain : 1;

This field specifies the index into the domain table for this address (0 or 1).

unsigned backlog : 6;

This field specifies an estimate of the channel backlog that would be created by an acknowledged or request/response message broadcast using this address. It should be set to the expected number of acknowledgments or responses. For example, this might be the worst case number of nodes expected to respond to a QUERY_ID message on a channel. If this is unknown, this field can be set to 0, in which case a backlog of 15 is assumed.

unsigned subnet : 8;

This field specifies the subnet number (1 – 255) within the specified domain. The message is delivered to all nodes in this subnet. If the subnet number is 0, the message is delivered to all nodes in the domain. If request/response or acknowledged service is used with a broadcast address, the transaction completes as soon as the first response or acknowledgment is received. Subsequent responses or acknowledgments are discarded.

A.3.7 Declaration of Turnaround Address Format

```
typedef struct {
    addr_type  type;                // offset 0x00
    unsigned   turnaround;          // offset 0x01
    unsigned   rpt_timer; : 4;      // offset 0x02
    unsigned   retry      : 4;
    unsigned   : 4;                // offset 0x03
    unsigned   tx_timer;  : 4;
} turnaround_struct;
```

A.3.8 Turnaround Address Field Descriptions

addr_type type;

Contains the value UNBOUND (0).

```
unsigned    turnaround;
```

This field contains the value 1. If the turnaround field is 0, this address table entry is not in use.

A.3.9 Declaration of Neuron ID Address Format

Note: This format is not used in the address table, but it may be used as a destination address for an explicitly addressed message.

```
typedef struct {
    addr_type    type;                // offset 0x00
    unsigned     domain      : 1;     // offset 0x01
    unsigned     : 7;
    unsigned     rpt_timer   : 4;     // offset 0x02
    unsigned     retry       : 4;
    unsigned     : 4;               // offset 0x03
    unsigned     tx_timer    : 4;
    unsigned     subnet      : 8;     // offset 0x04
    unsigned     nid[ NEURON_ID_LEN ]; // offset 0x05
} nrnid_struct;
```

A.3.10 Neuron ID Address Field Descriptions

```
addr_type    type;
```

This field contains the value NEURON_ID (2).

```
unsigned     domain      : 1;
```

This field specifies the index into the domain table for the destination address (0 or 1). However, unique-ID addressed messages may be sent on any domain. Unconfigured nodes will receive messages addressed in unique-ID format on any domain. When this occurs, the message is said to be received on the flexible domain, and any response or acknowledgment will be sent back on the domain from which it was received, with source subnet and node identifiers of 0.

```
unsigned     subnet      : 8;
```

This field specifies the destination subnet number (1 – 255) within the domain. It is only used for routing of the message, and may be set to 0 if the message should pass through all routers in the domain.

```
unsigned     nid[ NEURON_ID_LEN ];
```

This field specifies the unique 48-bit ID of the destination Neuron Chip.

A.3.11 Timer Field Descriptions

In the development environment, the user specifies these values in the network variable or message connection screens.

```
unsigned     rpt_timer   : 4;
```

This field specifies the time interval between repetitions of an outgoing message when unacknowledged-repeated service is used. The encoding of this field is specified in Table A-2.

```
unsigned     retry       : 4;
```

This field specifies the number of retries for acknowledged, request/response, or unacknowledged-repeated service (0 – 15). The maximum number of messages sent is one more than this number.

```
unsigned   rcv_timer   : 4;
```

When the node receives a multicast (group) message, the receive timer is set to the time interval specified by this field. If a message with the same transaction ID is received before the receive timer expires, it is considered to be a retry of the previous message. The encoding of this field is specified in Table A-2.

```
unsigned   tx_timer    : 4;
```

This field specifies the time interval between retries when acknowledged or request/response service is used. The transaction retry timer is restarted when each attempt is made, and also when any acknowledgment or response (except for the last one) is received. For request/response service, the requesting node should take into account the delay necessary for the application to respond when setting the transaction timer. This is important, for example, for network management messages that write into EEPROM. The encoding of this field is specified in Table A-2.

See Section A.6.1 for a description of the non_group_timer defined in this table.

Table A-2. Encoding of Timer Field Values (ms)

Value	rpt_timer	rcv_timer	tx_timer	non_group_timer
0	16	128	16	128
1	24	192	24	192
2	32	256	32	256
3	48	384	48	384
4	64	512	64	512
5	96	768	96	768
6	128	1,024	128	1,024
7	192	1,536	192	1,536
8	256	2,048	256	2,048
9	384	3,072	384	3,072
10	512	4,096	512	4,096
11	768	6,144	768	6,144
12	1,024	8,192	1,024	8,192
13	1,536	12,288	1,536	12,288
14	2,048	16,384	2,048	16,384
15	3,072	24,576	3,072	24,576

A.4 NETWORK VARIABLE AND ALIAS TABLES

There are three tables associated with network variables: the network variable configuration table, the network variable alias table, and the network variable fixed table. The network variable configuration table defines the configurable attributes of the network variables in this node. It is located in EEPROM so that it can be modified during node installation, and is part of the network image written during node installation. The network variable alias table defines the configurable attributes of the alias network variables in the node, and is located in EEPROM, immediately following the network variable configuration table. The network variable fixed table defines the compile-time attributes of the network variables in this node. It may be located in ROM, and is part of the application image written during application download.

For a node using a LONWORKS network interface, the network variable fixed table and the network variables themselves are located in the memory of the host microprocessor. The network variable configuration and alias tables may be in either the memory of the Neuron Chip, or in the memory of the host microprocessor. In the latter case, the limit of 62 bound network variables and 62 alias network variables per node both increase to 4096, and the LonTalk protocol firmware on the Neuron Chip does not process network variable update messages, but instead passes them to the host microprocessor.

Declarations from ACCESS.H

```
#define MAX_NV 62
typedef struct {
    unsigned    nv_priority      : 1;    // offset 0x00
    unsigned    nv_direction    : 1;
    unsigned    nv_selector_hi  : 6;
    unsigned    nv_selector_lo  : 8;    // offset 0x01
    unsigned    nv_turnaround    : 1;    // offset 0x02
    unsigned    nv_service      : 2;
    unsigned    nv_auth          : 1;
    unsigned    nv_addr_index    : 4;
} nv_struct;
typedef struct {
    nv_struct    nv_cnfg;            // offset 0x00
    unsigned     primary;            // offset 0x03
    unsigned     long host_primary;  // offset 0x04
} alias_struct;

const nv_struct *access_nv( int index );
void update_nv( const nv_struct *nv_entry, int index );
const alias_struct *access_alias (int index);
void update_alias ( const alias_struct *alias_entry, int index );

typedef struct {
    unsigned     nv_sync          : 1;    // offset 0x00
    unsigned     : 2;
    unsigned     nv_length        : 5;
    void         *nv_address;        // offset 0x01
} nv_fixed_struct;
```

The application program may read from and write to any entry in the network variable configuration and alias tables using the access routines `access_nv()`, `update_nv()`, `access_alias()`, and `update_alias()`.

The index of the network variable configuration table entry corresponding to any network variable declared in the program may be determined with the function `nv_table_index ()`. For the alias table functions, the index is that of the alias entry. The base address of the network variable fixed table may be retrieved from the `read_only_data.nv_fixed` (for non-host-based nodes only).

For the network variable configuration and fixed tables on a Neuron Chip-hosted node, each of the tables consists of up to 62 entries, each 3 bytes in length. The number of entries is determined by the number of network variables declared in the application program — each element of a network variable array counts separately. For a node running the microprocessor interface program, the network variable tables are implemented on the host microprocessor and not in the Neuron Chip memory. In this case, the number of network variables is 4096. For a node running a regular application program, the index into either of these tables corresponding to a particular network variable is determined by the order of declaration of the network variables in the application program. Entries in the network variable configuration table may be read and written over the network with the *Query/Update Net Variable Configuration* network management messages. The network variable fixed table may not be written, except when downloading the application image. The application image may be in ROM for an MC143150.

The network variable alias table can have up to 62 entries, each either 4 bytes (Neuron Chip-hosted or host-based node) or 6 bytes (host-based node only) in length. The actual number of entries for a Neuron

Chip-hosted node is set through the use of the `#pragma num_alias_table_entries` compiler directive.

A.4.1 Network Variable Configuration Table Field Descriptions

```
unsigned    nv_priority      : 1;
```

This bit is set to 1 if the network variable uses priority messaging. This is specified by `bind_info(priority | nonpriority)` in the Neuron C declaration of the network variable. This may be overridden in the LonBuilder tool connection screen.

```
unsigned    nv_direction    : 1;
```

This bit is set to 1 if this is an output network variable, 0 if this is an input. This bit must not be changed by the application program or an *Update Net Variable Configuration* network management message. This bit is technically not configuration data; it resides in this table for efficiency.

```
unsigned    nv_selector_hi   : 6;
```

```
unsigned    nv_selector_lo   : 8;
```

These two fields form a 14-bit network variable selector in the range 0 – 0x3FFF. Selector values 0x3000 – 0x3FFF are reserved for unbound network variables, with the selector value equal to 0x3FFF minus the network variable index. Selector values 0 – 0x2FFF are available for bound network variables. The input network variables on any one node must all have different selectors. The output network variables on any one node must all have different selectors. The LonBuilder, LonManager, and NSS binders ensure this by assigning the same network variable selector to the union of all the connection sets in which a network variable participates.

```
unsigned    nv_turnaround    : 1;
```

This bit is set to 1 if this is a turnaround network variable; that is, bound to another network variable on the same node.

```
unsigned    nv_service       : 2;
```

This field specifies the type of service used to deliver this network variable. It is specified by `bind_info(ackd | unackd_rpt | unackd)` in the Neuron C declaration of the network variable. This may be overridden in the LonBuilder connection screen. The encoding is as follows:

ACKD	= 0	acknowledged
UNACKD_RPT	= 1	unacknowledged/repeated
UNACKD	= 2	unacknowledged

```
unsigned    nv_auth          : 1;
```

This bit is set to 1 if this network variable uses authenticated transactions. It is specified by `bind_info(authenticated | nonauthenticated)` in the Neuron C declaration of the network variable. This may be overridden in the LonBuilder connection screen.

```
unsigned    nv_addr_index    : 4;
```

This field specifies the index into the address table for this network variable (0 – 14). The value 15 (0x0F) is used if the network variable is not associated with an address table entry. Multiple network variables may use the same address table index.

A.4.2 Network Variable Alias Table Field Descriptions

```
nv_struct nv_cfg;
```

This has the same definition as in the network variable configuration table.

```
unsigned primary;
```

This is the index into the network variable configuration table. For host-based nodes, a value of 0xFF indicates that the following 2 bytes should be used for the index instead.

```
unsigned long host_primary;
```

Network variable configuration table index. This field is present only if the `primary` field is set to 0xFF. This is valid for host-based nodes. These bytes are only necessary if the network variable index used is larger than 254.

A.4.3 Network Variable Fixed Table Field Descriptions

```
unsigned nv_sync : 1;
```

This bit is set to 1 if this is a synchronous network variable. This is specified with the modifier `sync` in the Neuron C declaration of the network variable.

```
unsigned nv_length : 5;
```

This field specifies the number of bytes in the network variable (1 – 31).

```
void *nv_address;
```

This field is a pointer to the location of the variable's data in RAM or EEPROM.

A.5 THE STANDARD NETWORK VARIABLE TYPE STRUCTURES

There are five structures associated with self-identification and self-documentation, as follows:

- Fixed-size standard network variable type (SNVT) structure
- Table containing a self-identification descriptor for each network variable
- Self-documentation string for the node
- Self-documentation information for network variables
- Self-identification data for binding and status information

This information forms part of the application image written during node manufacture. If the Neuron C compiler directive `#pragma disable_snvt_si` is specified in the application program, none of this information is present. In an MC143150, these tables are normally located in ROM, if space is available. However, if the Neuron C compiler directives `#pragma snvt_si_eecode` or `#pragma snvt_si_ramcode` are specified, the SNVT structures are compiled into EEPROM or nonvolatile RAM, respectively, where they may be modified by a network management tool.

Declarations from `ACCESS.H`

```
typedef struct {
    unsigned long length;           // offset 0x00
    unsigned      num_netvars;      // offset 0x02
    unsigned      version;          // offset 0x03
    union {
        struct {
            unsigned msb_num_netvars; // offset 0x04
        };
    };
};
```

```

        unsigned mtag_count;           // offset 0x05
    } ver1;
    struct {
        unsigned mtag_count;           // offset 0x04
    } ver0;
} variable_part;
} snvt_struct;
typedef struct {
    unsigned    ext_rec      : 1;       // offset 0x00
    unsigned    nv_sync      : 1;
    unsigned    nv_polled    : 1;
    unsigned    nv_offline   : 1;
    unsigned    nv_service_type_config : 1;
    unsigned    nv_priority_config : 1;
    unsigned    nv_auth_config : 1;
    unsigned    nv_config_class : 1;
    unsigned    snvt_type_index;
} snvt_desc_struct;
typedef struct {
    unsigned    binding_II;      : 1;
    unsigned    query_stats;     : 1;
    unsigned    alias_count;     : 6;
    unsigned long host_alias;    // Host node only
} alias_field;

```

For a node running a Neuron Chip-based application program, the base address of the SNVT structure may be retrieved from the `read_only_data.snvt` read-only data field. For a node using a LONWORKS network interface, the SNVT structures are located in the memory of the host processor and may be read with the *Query SNVT* network management message (the maximum size of the four SNVT structures is 65,535 bytes). The SNVT header structure is 5 bytes long. The SNVT descriptor table follows immediately afterwards, and it has one entry for every network variable declared in the application program. Version 0 format has a separate entry for each element of a network variable array. Version 1 format allows elements of a network variable array to share a single entry. Each entry in the SNVT descriptor table is 2 bytes long.

Following the SNVT descriptor table is a null-terminated text string containing the self-documentation of the node. This string may be up to 1023 bytes in length, and it is the string specified in the Neuron C compiler directive `#pragma set_node_sd_string "sss"`. If there is no self-documentation string, the null termination byte is still present.

Following the self-documentation string for the node are extension records for those network variables that require them. For nodes with a Neuron Chip host, the self-identification and self-documentation information may be read over the network with the *Read Memory* network management command using `address_mode=0`.

Following the extension records for the network variables are the self-identification bits which reflect the state of the various binding and query status conditions.

A.5.1 SNVT Structure Field Descriptions

```
unsigned long length;
```

This field specifies the total number of bytes in the self-identification and self-documentation data structures described here.

```
unsigned          num_net_vars;
```

This field specifies the number of network variables declared on this node. Each element of a network variable array counts separately. In version 1 firmware format, this byte is the least significant byte of the number of network variables declared in this node, where an NV array counts as only one item.

```
unsigned          version;
```

This field contains the version number of the SNVT definition file used to compile the application program in this node. If the network management node's SNVT version is less than the SNVT version of the node being managed, then there may be some SNVTs in the node that are unknown to the network management node. Data types in the SNVT definition file are never deleted or modified, but new data types may be added in later versions of the file.

```
unsigned      msb_num_net_vars;
```

If the version number in the previous byte is 0, this byte is not present. If the version number is 1, this byte is the MSB of the number of network variables declared in this node. If the firmware version number is 1, this byte is the MSB of the number of network variables declared in this node, where an NV array counts as only one item.

```
unsigned          mtag_count;
```

This field specifies the number of bindable message tags declared by the application program on this node. These message tags take the first entries in the address table (see Section A.3).

A.5.2 SNVT Descriptor Table Field Descriptions

```
unsigned    ext_rec           : 1;
```

This bit is set to 1 if this network variable has an extension record following the node's self-documentation string.

```
unsigned    nv_sync                : 1;
```

This bit is set to 1 if this is a synchronous network variable. This is specified with the *sync* modifier in the Neuron C declaration of the network variable.

```
unsigned    nv_polled                : 1;
```

This bit is set to 1 if this network variable is polled. If it is an output network variable, this is specified with the *polled* modifier in the Neuron C declaration. If this is an input network variable, this means that the network variable is mentioned as the argument of a qualified *poll()* system call, or that there is an unqualified *poll()* call in the application program.

```
unsigned    nv_offline           : 1;
```

This bit is set to 1 if the network management node should take the node offline before this network variable is updated. This is specified by `bind_info(offline)` in the Neuron C declaration of the network variable.

```
unsigned nv_service_type_config : 1;
```

This bit is set to 1 if the service type of this network variable (ACKD, UNACKD, UNACKD_RPT) may be modified by a network management message. This is specified by `bind_info(service_type(config | nonconfig))` in the Neuron C declaration of the network variable.

```
unsigned    nv_priority_config      : 1;
```

This bit is set to 1 if the priority of this network variable may be modified by a network management message. This is specified by `bind_info(priority | nonpriority(config | nonconfig))` in the Neuron C declaration of the network variable.

```
unsigned    nv_auth_config          : 1;
```

This bit is set to 1 if the authentication of this network variable may be modified by a network management message. This is specified by `bind_info(auth | nonauth(config | nonconfig))` in the Neuron C declaration of the network variable.

```
unsigned    nv_config_class         : 1;
```

This bit is set to 1 if this is a configuration network variable whose value is stored in EEPROM. This is specified with the *config* modifier in the Neuron C declaration of the network variable.

```
unsigned    snvt_type_index;
```

If this is a network variable of a standard type, this field specifies the type (1 – 250). For a listing of the SNVTs, see *The SNVT Master List and Programmer's Guide* (EB173, in this data book). If this field is 0, the network variable is of a non-standard type.

A.5.3 SNVT Table Extension Records

Extension records may be present for any of the network variables. If an extension record is present, the field `ext_rec` is set to 1 in the SNVT descriptor. The extension records appear in the order that the network variables were declared in the application program. Each extension record begins with a 1-byte bit mask defining which fields are to follow. The fields follow in the order of the bits in the bit mask defined here. These bits appear in the mask starting with the MSB.

```
unsigned    mre : 1;
```

If this bit is set to 1, the extension record contains an estimate of the maximum rate at which this network variable is updated. The field is an unsigned int in the range 0 – 127, representing the rate in the range 0 – 1878.0 as specified by `bind_info(max_rate_est(nnn))` in the Neuron C declaration of the network variable. The rate is given by the formula $2^{(n/8)-5}$ messages per second, rounded to the nearest tenth of a message per second.

```
unsigned    re : 1;
```

If this bit is set to 1, the extension record contains an estimate of the average rate at which this network variable is updated. The field is an unsigned int in the range 0 – 127, representing the rate in the range 0 – 1878.0 as specified by `bind_info(rate_est(nnn))` in the Neuron C declaration of the network variable. The rate is given by the formula $2^{(n/8)-5}$ messages per second, rounded to the nearest tenth of a message per second.

```
unsigned    nm : 1;
```

If this bit is set to 1, the extension record contains the name of the network variable as declared in the Neuron C program. This information is present if the compiler directive `#pragma enable_sd_nv_names` is specified. The name is represented as a null-terminated string of up to 17 bytes, or up to 22 bytes if it is a network variable array element. Each array element has its own extension record containing the name of the array, a left square bracket, one, two, or three decimal digits denoting the index of the element, and a right square bracket.

```
unsigned    sd : 1;
```

If this bit is set to 1, the extension record contains the self-documentation string for the network variable. This is a null-terminated string of up to 1023 bytes, which may be specified by the modifier `sd_string` (`"sss"`) in the Neuron C declaration of the network variable.

```
unsigned    nc : 1;
```

If this bit is set, the extension record contains a 16-bit count of the number of network variables of this type (version 1 format only). This is used to define network variable arrays. If this bit is clear, one variable of this type is defined by this record.

A.5.4 SNVT Alias Field Descriptions

```
unsigned    binding_II : 1;
```

This bit is 1 if the node is using the new binding constraints. Nodes supporting these binding constraints do not require that a unique selector be assigned to output network variables in non-polling connections. For example, two different output network variables in a node supporting these constraints could be connected to the same input network variable in any type of node as long as that input network variable does not initiate polls. Note that polling of such output network variables by a network management or monitor node would have to be done using *Fetch Network Variable* network management messages rather than network variable poll messages.

```
unsigned    query_stats : 1;
```

This bit is 1 if the query statistics addressing mode of the *Read Memory* network management command can be used to extract the extended statistics information.

```
unsigned    alias_count : 6;
```

This field specifies the number of alias network variables used by the node (0 – 62). A value of 63 indicates that the next 2 bytes contain the actual number of alias network variables.

```
unsigned long host_alias;
```

This field specifies the number of alias network variables used by the host node (0 – 4095). This field is only present if `alias_count` is 63.

A.6 THE CONFIGURATION STRUCTURE

This structure defines the hardware and transceiver properties of this node. It is located in EEPROM, and is part of both the application image written during node manufacture and the network image written during node installation.

Declarations from `ACCESS.H`

```
#define LOCATION_LEN 6
#define NUM_COMM_PARAMS 7

typedef struct {
    unsigned long channel_id;           // offset 0x00
    char          location[ LOCATION_LEN ]; // offset 0x02
    unsigned      comm_clock           : 5; // offset 0x08
    unsigned      input_clock          : 3;
    unsigned      comm_type            : 3; // offset 0x09
    unsigned      comm_pin_dir         : 5;
    unsigned      reserved[ 5 ];       // offset 0x0A
    unsigned      node_priority;       // offset 0x0F
}
```

```

        unsigned          channel_priorities;           // offset 0x10
        union {
            unsigned xcvr_params[ NUM_COMM_PARAMS ];
            direct_param_struct  dir_params;             // offset 0x11
        }

        unsigned          non_group_timer      : 4;      // offset 0x18
        unsigned          nm_auth              : 1;
        unsigned          preemption_timeout   : 3;
    } config_data_struct;

    typedef struct {
        unsigned          collision_detect      : 1;      // offset 0x11
        unsigned          bit_sync_threshold   : 2;
        unsigned          filter                : 2;
        unsigned          hysteresis           : 3;
        unsigned          : 6;                  // offset 0x12
        unsigned          cd_tail              : 1;
        unsigned          cd_preamble          : 1;
    } direct_param_struct;

    const config_data_struct config_data;

    void update_config_data (const config_data_struct *config_data);

```

The application program may read this structure using the global declaration `config_data`, and may write it using the function `update_config_data()`. The structure is 25 bytes long, and it may be read and written over the network using the *Read Memory* and *Write Memory* network management messages with `address_mode=2`. The MAC processor reads the channel parameters (offsets 0x08 through 0x17) from the configuration structure when the node is reset and initializes the transceiver. Therefore, after changing any of the channel parameters, the node should be reset for the changes to take effect.

A.6.1 Configuration Structure Field Descriptions

```
unsigned long channel_id;
```

This field specifies the ID of the channel that this node is assigned. In the LonBuilder environment, this is assigned automatically when the channel object is created. The NodeBuilder software uses a fixed channel ID. The Neuron Chip firmware does not reference this field, but it is available to assist in recovering the network topology by interrogating the nodes.

```
char          location[ LOCATION_LEN ];
```

This location field is used to pass a 6-byte ASCII string describing the physical location of the node to the network management node. In the LonBuilder environment, it is defined in the node specification screen.

```
unsigned          comm_clock          : 5;
```

For direct mode transceivers, this field specifies the ratio between the Neuron Chip input clock oscillator frequency and the transceiver bit rate. For special-purpose mode transceivers, it specifies the rate of the bit clock between the Neuron Chip and the transceiver. In the development environment, the transceiver bit rate is specified in the channel screen. Table A-3 shows the transceiver bit rate as a function of the `input_clock` field and the `comm_clock` field.

**Table A-3. Transceiver Bit Rate (kbps) as a Function of
comm_clock and input_clock**

comm_clock	ratio	input_clock				
		5 10 MHz	4 5 MHz	3 2.5 MHz	2 1.25 MHz	1 625 kHz
0	8:1	1250	625	312.5	156.3	78.1
1	16:1	625	312.5	156.3	78.1	39.1
2	32:1	312.5	156.3	78.1	39.1	19.5
3	64:1	156.3	78.1	39.1	19.5	9.8
4	128:1	78.1	39.1	19.5	9.8	4.9
5	256:1	39.1	19.5	9.8	4.9	2.4
6	512:1	19.5	9.8	4.9	2.4	1.2
7	1024:1	9.8	4.9	2.4	1.2	0.6

unsigned input_clock : 3;

This field specifies the Neuron Chip input clock (oscillator frequency). In the LonBuilder environment, the user specifies this value in the hardware properties screen for the node. The encoding is as follows:

Input Clock	Frequency
5	10.0 MHz
4	5.0 MHz
3	2.5 MHz
2	1.25 MHz
1	625 kHz
0	Not Used

unsigned comm_type : 3;

This field specifies the type of transceiver. In the LonBuilder environment, it is specified in the channel screen. The encoding is as follows:

0	Blank Neuron Chip
1	Single-ended
2	Special-purpose
5	Differential

unsigned comm_pin_dir : 5;

This field specifies the direction of the Neuron Chip's communications port pins. A 0 indicates an input, a 1 indicates an output with respect to the Neuron Chip. The LSB corresponds to pin CP0. Values used in this field include:

0x00	Blank Neuron Chip
0x0C	Direct mode — differential
0x0E	Direct mode — single-ended
0x17	Special-purpose — wake-up pin is an input
0x1E	Special-purpose — wake-up pin is an output

unsigned reserved [5];

The following five fields specify the raw transceiver parameters used by the MAC processor. In the LonBuilder environment, these parameters are specified as raw data in the channel screen. The values in these fields are the repetition counts for software delay loops in the firmware that control the timing of the media access algorithm. In the following descriptions, the timing formulae are given in terms of "v," the

value in the control field, which may be 0 ... 255 unless otherwise specified. A Neuron Chip processor cycle is 0.6 μ s at 10 MHz input clock, 1.2 μ s at 5 MHz, and so on. See Figure 4-5 for a description of the timing of packet transmission. The fields are as follows (the numbers are determined by the transceiver design and/or the interoperability guidelines).

<code>preamble_length</code>	This field determines the length of the preamble for direct mode. Time = [(209 ... 223) + (32) v] cycles (v = 1 ... 253). The minimum preamble time for a 10 MHz input clock is 600 ns x 209 = 125 μ s. For special-purpose mode transceivers, this value should be 0.
<code>packet_cycle</code>	This field determines the packet cycle duration for counting down the backlog. Time = 1675 v cycles for direct mode, 1794 v cycles for special-purpose mode.
<code>beta2_control</code>	This field determines the <i>beta2</i> slot width. Time = 40 + 20 v cycles.
<code>xmit_interpacket</code>	This field determines the interpacket padding after transmitting. Time = 41 v cycles for v < 128, 145 (v – 128) for v \geq 128.
<code>recv_interpacket</code>	This field determines the interpacket padding after receiving. Time = 41 v cycles for v < 128, 145 (v – 128) for v \geq 128.

`unsigned node_priority;`

This field specifies the priority slot used by the node when sending priority messages on the channel (1 – 255). It should not be greater than the number of priority slots on the channel. If the node has no priority slot allocated, this is 0. In the LonBuilder environment, the node priority is specified in the hardware properties screen for the node.

`unsigned channel_priorities;`

This field specifies the number of priority slots on the channel (0 – 255). The slots are numbered starting at 1. In the LonBuilder environment, this is specified in the channel screen, with a maximum value of 127.

`unsigned xcvr_params[NUM_COMM_PARAMS];`

This field forms an array of seven transceiver-specific parameters for special-purpose mode transceivers. All seven parameters are loaded into the transceiver when the node is initialized. In the LonBuilder environment, these are specified as general purpose data in the channel screen. The MSB of the first transceiver parameter is defined to be the alternate channel bit. The alternate channel is used for the last two transmission attempts when using acknowledged, request/response, or repeated service. In the case of the powerline transceiver, this bit determines whether the transceiver uses QPSK modulation at 9600 bps (bit is 0), or BPSK modulation at 4800 bps (bit is 1). All other transceiver parameters are user-defined. For direct-mode transceivers, the first 2 bytes are overlaid by the direct-mode parameter structure defined below.

`unsigned non_group_timer : 4;`

When the node receives a unicast (non-group) or broadcast message requiring a response or acknowledgment, the receive timer is set to the time interval specified by this field. If a message with the same transaction ID, priority, source, and destination address is received before the receive timer expires, it is considered to be a retry of the previous message. In the LonBuilder environment, this is implicitly set to the maximum of the non-group receive timers specified in the connection screens (with a minimum value of 768 ms). The encoding of this field is shown in Table A-2. When network management messages using Neuron ID addressing are received, a receiver timer of 8.192 seconds is used, instead of the value of this field.

```
unsigned      nm_auth          : 1;
```

This field specifies that network management messages are to be authenticated. Setting this bit prevents the node from being configured by an unauthorized network management tool. However, network management messages received when the node is unconfigured can not be authenticated. Before setting a node's state to configured, a network management tool should ensure that the network management authentication bit is set to the desired state. In the LonBuilder environment, network management authentication is defined in the node specification screen.

```
unsigned      preemption_timeout : 3;
```

This field specifies the maximum time the node will wait for a free buffer in preemption mode. A preemption mode timeout logs an error and resets the Neuron Chip. In the LonBuilder environment, this is defined in the node specification screen. The encoding is as follows:

0	forever
1	2 seconds
2	4 seconds
3	6 seconds
4	8 seconds
5	10 seconds
6	12 seconds
7	14 seconds

A.6.2 Direct-Mode Transceiver Parameters Field Descriptions

For direct (single-ended or differential) mode transceivers, these parameters are used to control the operation of the transceiver port. In the LonBuilder environment, these parameters are specified in the channel screen. In the NodeBuilder environment, these parameters are specified in the device template.

```
unsigned      collision_detect   : 1;
```

This field specifies that the Neuron Chip monitors pin CP4 for an indication of a collision on the network.

```
unsigned      bit_sync_threshold : 2;
```

This field specifies the number of logic 1 bits received that are to be interpreted as the bit sync indicating the start of a packet. The encoding is as follows:

Threshold	Number of Bits
0	4
1	5
2	6
3	7

```
unsigned      filter            : 2;
```

For differential mode transceivers, this field specifies the setting of the receive glitch filter (0 – 3). See the electrical specifications of the Neuron Chip (Table 4-4) for details of the available glitch filter settings.

```
unsigned      hysteresis        : 3;
```

For differential mode transceivers, this field specifies the setting of the receive hysteresis filter (0 – 7). See the electrical specifications of the Neuron Chip (Table 4-3) for details of the available hysteresis filter settings.

```
unsigned      cd_to_end_packet      : 6;
```

This field controls how close to the end of a packet the collision detect signal is checked in a transmitting Neuron Chip. It is set as a function of the bit rate and the input clock rate.

```
unsigned      cd_tail                : 1;
```

This bit specifies that collisions are to be detected at the end of the transmitted packet following the code violation.

```
unsigned      cd_preamble            : 1;
```

This bit specifies that collisions are to be detected during the preamble at the beginning of the transmitted packet. When specified, the packet is terminated at the end of the preamble if a collision is detected during the preamble.

APPENDIX B

NETWORK MANAGEMENT AND DIAGNOSTIC SERVICES

In addition to application message services, the LonTalk protocol provides network management services for installation and configuration of nodes, downloading of software, and diagnosis of the network. Message codes used by the LonTalk protocol are as follows:

Message Type	Hexadecimal Message Codes
Application Messages	0x00 – 0x3E
Foreign Messages	0x40 – 0x4E
Network Diagnostics Messages	0x50 – 0x5F
Network Management Messages	0x60 – 0x73
Router Configuration Messages	0x74 – 0x7E
SERVICE Pin Message	0x7F
Network Variable Messages	0x80 – 0xFF

Response codes used by the LonTalk protocol are as follows:

Response Type	Hexadecimal Message Codes
Application Responses	0x00 – 0x3E
Response if Node is Offline	0x3F
Foreign Responses	0x40 – 0x4E
Response if Node is Offline	0x4F
Network Diagnostic Success	0x31 – 0x3F
Network Diagnostic Failure	0x11 – 0x1F
Network Management Success	0x21 – 0x3D
Network Management Failure	0x01 – 0x1D
Router Configuration Success	0x34 – 0x3E
Router Configuration Failure	0x14 – 0x1E
Network Variable Poll Responses	0x80 – 0xFF

Notice that response codes are not unique. They must be interpreted based on the original request.

Section 7.3 describes the use of application messages. A foreign message is a packet of another protocol embedded in the LonTalk protocol packet. The application program allocates space for foreign message and response codes.

Network Diagnostic Messages	Request Code	Success Response	Failed Response
Query Status	0x51	0x31	0x11
Proxy Command	0x52	0x32	0x12
Clear Status*	0x53	0x33	0x13
Query XCVR Status	0x54	0x34	0x14

*This is what the LonBuilder Developer's Workbench uses for a reset command.

Network Management Messages	Request Code	Success Response	Failed Response
Query ID	0x61	0x21	0x01
Respond to Query	0x62	0x22	0x02
Update Domain	0x63	0x23	0x03
Leave Domain	0x64	0x24	0x04
Update Key	0x65	0x25	0x05
Update Address	0x66	0x26	0x06
Query Address	0x67	0x27	0x07
Query Net Variable Config	0x68	0x28	0x08
Update Group Address Data	0x69	0x29	0x09
Query Domain	0x6A	0x2A	0x0A
Update Net Variable Config	0x6B	0x2B	0x0B
Set Node Mode	0x6C	0x2C	0x0C
Read Memory	0x6D	0x2D	0x0D
Write Memory	0x6E	0x2E	0x0E
Checksum Recalculate	0x6F	0x2F	0x0F
Wink	0x70	0x30*	0x10*
Memory Refresh	0x71	0x31	0x11
Query SNVT	0x72	0x32	0x12
Network Variable Fetch	0x73	0x33	0x13
Device Escape Code	0x7D	0x3D	0x1D

*Only for request/response messaging service.

Router Configuration Messages	0x74 – 0x7E
Router Config Success Responses	0x34 – 0x3E
Router Config Failed Responses	0x14 – 0x1E

Router messages are used by network management nodes to configure nodes that run the special router system image. They are not useful for application nodes, which will return the failed response.

SERVICE Pin Message	0x7F	(Unsolicited Message)
Network Variable Messages	0x80 – 0xFF	

Network variable messages can not be received by an application program using explicit messaging syntax. They are sent by updating network output variables in the application program, and are implicitly received by network input variables in the same connection. Polling of network variables is implemented with request/response service. For a discussion of network variable messages, see Section B.3.

Network management and network diagnostic messages may be delivered using request/response service (except for *mode online*, *mode offline*, and *wink*, which are delivered to the application processor and do not have response data associated with them). Network management messages that do not have response data associated with them may also be delivered with the other classes of service, namely acknowledged, unacknowledged, and unacknowledged-repeated. These messages are: *Respond to Query*, *Update Domain*, *Leave Domain*, *Update Key*, *Update Address*, *Update Group Address Data*, *Update Net Variable Config*, *Set Node Mode*, *Write Memory*, *Checksum Recalculate*, *Memory Refresh*, and *Clear Status*. If broadcast addressing is used with acknowledged or request/response service, then only the first acknowledgment or response can be handled.

Application messages and network variable updates are delivered with the specified class of service. Note that most network management and network diagnostic messages may be authenticated (if the `nm_auth` bit is set in the Configuration Structure). Authentication never applies to *Query ID*, *Respond to Query*, *Query Status*, or *Proxy messages*.

For Neuron C programs, these message structures are defined in the include file `NETMGMT.H`.

In the following descriptions of the network management messages, the data structures named `NM_XXX_request` specify the data field of the outgoing message (following the code field). Similarly, the data structures named `NM_XXX_response` specify the data field of the corresponding response. Network management messages are sent like any other explicit message, either from a host microprocessor or from a Neuron Chip. The following example shows how a Neuron C program running on a Neuron Chip could use the *Read Memory* network management message (see Section B.1.5) to retrieve the 6-byte Neuron ID from another Neuron Chip at offset 0x0000 into the Read-Only Structure:

```
// Note: the following structure is in the header file.

#include <ADDRDEFS.H>
#include <ACCESS.H>
#include <NETMGMT.H>

struct {
    enum {
        absolute           = 0,
        read_only_relative = 1,
        config_relative     = 2,
    } mode;
    unsigned long offset;
    unsigned count;
} read_rq; // a local copy of the request msg

msg_tag read_mem_tag; // declare a destination address

when ( reset ) {
    msg_out.code = 0x6D; // Read-memory code
    read_rq.mode = read_only_relative; // Address mode
    read_rq.offset = 0x0000; // Address offset
    read_rq.count = 6; // Byte count
    memcpy( msg_out.data, &read_rq, sizeof( read_rq ) );
    // Copy into msg_out data array

    msg_out.service = REQUEST; // Expect a response
    msg_out.tag = read_mem_tag; // Destination address
    msg_send( ); // Send the message
}

unsigned neuron_id[ 6 ]; // Place to save the returned ID

when( resp_arrives( read_mem_tag ) ) {
    memcpy( neuron_id, resp_in.data, 6 ); // copy the response data to a
    // local variable
}
```

The failed response is returned when the destination Neuron Chip can not process the message, for example when a table index is out of range, there is a memory failure when writing to EEPROM, or an attempt is made to violate read/write protection.

Sending Network Management or Diagnostic Messages

Most network management/network diagnostic (NM/ND) commands are delivered using the request/response service type. A few are limited to acknowledged service type. Throughout the messages descriptions that follow, request/response is assumed unless otherwise mentioned.

When the node is configured for network management authentication, most NM/ND transactions must be authenticated in order to take effect. However, if a node is not in the configured state, the network management authentication bit is ignored. Before setting a node's state to configured, the network manager should ensure that the network management authentication bit is set to the desired state. Network management messages that do not require authentication to be executed are so noted.

The transmit transaction timer value of the client node must be extended to handle the lengthy delays involved with any command that alters EEPROM. When Neuron ID addressing is used, the node that receives the NM or ND message automatically extends the non-group receive transaction timer to about 8 seconds. This allows the non-group receive transaction timer to be tuned for normal application traffic without concerns for lengthy network management transactions.

Addressing

Neuron ID addressed messages are received regardless of the domain in which they are sent. Unconfigured nodes will also accept any subnet or domain-wide broadcast, regardless of the domain. In either of these cases, acknowledgments and responses are returned on the domain in which the message was received with a source subnet/node pair of 0/0. Messages received in a domain in which the node is not a member (either because the node is unconfigured or is simply not in the domain) are termed as being received on a flexible domain. Some commands are not permitted under these circumstances and are noted below.

An advantage of using Neuron ID addressing for network management commands is that if a node were to accidentally go unconfigured (e.g., due to a checksum error resulting from a power cycle while changing configuration), the network manager would not lose its ability to communicate with the node.

A disadvantage of using Neuron ID addressing is that the extended receive transaction timeout could lead to subsequent false detection of duplicates. Therefore, it is recommended that, if possible, subnet/node addressing be used for network management activity. Neuron ID addressing should be used only for communicating with nodes that are not in the configured state.

Configuration Changes

The paradigm for making configuration changes is as follows:

1. Alter the node state or condition (optional).
2. Perform the change or changes. Most messages automatically update the configuration checksum. The exception is memory writes to the configuration data structure.
3. Update the configuration checksum if necessary.
4. Return to step 2 if more needs to be done.
5. Restore the node state or condition if changed in step 1.
6. Reset the node if communication parameter changes were made. Communication parameters are copied from on-chip EEPROM to RAM at node reset, so that the MAC processor can access them even during EEPROM writes.

Step 1 typically involves taking a node offline (with step 5 putting a node online). This is to ensure that the application is not accessing configuration addressing information as it is changing. It may be acceptable to eliminate this step in some circumstances.

In addition to going offline, the actual node state may be changed to unconfigured or hard-offline (with step 5 restoring it to configured). This has the advantage that, were the node to reset during the update, it would come up with the application not running. A disadvantage of making a state change is that the node state is considered to be part of the application. If it is corrupted (e.g., due to a power cycle while the state is being changed), the node will come up applicationless. If the network manager does not have the application available to reload, this can be catastrophic for the node. It is also important to note that the node should not leave the applicationless state as a result of reconfiguration alone. If the node is initially applicationless, setting it to configured will probably cause it to crash.

Application Downloading

The paradigm for downloading an application is as follows:

1. Take the node offline.
2. Alter the node state to applicationless.
3. If node went bypass offline (see Section B.1.6) in step 1, reset the node.
4. Perform a sequence of write memory commands to load the application. Writes should be limited to 11 bytes and the checksum should not be computed after each write. The order and contents of the writes should be determined by the contents of a .NXE (not NEI) load file generated by the LonBuilder export utility, with two exceptions. If the first record has a length of 1 byte, then it should not be written until after all the other records have been written. This is because it contains the read/write protect bit which could prevent further downloading if it is set. Also, if a record has a data count of 0, then the node should be reset at this point.
5. If no record with a data count of 0 was encountered in step 4 (example: 59030000FC), reset the node in order to cause the RAM to be partitioned according to the new application's requirements.
6. Compute the application checksum.
7. Enter the unconfigured state.

Before the node may be put into the configured state, the network manager should make sure that the domain table, address table, and network variable configuration table have known states. Note that after loading an application followed by loading of the configuration, a node comes up in the offline condition. To get the application running, you should initiate an online request.

Bypass offline is defined as the application checking for “offline” events directly and invoking “offline_confirm” to effect the offline condition. Normally, going offline is handled by the scheduler.

Note that node resets can take quite a while. The slower a node's input clock, the longer the reset. The more off-chip EEPROM or RAM, the longer the reset. Durations up to 18 seconds are possible in the worst case. See Section 4.6.4 for details of the reset process.

Node Resets or Power Cycles: If a node resets while a network management command is in progress, the reset will likely manifest itself as either a communication problem or a transaction failure. When EEPROM writes are involved, there is a significant probability that a location being modified at the time of the reset will become corrupted (most likely with the erase pattern of 0xFF).

Read/Write Protect Violations: If a node is read/write protected, attempts to write to the application code area are denied. The client can verify that a write memory attempt failed for this reason by reading the `read_write_protect` field of the `read_only_data` structure.

B.1 NETWORK MANAGEMENT MESSAGES

These messages are described in six main groups: node identification messages, domain table messages, address table messages, network variable-related messages, memory-related messages, and special-purpose messages.

B.1.1 Node Identification Messages

Query ID (Request/Response Only)

This message requests selected nodes to respond with a message containing their 48-bit unique ID and program ID. This message is normally broadcast during network installation to find specific nodes in the domain. It can be used to find unconfigured nodes or explicitly selected nodes, or nodes with specified memory contents at a specified address. This address may be specified absolutely, relative to the Read-Only Structure (see Section A.1), or relative to the Configuration Structure (see Section A.6). Only data within the Read-Only Structure, the SNVT Structures (see Section A.5), or the Configuration Structure may be matched. This can be used, for example, to find nodes with a particular channel ID or location string (in the Configuration Structure) or program ID string (in the Read-Only Structure). In a group request/response, the sender will accept only the first response.

Message declarations:

```
// setup request
typedef struct {
    enum {
        unconfigured          = 0,
        selected              = 1,
        selected_uncnfg       = 2, // note: selected and unconfigured
    } selector;                // with respect to an address
    enum {                     // Begin optional fields
        absolute              = 0, // all with respect to address.
        read_only_relative    = 1,
        config_relative        = 2,
    } mode;
    unsigned long offset;
    unsigned count;
    unsigned data[];
} NM_query_id_request;

// setup response
typedef struct {
    unsigned neuron_id[ NEURON_ID_LEN ]; // not optional
    unsigned id_string[ ID_STR_LEN ];
} NM_query_id_response;
```

The first byte of the request message specifies which nodes are to respond, whether unconfigured nodes, selected nodes, or nodes that are both selected and unconfigured. A node may be selected with the Respond to Query message. Optional fields may be present in the message which specify an address mode, a byte count, and an array of bytes which must match a part of the destination node's memory. For interoperability, the length of the matched region can be up to 11 bytes long. The matched region may be in the Read-Only Structure, the Configuration Structure, the SNVT Structure, or the application RAM data variable area. The address mode specifies whether the address of the matched memory is an absolute address in the memory space of the Neuron Chip, an address relative to the Read-Only Data Structure (see Section A.1), or an address relative to the Configuration Data Structure (see Section A.6). If the memory matching feature is not required, the optional fields must not be present in the message; the length of the data part of the message must be 1. Setting the *count* field to 0 does not disable the memory

matching feature. The response message contains the 6-byte Neuron ID and the 8-byte program ID. Authentication is never used with this message.

Respond to Query

This message explicitly selects or deselects a node to respond to a Query ID message. It can be used to determine network topology. Resetting the node clears the selection.

Message declaration:

```
typedef enum {
    disable    = 0,
    enable     = 1,
} NM_respond_to_query_request;
```

The request message consists of a single byte specifying whether the node should be selected or deselected. Authentication is never used with this message.

SERVICE Pin Message (Unsolicited)

This is an unsolicited message sent by a node when the **SERVICE** pin is grounded. It contains the node's unique ID followed by the program ID.

Message declaration from `netmgmt.h`:

```
typedef struct {
    unsigned neuron_id[ NEURON_ID_LEN ];
    unsigned id_string[ ID_STR_LEN ];
} NM_service_pin_msg;
```

The message data contains the 6-byte unique Neuron ID assigned by the manufacturer of the Neuron Chip, followed by the 8-byte ID of the application program in the Neuron Chip. See Section for details on these values. The **SERVICE** pin message is sent as a domain-wide broadcast on the domain whose length is 0. The source subnet and node IDs are both 0. Routers retransmit **SERVICE** pin messages on the domains in which they are configured.

B.1.2 Domain Table Messages

Update Domain

This message overwrites a domain table entry with a new value, and recomputes the configuration checksum. This assigns a domain, subnet, and node identifier to the node, as well as an authentication key for that domain. For security reasons, the authentication key should not be transmitted on an open network, where it is possible for others to tap into the network and learn the authentication key value. The node does not enter the configured state until a Set Node Mode message is sent to change its state to configured. The Update Domain message is honored even if the node is read/write protected. For a description of the domain table, see Section A.2. If the Update Domain message is received by a node on the domain that is being updated, the response is returned in the new domain. A node that receives this message can take up to 330 ms to execute the function.

Message declaration from netmgmt.h:

```
typedef struct {
    unsigned domain_index;
    unsigned id[ DOMAIN_ID_LEN ];
    unsigned subnet;
    unsigned must_be_one      :1; // clone domain bit this bit must be set to 1
    unsigned node              :7;
    unsigned len;
    unsigned key[ AUTH_KEY_LEN ];
} NM_update_domain_request;
```

The request message consists of an index into the domain table (0 or 1), followed by an image of the domain table entry to be written, in the format of a `domain_struct` declared in `\..\INCLUDE\ACCESS.H`. The MSB of the byte containing the node ID must be set for normal operation. If this bit is clear, the node is treated as a “cloned” node. That is, the node will not be able to receive messages addressed to it using subnet/node address format. It will, however, be able to receive messages from another node that has the same subnet and node IDs in that domain.

Query Domain (Request/Response Only)

This message retrieves an entry in the domain table. This message is honored even if the node is read/write protected. For a description of the domain table, see Section A.2.

Message declarations:

```
typedef unsigned /* domain_index */ NM_query_domain_request;

typedef struct {
    unsigned id[ DOMAIN_ID_LEN ];
    unsigned subnet;
    unsigned      : 1;
    unsigned node      : 7;
    unsigned len;
    unsigned key[ AUTH_KEY_LEN ];
} NM_query_domain_response;
```

The request message consists of an index into the domain table (0 or 1). The response message contains an image of the domain table entry that was read, in the format of a `domain_struct` declared in `\LB\INCLUDE\ACCESS.H`.

Leave Domain

This message deletes a domain table entry, and recomputes the configuration checksum. After the message is processed, if the node does not belong to any domain, it becomes unconfigured and is reset. This message is honored even if the node is read/write protected. For a description of the domain table, see Section A.2. If the Leave Domain message is received by a node on the domain which is to be left, no response is returned. If the message causes the node to leave the last domain where it is configured, its state becomes unconfigured. A node that receives this message can take up to 330 ms to execute the function.

Message declaration from netmgmt.h:

```
typedef unsigned /* domain_index */ NM_leave_domain_request;
```

The request message consists of an index into the domain table (0 or 1).

Update Key

This message adds an increment to the current encryption key in a domain table entry to form a new key, and recomputes the configuration checksum. This allows rekeying of a node without having to transmit the new key on an open or public network. This message is honored even if the node is read/write protected. For a description of the domain table, see Section A.2. Senders of this message should be especially careful that the message is not received more than once, since its function is incremental. A node that receives this message can take up to 150 ms to execute this function.

Message declaration from `netmgmt.h`:

```
typedef struct {
    unsigned domain_index;
    unsigned key[ AUTH_KEY_LEN ];
} NM_update_key_request;
```

The request message consists of an index into the domain table (0 or 1), followed by 6 bytes of authentication key increment.

B.1.3 Address Table Messages

Update Address

This message overwrites an address table entry with a new value, and recomputes the configuration checksum. An address table entry allows the node to implicitly address another node, or join a group. This message is honored even if the node is read/write protected. For a description of the address table, see Section A.3. A node that receives this message can take up to 130 ms to execute the function.

Message declaration from `netmgmt.h`:

```
typedef struct {
    unsigned addr_index;
    unsigned type;                               //addr_type or (0x80 |group_size)
    unsigned domain          : 1;
    unsigned member_or_node : 7;
    unsigned rpt_timer       : 4;
    unsigned retry           : 4;
    unsigned rcv_timer       : 4;
    unsigned tx_timer        : 4;
    unsigned group_or_subnet;
} NM_update_addr_request;
```

The request message consists of an index into the address table (0 to 14), followed by an image of the address table entry to be written, in the format of an `address_struct` declared in `\..\INCLUDE\ACCESS.H`. The address type field is 0 for unbound, 1 for subnet_node, 3 for broadcast, and for group addressing it contains the group size with the MSB set.

Query Address (Request/Response Only)

This message retrieves an entry in the address table. This message is honored even if the node is read/write protected. For a description of the address table, see Section A.3.

Message declarations from `netmgmt.h`:

```
typedef unsigned /* addr_index */ NM_query_addr_request;

typedef struct {
    unsigned type;                                //addr_type or (0x80 |group_size)
    unsigned domain          : 1;
    unsigned member_or_node : 7;
    unsigned rpt_timer       : 4;
    unsigned retry           : 4;
    unsigned rcv_timer       : 4;
    unsigned tx_timer        : 4;
    unsigned group_or_subnet;
} NM_query_addr_response;
```

The request message consists of an index into the address table (0 to 14). The response message contains an image of the address table entry that was read, in the format of an `address_struct` declared in `\LB\INCLUDE\ACCESS.H`. The address type field is 0 for unbound, 1 for subnet_node, 3 for broadcast, and for group addressing it contains the group size with the MSB set.

Update Group Address Data

This message updates a group entry in the address table with a new group size and timer fields, and recomputes the configuration checksum. The message is sent to all members of the group, and updates the corresponding entry in the address table. This is used when nodes join or leave the group. This message is honored even if the node is read/write protected. For a description of the address table, see Section A.3. A node that receives this message can take up to 130 ms to execute the function.

Message declaration from `netmgmt.h`:

```
typedef struct {
    unsigned type          : 1;          //must be 1
    unsigned size          : 7;
    unsigned domain        : 1;
    unsigned member        : 7;
    unsigned rpt_timer     : 4;
    unsigned retry         : 4;
    unsigned rcv_timer     : 4;
    unsigned tx_timer      : 4;
    unsigned group;
} NM_update_group_addr_request;
```

The request message consists of an image of the address table entry to be written, and must be delivered with group addressing. The group size and timer values are updated with new values, but the member number and domain index are left unchanged. The group number must not change.

B.1.4 Network Variable-Related Messages

Network variable update and poll messages are described in Section B.3.

For the network variable-related messages that have an optional 16-bit field following a 1-byte network variable index (*Update Net Variable Config*, *Query Net Variable Config*, and *Network Variable Fetch*), the 2 optional bytes are only present if the 1-byte index is 0xFF. This is valid only for host-based nodes. The 2 optional bytes are only necessary if the network variable index used is larger than 254.

Update Net Variable Config

This message overwrites a network variable configuration or address table entry with a new value, and recomputes the configuration checksum. This assigns a network variable selector to effect the binding of the network variable to network variables with the same selector on other nodes. This message is honored even if the node is read/write protected. For a description of the network variable configuration table, see Section A.4. For a node using a LONWORKS network interface with host selection enabled (ex: MIP with the network variable configuration table on the host), the *Update Net Variable Config* message is passed to the host microprocessor. In this case, the network variable index value of 255 is reserved to indicate that the following 2 bytes in the message form a 16-bit network variable index, allowing up to 16,383 network variables to be bound. Note, the MIP only supports up to and including 4096 network variables.

Values 0 – 0xFFFF are valid network variable configuration table indices, allowing up to 4096 network variable table entries to be updated. For updating the network variable alias table, the actual index used is the index of the alias table plus the `nv_count`. Values `(nv_count) – (nv_count + 0xFFFF)` are valid network variable alias table indices, allowing up to 4096 network variable alias table entries to be updated. A Neuron Chip-hosted node that receives this message can take up to 110 ms to execute the function.

```
typedef struct {
    unsigned nv_index;
    unsigned nv_priority    : 1;
    unsigned nv_direction  : 1;
    unsigned nv_selector_hi : 6;
    unsigned nv_selector_lo : 8;
    unsigned nv_turnaround  : 1;
    unsigned nv_service     : 2;
    unsigned nv_auth        : 1;
    unsigned nv_addr_index  : 4;
} NM_update_nv_cnfg_request;
```

The request message consists of an index into the network variable table, followed by an image of the network variable configuration table entry to be written, in the format of an `nv_struct` declared in `..\INCLUDE\ACCESS.H`.

Query Net Variable Config (Request/Response Only)

This message retrieves an entry in the network variable configuration table. This message is honored even if the node is read/write protected. For a description of the network variable configuration table and alias table, see Section A.4. For a node using a LONWORKS network interface with host selection enabled (ex: MIP with the network variable configuration table on the host), the *Query Net Variable Config* message is passed to the host microprocessor. In this case, the network variable index value of 255 is reserved to indicate that the following 2 bytes in the message form a 16-bit network variable index, allowing up to 16,383 network variable configuration table entries to be queried.

Values 0 – 0xFFFF are valid network variable configuration table indices, allowing up to 4096 network variable table entries to be updated. For updating the network variable alias table, the actual index used is the index of the alias table plus the `nv_count`. Values `(nv_count) – (nv_count + 0xFFFF)` are valid network variable alias table indices, allowing up to 4096 network variable alias table entries to be updated.

Message declaration from `netmgmt.h`:

```
typedef unsigned /* nv_index */          NM_query_nv_cnfg_request;

typedef struct {
    unsigned nv_priority      : 1;
    unsigned nv_direction    : 1;
    unsigned nv_selector_hi   : 6;
    unsigned nv_selector_lo   : 8;
    unsigned nv_turnaround    : 1;
    unsigned nv_service       : 2;
    unsigned nv_auth          : 1;
    unsigned nv_addr_index    : 4;
} NM_query_nv_cnfg_response;
```

The request message consists of an index into the network variable configuration table. The response message contains an image of the network variable configuration table entry that was read, in the format of an `nv_struct` declared in `\..\INCLUDE\ACCESS.H`.

Query SNVT (Request/Response Only)

This message retrieves self-identification and self-documentation data from the host processor memory of a node using a LONWORKS network interface (ex: MIP). The MIP is a special application used to attach the node to a host processor at layer 4 of the protocol. In these cases, the address table is located in the Neuron Chip memory, but the network variable fixed table and SNVT information is located in the host's memory. The specified amount of data is retrieved from the specified offset into the SNVT Structure on the host. The number of bytes read in one message is limited by the network buffer sizes on both Neuron Chips. For interoperability, this should be limited to 16 bytes. For a Neuron Chip-hosted node, the *Query SNVT* message will return the failed response. In this case, the *Read Memory* message should be used to retrieve the SNVT information using the `snvt` pointer in the Read-Only Structure (see Section A.1). For a description of the SNVT Structure, see Section A.5.

Message declarations from `netmgmt.h`:

```
typedef struct {
    unsigned long offset;
    unsigned count;
} NM_query_SNVT_request;

typedef unsigned NM_query_SNVT_response[ ];
```

The request message consists of two bytes specifying the offset into the addressed memory, and a byte specifying the number of bytes to be retrieved. The address is passed with the most significant byte first, whether or not this is the native address format of the host microprocessor. The response message contains the data in the memory that was read.

Network Variable Fetch (Request/Response Only)

This message retrieves the value of a network variable by its index into the network variable tables. This can be used to poll the value of a network variable, even if the node is offline. For a description of the network variable tables, see Section A.4. The normal way to poll a network variable value is with an application message specifying the network variable's selector (see Section B.3). For a node using a LONWORKS network interface (ex: MIP), the *Network Variable Fetch* message is passed to the host microprocessor. In this case, the network variable index value of 255 is reserved to indicate that the following 2 bytes in the message form a 16-bit network variable index, allowing up to 16,383 network variables to be fetched. Only values 0 – 0xFFFF are valid, allowing up to 4096 network variables to be fetched.

Message declarations from `netmgmt.h`:

```
typedef unsigned /* nv_index */ NM_NV_fetch_request;

typedef struct {
    unsigned nv_index;
    unsigned data[];
} NM_NV_fetch_response;
```

The request message consists of the network variable index, which is the index into either of the network variable tables. The response message contains the network variable index, followed by the network variable data itself.

B.1.5 Memory-Related Messages

Read Memory (Request/Response Only)

This message reads data from the node's memory. Addresses may be specified relative to the Read-Only Structure (see Section A.1), relative to the Configuration Structure (see Section A.6), relative to the Query Statistics Structure (see below), or absolutely. To read the domain table, the address table, or the network variable configuration table, the *Query Domain/Address/NV Config* messages should be used. The number of bytes that can be read in one message is limited only by the network buffer sizes on both Neuron Chips. If the node is read/write protected, none of the node's memory may be read, except for the Read-Only Structure (see Section A.1), the SNVT Structures (see Section A.5), and the Configuration Structure (see Section A.6).

Message declaration from `netmgmt.h`:

```
typedef struct {
    enum {
        absolute                = 0,
        read_only_relative      = 1,
        config_relative          = 2,
        statistics_relative      = 3,
    } mode;
    unsigned long offset;
    unsigned count;
} NM_read_memory_request;

typedef unsigned NM_read_memory_response[];
```

The request message consists of a byte specifying the address mode, 2 bytes specifying the offset into the addressed memory (most significant byte of the address first), and a byte specifying the number of bytes to be read. The response message contains the data in the memory that was read.

The following structure definition should be used for a read memory command with statistics-relative addressing mode:

```
typedef struct {
    unsigned long    transmission_errors;
    unsigned long    transmit_tx_failures;
    unsigned long    receive_tx_full;
    unsigned long    lost_messages;
    unsigned long    missed_messages;
    unsigned long    layer2_received;
    unsigned long    layer3_received;
```



```

        unsigned long    layer3_transmitted;
        unsigned long    transmit_tx_retries;
        unsigned long    backlog_overflows;
        unsigned long    late_acknowledgments;
        unsigned long    collisions;
        unsigned int     eeprom_lock;
    } stats_struct;

```

The fields for the above structure are defined as follows:

```

        unsigned long    transmission_errors;

```

The number of CRC errors detected during packet reception. These may be due to collisions or noise on the transceiver input.

```

        unsigned long    transmit_tx_failures;

```

The number of times that the node failed to receive expected acknowledgments or responses after retrying the configured number of times. These may be due to destination nodes being inaccessible on the network, transmission failures because of noise on the channel, or any destination node having insufficient buffers or receive transaction records. When using request/response service or network variable polling, a transaction timeout can also occur if the destination node application program does not return to the scheduler frequently enough because responses are synchronized with the application tasks.

```

        unsigned long    receive_tx_full;

```

The number of times that an incoming packet was discarded because there was no room in the transaction database. These may be due to excessively long receive timers (Section A.3.11), or inadequate size of the transaction database.

```

        unsigned long    lost_messages;

```

The number of times that an incoming packet was discarded because there was no application buffer available. These may be due to an application program being too slow to process incoming packets, to insufficient application buffers, or to excess traffic on the channel. If the incoming message is too large for the application buffer, an error is logged, but the lost message count is not incremented.

```

        unsigned long    missed_messages;

```

The number of times that an incoming packet was discarded because there was no network buffer available. These may be due to excess traffic on the channel, to insufficient network buffers, or to the network buffers not being large enough to accept all packets on the channel, whether or not addressed to this node.

```

        unsigned long    layer2_received;

```

The number of layer 2 messages received by this node. Layer 2 messages are those that have correct CRC and can be addressed to any node.

```

        unsigned long    layer3_received;

```

The number of layer 3 messages received by this node. Layer 3 messages are those layer 2 messages that are addressed to this node.

```

        unsigned long    layer3_transmitted;

```

The number of messages transmitted from layer 3 of the Neuron Chip. These can include network variable updates, explicit messages, acknowledgments, retries, reminders, SERVICE pin messages, and any other type of message.

```
unsigned long    transmit_tx_retries;
```

The number of retries sent by this node. This does not include retries used for messages sent with the repeated service.

```
unsigned long    backlog_overflows;
```

The number of times the backlog reached its maximum value of 63.

```
unsigned long    late_acknowledgments;
```

The number of acknowledgments or responses that arrived at this node after the transmit transaction had expired.

```
unsigned long    collisions;
```

This field specifies the number of occurrences of either collision detection or collision resolution, if those features are enabled.

```
unsigned int     eeprom_lock;
```

The state of the EEPROM lock for the node. If this field is a 1, then the checksummed EEPROM on the node is protected against memory write.

Write Memory

This message writes data to the node's memory. Addresses may be specified relative to the Read-Only Structure (see Section A.1), relative to the Configuration Structure (see Section A.6), or absolutely. This message may be used to download an application program to read/write memory on the node. The number of bytes that can be written in one message is limited by the network buffer sizes on both Neuron Chips. To write the domain table, the address table, or the network variable configuration table, the *Update Domain/Address/NV Config* messages should be used. The number of bytes that can be written in one message is limited by the network buffer sizes on both Neuron Chips. For interoperability, this should be limited to 11 bytes. If writing to EEPROM, the application checksum and the configuration checksum may be recalculated. In this case, the number of bytes written in one message should be limited to 38 to avoid watchdog timeouts at maximum input clock rate. The node may also be reset. If the node is read/write protected, none of the node's memory may be written except for the Configuration Structure.

Message declaration:

```
typedef struct {
    enum {
        absolute           = 0,
        read_only_relative = 1,
        config_relative     = 2,
        statistics_relative = 3,
    } mode;
    unsigned long offset;
    unsigned count;
    enum {
        no_action           = 0,
        both_cs_recalc      = 1,
        cnfg_cs_recalc      = 4,
        only_reset          = 8,
        both_cs_recalc_reset = 9,
        cnfg_cs_recalc_reset = 0xC,
    } form;
    unsigned data[];
} NM_write_memory_request;
```

The request message consists of a byte specifying the address mode, 2 bytes specifying the offset into the addressed memory (most significant byte of the address first), a byte specifying the number of bytes to be written, and a byte specifying the action to be taken at the completion of the write operation, followed by the data bytes to be written.

The following structure definition should be used for a read memory command with statistics-relative addressing mode:

```
typedef struct {
    unsigned long    transmission_errors;
    unsigned long    transmit_tx_failures;
    unsigned long    receive_tx_full;
    unsigned long    lost_messages;
    unsigned long    missed_messages;
    unsigned long    layer2_received;
    unsigned long    layer3_received;
    unsigned long    layer3_transmitted;
    unsigned long    transmit_tx_retries;
    unsigned long    backlog_overflows;
    unsigned long    late_acknowledgments;
    unsigned long    collisions;
    unsigned long    eeprom_lock
} stats_struct;
```

The fields for the above structure are defined as follows:

```
unsigned long    transmission_errors;
```

The number of CRC errors detected during packet reception. These may be due to collisions or noise on the transceiver input.

```
unsigned long    transmit_tx_failures;
```

The number of times that the node failed to receive expected acknowledgments or responses after retrying the configured number of times. These may be due to destination nodes being inaccessible on the network, transmission failures because of noise on the channel, or any destination node having insufficient buffers or receive transaction records. When using request/response service or network variable polling, a transaction timeout can also occur if the destination node application program does not return to the scheduler frequently enough because responses are synchronized with the application tasks.

```
unsigned long    receive_tx_full;
```

The number of times that an incoming packet was discarded because there was no room in the transaction database. These may be due to excessively long receive timers (see Section A.3.11), or to inadequate size of the transaction database.

```
unsigned long    lost_messages;
```

The number of times that an incoming packet was discarded because there was no application buffer available. These may be due to an application program being too slow to process incoming packets, to insufficient application buffers, or to excess traffic on the channel. If the incoming message is too large for the application buffer, an error is logged, but the lost message count is not incremented.

unsigned long missed_messages;

The number of times that an incoming packet was discarded because there was no network buffer available. These may be due to excess traffic on the channel, to insufficient network buffers, or to the network buffers not being large enough to accept all packets on the channel, whether or not addressed to this node.

unsigned long layer2_received;

The number of layer 2 messages received by this node. Layer 2 messages are those that have correct CRC and can be addressed to any node.

unsigned long layer3_received;

The number of layer 3 messages received by this node. Layer 3 messages are those layer 2 messages that are addressed to this node.

unsigned long layer3_transmitted;

The number of messages transmitted from layer 3 of the Neuron Chip. These can include network variable updates, explicit messages, acknowledgments, retries, reminders, $\overline{\text{SERVICE}}$ pin messages, and any other type of message.

unsigned long transmit_tx_retries;

The number of retries sent by this node. This does not include retries used for messages sent with the repeated service.

unsigned long backlog_overflows;

The number of times the backlog reached its maximum value of 63.

unsigned long late_acknowledgments;

The number of acknowledgments or responses that arrived at this node after the transmit transaction had expired.

unsigned long collisions;

The number of occurrences of either collision detection or collision resolution, if those features are enabled.

unsigned int eeprom_lock;

The state of the EEPROM lock for the node. If this field is a 1, then the checksummed EEPROM on the node is protected against memory write.

Checksum Recalculate

This message recomputes either the network image checksum, or both the network and application image checksums in EEPROM. The application image and the network image are independently checked whenever the node is reset. They are also checked by a continually running background task. If the configuration checksum is invalid, the node enters the unconfigured state. If the application checksum is invalid, the node enters the applicationless state. The LonBuilder compiler will generate the configuration and application checksums. When a program is loaded over the network, these two checksums are also loaded. After reset and before program execution the checksums are verified. Refer to Section 4.6.4, for more information.

The network variable messages to update the domain, address, or network variable tables automatically update the configuration checksum. The *Checksum Recalculate* message is intended for use after a series

of *Write Memory* messages, for example, when downloading an application program. The checksum recalculate is used after a LonBuilder memory write command.

Message declaration:

```
typedef enum {
    both_cs = 1,           // Application image and network image checksums
    cnfg_cs = 4,           // Network image checksum only
} NM_checksum_recalc_request;
```

The request message consists of a single byte specifying which checksums should be recalculated.

Memory Refresh

This message rewrites EEPROM memory at the specified offset. Either on-chip or off-chip EEPROM may be refreshed. The number of bytes refreshed in one message should be limited to 38 if the node is offline, 8 bytes if the node is online, to avoid watchdog timeouts at maximum input clock rates. This message may be used periodically to extend the 10-year data retention of most EEPROM devices. Note that most EEPROM devices are specified as supporting 10,000 write cycles, therefore this message should not be used very frequently. The 48-bit Neuron ID can not be refreshed. The *Memory Refresh* message returns the failed response if the address specified is outside of the on-chip or off-chip EEPROM regions.

Message declaration from `netmgmt.h`:

```
typedef struct {
    unsigned long offset;
    unsigned count;
    enum {
        ON_chip = 0,
        OFF_chip = 1,
    } which;
} NM_memory_refresh_request;
```

The request message consists of 2 bytes specifying the offset into the addressed memory (most significant byte of the address first), a byte specifying the number of bytes to be refreshed, and a byte indicating whether on-chip or off-chip EEPROM is to be refreshed.

B.1.6 Special-Purpose Messages

Set Node Mode (Service Class Varies)

A Neuron Chip's condition consists of two parts: state and mode. Generally, the node state is preserved across resets, and node mode is not. This message puts the application in an offline mode, changes the state of the node, or resets the node.

A Neuron Chip can be in one of six states. These states are maintained in EEPROM and are as follows:

Node State	State Code	Service LED
Applicationless and Unconfigured	3	On
Unconfigured (but with an Application)	2	Flashing
Configured, Hard Offline	6	Off
Configured	4	Off

Applicationless and Unconfigured (3): No application is loaded yet, the application is in the process of being loaded, or the application is deemed corrupted due to application checksum error or signature inconsistency. The application does not run in this state. The service LED is steadily on in this state.

Unconfigured (2): The application is loaded but the configuration either is not loaded, is being reloaded, or is deemed corrupted due to configuration checksum error. A program can make itself unconfigured by calling the `go_unconfigured ()` function. The service LED flashes at a 1-second rate in this state.

Configured, Hard Offliine (6): The application is loaded but not running. The configuration is considered valid in this state; the network management authentication bit is honored. The service LED is off in this state.

Configured (4): Normal Node State. The application is running and the configuration is considered valid. This is the only state in which messages addressed to the application are received. In all other states, they are discarded. The service LED is off in this state.

The configured state has an additional modifier, which is the online/offline mode. This mode is **not** maintained in EEPROM. The states and online/offline condition are controlled via different mechanisms. However, they are reported together in the *Query Status* network management message. The configured/offline condition is also known as soft offline.

A node that is in the soft offline state will go online when it is reset. The hard offline state is preserved across a reset. When the application is of either type offline, the scheduler is disabled. When soft offline, polling a network variable will return null data, but incoming network variable updates will be processed normally, except that the `nv_update_occurs` events will be lost. In all states other than configured, no response is returned to network variable polls and incoming network variable updates are discarded.

If a node is in a non-configured state, is reset, and is then issued a command to go configured, it will come up in a soft offline condition.

If a set node mode message changes the mode to offline or online, the appropriate Neuron C task (if any) will be executed. *Mode online* and *mode offline* messages must not be delivered with request/response service. There will be no response to a *Reset* message, since the node is reset immediately. Changing the state of a node recomputes the configuration checksum, which takes some time, so verification of state changes should always be made with the *Query Status* message.

Message declaration from `netmgmt.h`:

```
typedef struct {
    enum {
        appl_offline           = 0,    // soft offline state
        appl_online            = 1,
        appl_reset             = 2,
        change_state           = 3,
    } mode;
    enum {
        appl_uncnfg            = 2,
        no_appl_uncnfg         = 3,
        cnfg_online            = 4,
        cnfg_offline           = 6,    // hard offline state
    } node_state;              // Optional field if mode = 3
} NM_set_node_mode_request;
```

The request message consists of a byte specifying whether this is a request to put the node in the soft offline state or online mode, reset it, or change the state of the Neuron Chip. In the last case, there is a second byte specifying which state the node should enter.

Wink (Any Service Class Except Request/Response)

This message has two formats. If the message is sent with no data, it causes the receiving node to execute the *wink* clause in the application program (if any). This may be used to identify nodes visually or audibly if it is more convenient than grounding the $\overline{\text{SERVICE}}$ pin. Wink messages may not be delivered with request/response service.

An example on a node receiving a wink message to turn on (off) the service LED is:

```
boolean service_status;           // state of service pin
when (wink)
{
    service_status = 1 - service_status;           // keeping track of
                                                    // current STATE of
                                                    // service pin LED.
    activate_service_led = service_status;         // set service
                                                    // pin output.
}
```

Do not confuse the state of the $\overline{\text{SERVICE}}$ pin LED when the wink command is sent. The $\overline{\text{SERVICE}}$ pin LED is used, in this case, to signify reception of a wink command. Once the node is identified, the Neuron 48-bit ID can be extracted. Toggle the $\overline{\text{SERVICE}}$ pin again to put the service LED back into its original state.

The second format of this message is used only with host-based nodes, that is; nodes implemented with a LONWORKS network interface on a host processor. This format is needed when installing application nodes with multiple network interfaces attached. When installing a host-based node, depressing a $\overline{\text{SERVICE}}$ pin results in a single $\overline{\text{SERVICE}}$ pin message being generated. A network management toll can send this command to interrogate the node about any additional network interfaces that might be attached.

The first byte of the host-node format specifies a subcommand, which may be either a wink message or a request to send identifying information from the host. The *wink* subcommand may not be delivered with request/response service. The *send_id_info* subcommand should be delivered with request/response service, and the following byte in the message specifies a network interface number to support hosts that may have multiple network interfaces.

For host-based nodes, all forms of this message are passed to the host application, where they should be handled appropriately, either by executing a wink function or responding with the Neuron ID and program ID for the specified network interface. The first byte of the response should be 0 if the specified network interface is functional, non-zero otherwise.

Message declaration from `netmgmt.h`:

```
typedef struct {                               // used for host-based nodes only
    enum {
        WINK                = 0,
        SEND_ID_INFO        = 1,
    } subcommand;
    unsigned network_interface; // present if subcommand = SEND_ID_INFO
} NM_wink_request;

typedef struct {                               // response to SEND_ID_INFO from a host
                                           node
    boolean interface_down;
    unsigned neuron_id[ NEURON_ID_LEN ];
    unsigned id_string[ ID_STR_LEN ];
} NM_wink_response;
```

B.2 NETWORK DIAGNOSTIC MESSAGES

Query Status (Request/Response Only)

This message retrieves the network error statistics accumulators, the cause of the last reset, the state of the node, and the last runtime error logged. This message is used after a node has been reset to verify that the reset has occurred, since resets are not acknowledged.

Message declaration:

```
typedef struct {
    unsigned long xmit_errors;           // offset 0x00
    unsigned long transaction_timeouts;  // offset 0x02
    unsigned long rcv_transaction_full;  // offset 0x04
    unsigned long lost_msgs;             // offset 0x06
    unsigned long missed_msgs;           // offset 0x08
    unsigned reset_cause;                 // offset 0x0A
    unsigned node_state;                  // offset 0x0B
    unsigned version_number;              // offset 0x0C
}

enum {
    no_error = 0,
    bad_event = 129,
    nv_length_mismatch = 130,
    nv_msg_too_short = 131,
    eeprom_write_fail = 132,
    bad_address_type = 133,
    preemption_mode_timeout = 134,
    already_preempted = 135,
    sync_nv_update_lost = 136,
    invalid_resp_alloc = 137,
    invalid_domain = 138,
    read_past_end_of_msg = 139,
    write_past_end_of_msg = 140,
    invalid_addr_table_index = 141,
    incomplete_msg = 142,
    nv_update_on_output_nv = 143,
    no_msg_avail = 144,
    illegal_send = 145,
    unknown_PDU = 146,
    invalid_nv_index = 147,
    divide_by_zero = 148,
    invalid_appl_error = 149,
    memory_alloc_failure = 150,
    write_past_end_of_net_buffer = 151,
    appl_cs_error = 152,
    cnfg_cs_error = 153,
    invalid_xcvr_reg_addr = 154,
    xcvr_reg_timeout = 155,
    write_past_end_of_appl_buffer = 156,
    io_ready = 157,
    self_test_failed = 158,
    subnet_router = 159,
    Authentication_mismatch = 160,
    self_inst_semaphore_set = 161,
```



```

        read_write_semaphore_set          = 162,
        appl_signature_bad                = 163,
        router_firmware_version_mismatch  = 164,
    } error_log;                          // offset 0x0D
    unsigned model_number;                 // offset 0x0E
} ND_query_status_response;

```

The request message contains no data. The response message begins with five 16-bit error statistics accumulators as follows:

Transmission Errors — The number of CRC errors detected during packet reception. These may be due to collisions or noise on the transceiver input.

Transaction Timeouts — The number of times that the node failed to receive expected acknowledgments or responses after retrying the configured number of times. These may be due to destination nodes being inaccessible on the network, transmission failures because of noise on the channel, or if any destination node has insufficient buffers or receive transaction records. When using Request/Response service or network variable polling, a transaction timeout can also occur if the destination node application program does not return to the scheduler frequently enough, because responses are synchronized with the application tasks.

Receive Transaction Full Errors — The number of times that an incoming packet was discarded because there was no room in the transaction database. This may be due to excessively long receive timers (see Section A.3.11), or inadequate size of the transaction database.

Lost Messages — The number of times that an incoming packet was discarded because there was no application buffer available. This may be due to an application program being too slow to process incoming packets, insufficient application buffers, or excess traffic on the channel. If the incoming message is too large for the application buffer, an error is logged, but the lost message count is not incremented.

Missed Messages — The number of times that an incoming packet was discarded because there was no network buffer available. This may be due to excess traffic on the channel, insufficient network buffers, or the network buffers not being large enough to accept all packets on the channel, whether or not addressed to this node.

The response message also contains a byte with the cause of the last reset as follows (X = don't care):

Power-up reset	0bXXXXXXXX1
External reset	0bXXXXXXXX10
Watchdog reset	0bXXXXX1100
Software reset	0bXXX10100
Cleared	0b00000000

This is followed by a byte containing the current state and mode of the node. See Section B.1.6 for details on the node states. The byte may be:

Node Condition (State and Mode)	State/Mode Code	Service LED
Applicationless and Unconfigured	0x03	On
Unconfigured (but with an Application)	0x02	Flashing
Configured, Hard Offline	0x06	Off
Configured, Soft Offline	0x0C	Off
Configured, Bypass Offline	0x8C	Off
Configured, Online	0x04	Off

The state/mode byte has the following bit assignments:

State/Mode Byte							
B	x	x	x	M	S	S	S
S = state							
M = mode							
B = bypass							
x = not used							

The mode and bypass bits are meaningful only when the state bits reflect the configured state.

The next byte in the response message indicates the version number of the firmware executing on the target node. For the firmware distributed with LonBuilder 2.0, this is 2. The version number is followed by a byte indicating the reason for the last error detected by the firmware on the target node. A 0 means that no error has been detected since the last reset. For a description of the firmware errors, see the *Neuron C Programmer's Reference*, Chapter 11. LonBuilder and NodeBuilder tools support all versions of firmware from version 3 on, and also custom system images for Neuron 3150 Chips. Custom system images consist of a standard Neuron Chip firmware image combined with a custom system extension. They have version numbers in the range 128 – 254.

For custom system images (version numbers > 63), the version returned by the *Query Status* command can not be used to determine the original firmware version number since it reflects the version number assigned by the creator of the custom image. In order to obtain the original firmware version number on which the custom image was based, a single memory byte read of location 0x0000 must be performed over the network. This will return the original firmware version number. If the original firmware version reported is greater than 63, then the original firmware version can not be determined. Firmware version numbers 63 – 127 are reserved for use by Echelon. Any user developed custom system images should be assigned a firmware version number of 128 or greater.

The version number is followed by a byte indicating the reason for the last error detected by the firmware on the target node. This error number is stored in EEPROM. A 0 means that no error has been detected since the last reset. Errors 134, 135, 150, and 151 cause the node to reset itself. For a description of the firmware errors, see *Neuron C Reference Guide*, Chapter 11.

The last byte in the query status response is the Neuron Chip model number: 0 for a Neuron 3150 Chip, and 8 for a Neuron 3120 Chip.

Clear Status

This message clears the network error statistics accumulators and the error log. The request message contains no data.

Proxy Command (Request/Response Only)

This message requests the node to deliver a *Query ID*, *Query Status*, or *Query Transceiver Status* message to another node. This can be used when physical channel limitations prevent a message from the network management node from being received directly by the target tool (e.g., when the target node is out of reach of the network management node). The response is also relayed back to the original sender.

Message declaration:

```
typedef struct {
    enum {
        query_unconfigured      = 0,
        status_request          = 1,
        xcvr_status              = 2,
```

```

    } sub_command;
    unsigned type; // addr_type or (0x80 |group_size)
    unsigned : 1;
    unsigned member_or_node : 7;
    unsigned rpt_timer : 4;
    unsigned retry : 4;
    unsigned tx_timer : 4;
    unsigned group_or_subnet;
    unsigned neuron_id[ 6 ]; // for type = 2
} ND_proxy_request;

```

The request message contains a byte specifying which message is to be delivered by proxy, followed by a destination address in the format of a `msg_out_addr` declared in `\. . \INCLUDE\MSG_ADDR.H`. See Section A.3 for a description of destination address formats. The proxy message is delivered on the domain on which it was received. The address type field is 1 for `subnet_node`, 2 for `neuron_id`, 3 for broadcast, and for group addressing it contains the group size with the MSB set. The response message is the response appropriate to the specified status request. The node requesting the proxy service must take into account the response time through the agent when setting the transaction timer.

Query Transceiver Status (Request/Response Only)

This message retrieves transceiver status registers. This is a group of seven registers implemented in special-purpose mode transceivers. Even if the transceiver implements fewer than seven registers, seven values are returned in the response (see Section 4).

Message declaration from `netmgmt.h`:

```

typedef struct {
    unsigned xcvr_params[ NUM_COMM_PARAMS ];
} ND_query_xcvr_response;

```

The request message contains no data. The response message contains 7 bytes with the transceiver status register values. If the transceiver of the node receiving this message does not support status registers, the response will have the fail bit set.

B.3 NETWORK VARIABLE MESSAGES

Network variable messages are of two types — network variable updates and network variable polls. All network variable messages are implemented with message codes in the range 0x80 – 0xFF; i.e., the MSB of the code is set.

A network variable update message is sent whenever an output network variable is updated by the application program, and the variable has been declared without the *polled* qualifier. These messages may be sent with acknowledged, unacknowledged, or unacknowledged-repeated service class. Updating an output network variable that has been declared with the *polled* qualifier does not cause a network variable update to be sent. A network variable update message contains the selector of the network variable that was updated, along with the data value. When a network variable update message is addressed to a node that has an input network variable whose selector matches the network variable selector in the message, then a network variable update event occurs on the destination node, and the value of the input network variable is modified with the data in the message. For a node using a LONWORKS network interface, the comparison of selectors may be done in the Neuron Chip if host selection is disabled, or in the host processor if host selection is enabled.

A network variable poll message is sent when the application program calls the *poll()* system function specifying one or all of its input network variables. This message is sent with request/response service,

and contains the selector of the polled network variable. When a network variable poll message is addressed to a node which has an output network variable whose selector matches the network variable selector in the message, then that node responds with a message containing the data in the network variable. This response is treated the same as a network variable update message; a network variable update event occurs on the requesting node, and the value of the input network variable is modified with the data in the message.

Normally, network variable updates and polls are delivered using implicit addressing, namely using an entry in the address table of the source node as the destination address. However, for special applications, it is possible to explicitly address network variable updates and polls by sending explicit messages that have the same structure as network variable messages.

Network Variable Update (Acknowledged, Unacknowledged, or Unacknowledged-Repeated)

Normally, network variables are updated across the network using implicit addressing. When the application program on the source node updates the value of a bound output (non-polled) network variable, the Neuron Chip firmware automatically builds an outgoing network variable update message using the information from the network variable configuration table and the address table. The information in these two tables is created as a result of the binding process. In certain applications, it is desirable to explicitly address a network variable update, rather than have the address implicitly taken from the network variable configuration and address tables of the source node. For example, if a single node wishes to send network variable updates to more than 15 different destination addresses (single nodes or groups), then it can use explicit addressing to overcome the limit of 15 address table entries on a node.

In this case, the source node can create an explicit message that is functionally identical to a network variable update message (see Figure B-1). The code field of this message contains the most significant 6 bits of the network variable selector. The MSB of the code field is set, indicating a network variable message, and the second MSB is clear, indicating that the update is addressed to an input network variable. The first data byte of the message contains the least significant 8 bits of the network variable selector, and this is followed by the data in the network variable itself. The length of the data in the message must match the length of the destination network variable.

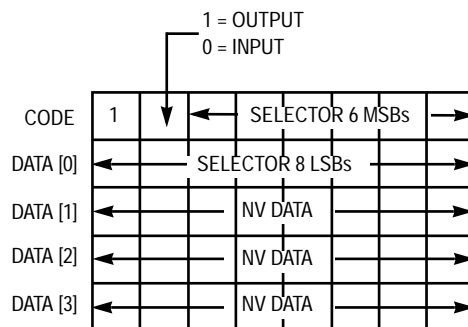


Figure B-1. Network Variable Message Structure

Example of updating a 2-byte network variable whose selector is 0x1234 with the data value 0x5678:

```
msg_out.code = 0x80 | 0x12;    // code field = 0x80 | nv_selector_hi
msg_out.data[ 0 ] = 0x34;      // data field = nv_selector_lo
msg_out.data[ 1 ] = 0x56;      // high byte of network variable value
msg_out.data[ 2 ] = 0x78;      // low byte of network variable value
```

Note that a network variable update message is processed by the network processor on the destination node. In a normal application program, the network variable update can not be received in the application

processor using explicit messaging syntax. If an application needs to extract the source address from an incoming network variable update message, the Neuron C `nv_in_addr` built-in variable can be used (see Section 7.2). Nodes using a LONWORKS network interface can optionally receive the network variable update on the host processor.

When a network variable update is addressed using group (multicast) addressing with acknowledged service, all members of the group acknowledge the update message. Those members of the group that have input network variables with a matching selector will update those variables as a result of receiving the message, and generate an `nv_update_occurs` application event. Those members of the group that have an output network variable with a matching selector, or no network variable with a matching selector, will not generate any application event, even though they acknowledge the update message. If all the acknowledgments are successfully received by the sending node, an `nv_update_succeeds` event is generated. If one or more acknowledgments are not received after the configured number of retries, an `nv_update_fails` event is generated.

Network Variable Poll (Request/Response Only)

Normally, network variables are polled across the network using implicit addressing. When the application program on the source node issues a `poll()` request to a bound input network variable, the Neuron Chip firmware automatically builds an outgoing network variable request message using the information from the network variable configuration table and the address table. The information in these two tables is created as a result of the binding process. In certain applications, it is desirable to explicitly address a network variable poll, rather than have the address implicitly taken from the network variable configuration and address tables of the source node. For example, if a single node wishes to poll network variables on more than 15 different destination nodes (single nodes or groups), then it can use explicit addressing to overcome the limit of 15 address table entries on a node. It can use a single input network variable to receive an unlimited number of responses to polls of any given data type.

In this case, the source node can create an explicit message that is functionally identical to a network variable poll message. The response to the poll will be processed by the network processor, and can not be received in the application program using explicit messaging syntax. If the node sending the poll message has an input network variable with the same selector and the same size as the polled network variable, then this network variable will be updated by the response to the poll. This will generate `nv_update_occurs` events. A more convenient method of reading the value of a network variable with an explicitly addressed message is with the *Network Variable Fetch* network management message described in Section B.1.4. In this case, the response is processed by the application processor with one of the request/response functions and not the network processor. Alternatively, a node using a LONWORKS network interface can handle the response to the poll explicitly on the host processor if the MIP is configured with network variable processing off.

The code field of the *Network Variable Poll* request message contains the most significant 6 bits of the network variable selector. The MSB of the code is set, indicating a network variable message, and the second MSB is set indicating that the poll is addressed to an output network variable. The first data byte of the request message contains the least significant 8 bits of the network variable selector. The response message contains the same code, except that the second MSB is clear, indicating that the response is addressed to an input network variable. The first data byte of the response contains the least significant 8 bits of the network variable selector, and this is followed by the data in the network variable itself. If the poll is received by a node that has no matching network variable, or the node is offline, then the response contains the selector, but no data is present.

When a network variable poll is addressed using group (multicast) addressing with acknowledged service, all members of the group acknowledge the poll request message. Those members of the group that have output network variables with a matching selector will respond with a message containing the value of the variable. These responses will generate `nv_update_occurs` events on the polling node. Those members

of the group that have an input network variable with a matching selector, or no network variable with a matching selector, or are offline, will generate a response containing no data. The generation of these responses requires the participation of the application processor in the polled node, and occurs at the end of the currently executing critical section. This should be taken into account when designing the application code for a node whose network variables may be polled, and when configuring the transaction timer for the poll message. If all the responses are successfully received by the polling node, an `nv_update_succeeds` event is generated. If one or more responses is not received after the configured number of retries, an `nv_update_fails` event is generated.

APPENDIX C

EXTERNAL MEMORY INTERFACING

For systems in which the supply current is limited to less than 300 mA or the reset line may be held low for extended periods of time, the following circuit is recommended.

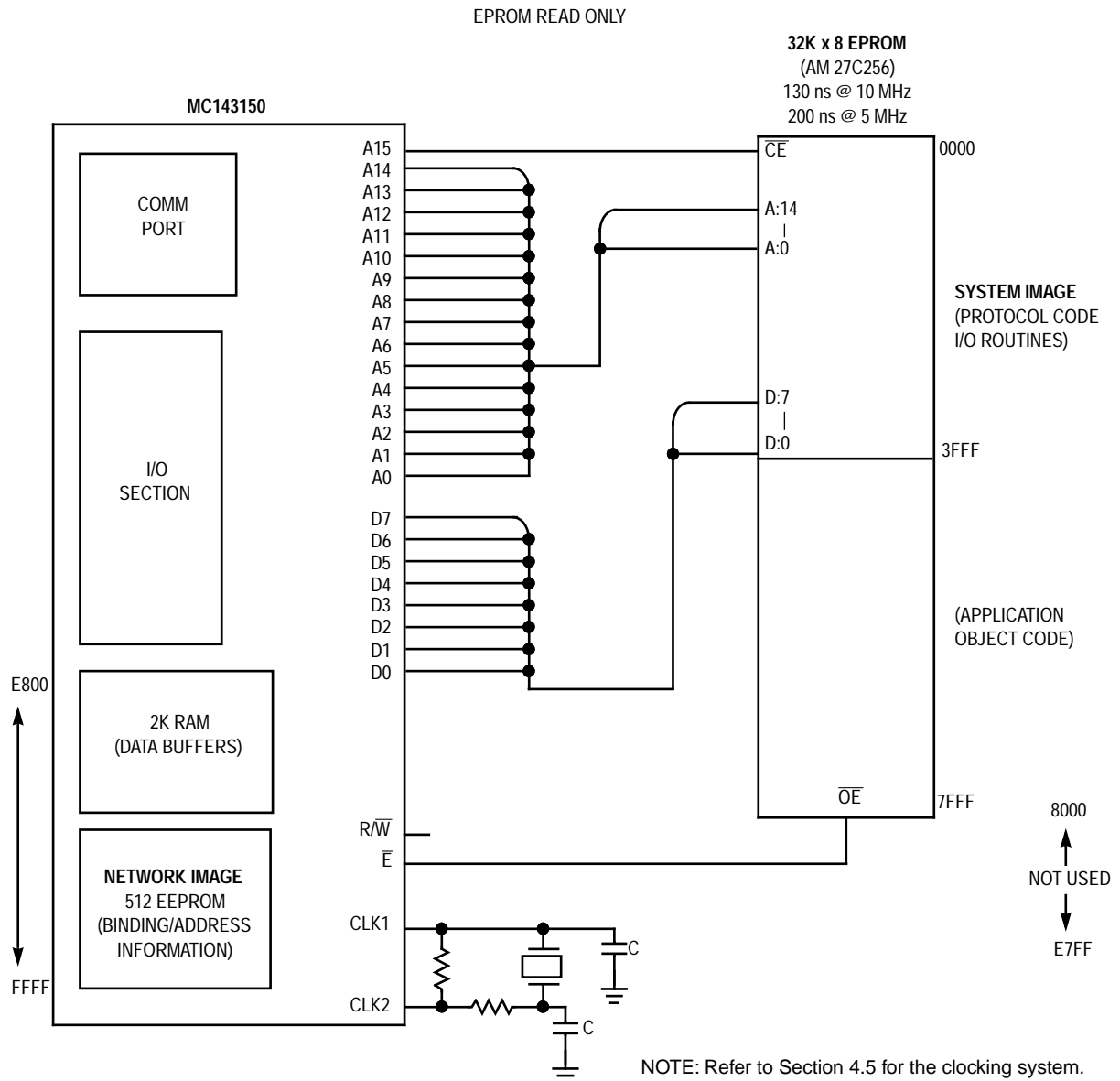


Figure C-1. EPROM Memory Interface

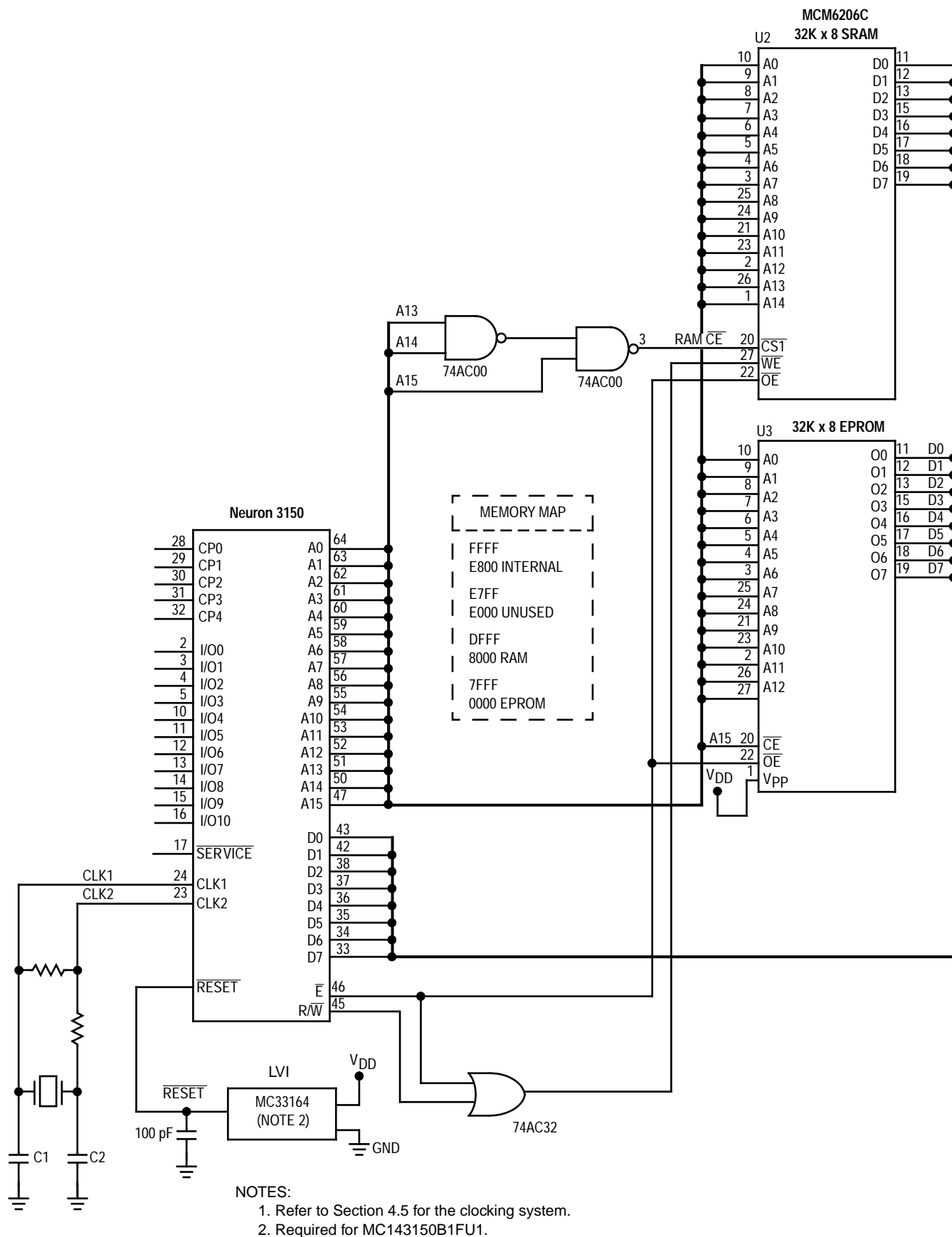


Figure C-2. MC143150B1FU1 External Memory Interface with 32 Kbyte EPROM and 24 Kbyte RAM

MEMORY INTERFACE ISSUES

The address lines A15 – A0 are driven to 0xFFFF during reset, thus disabling the external memories data output. This prevents databus contention between the external memory and the MC143150. For an MC143150 operating at 10 MHz, the available time is 95 ns ($1/2 \bar{E}$ clock cycle) – 20 ns (read data setup time), or 75 ns.

The MC143150 is designed not to drive D7 – D0 while \bar{E} is high. (Note: Data holds for 20 ns after \bar{E} clock is high.) Connecting \bar{E} to the \overline{OE} (active low) of the external memory will force the external memory to drive only when \bar{E} is low.

$$3150: \quad t = EB_{low} - t_{DSR} = 95 - 20 = 75 \text{ ns}$$

Memory Access Time

The access time requirements for an external memory interfacing to the Neuron is defined to be:

$$t_{acc} = t_{cyc} - t_{AD} - t_{DSR} - t_{decode}$$

After \bar{E} rises, the Neuron Chip outputs a new address. The external memory (and any associated board decode logic) must decode the new address and output data back to the Neuron Chip fast enough to meet data in the setup time requirements of the Neuron Chip (referenced back to the rising edge of \bar{E}). Note: Only t_{cyc} varies with frequency.

For MC143150:

$$t_{acc} = 200 - 50 - 20 = 130 \text{ ns} - t_{decode}$$

These devices allow the use of an external memory with a 120 ns access time, allowing for board decode logic and board delays.

Figure C-3. 32K Flash

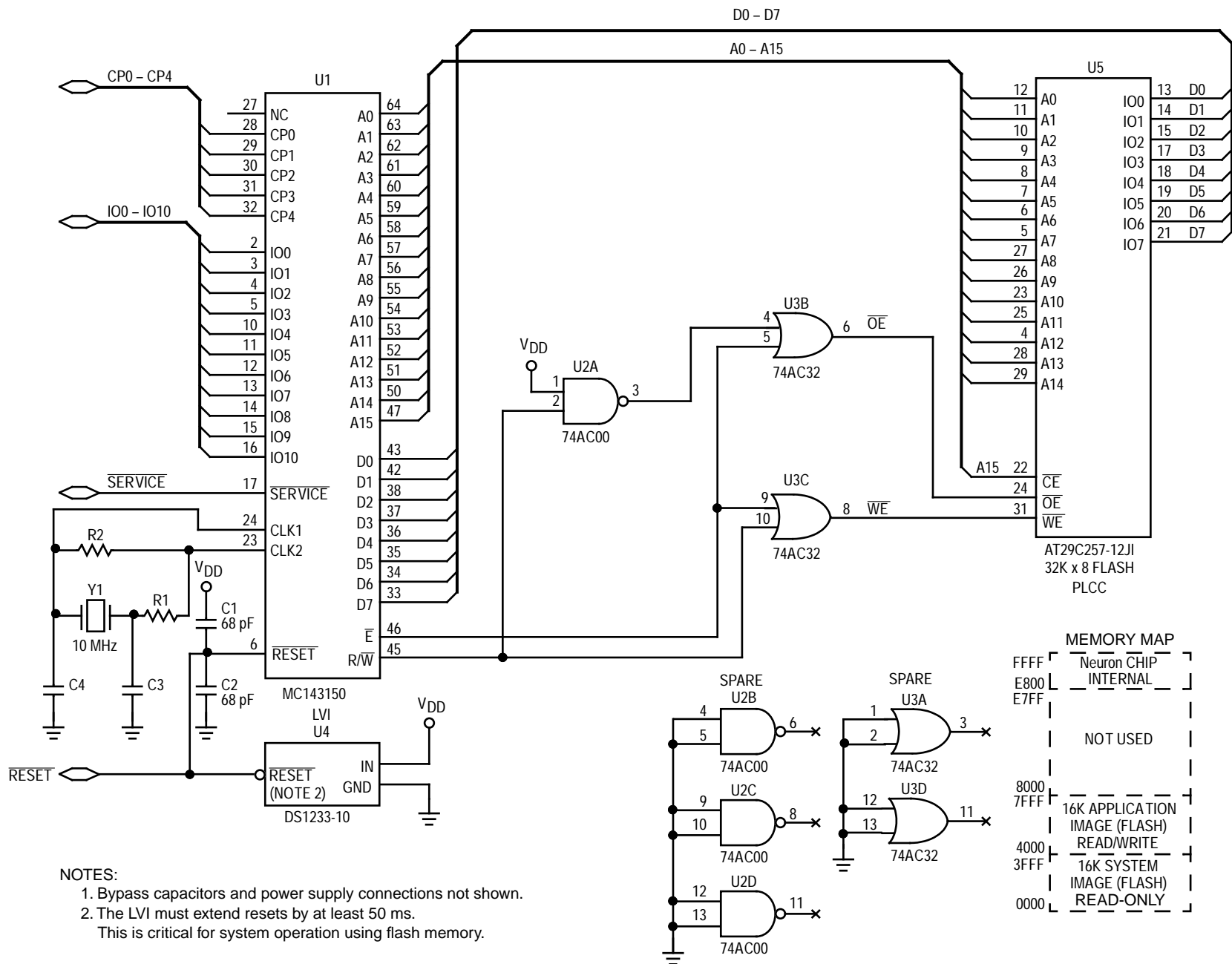
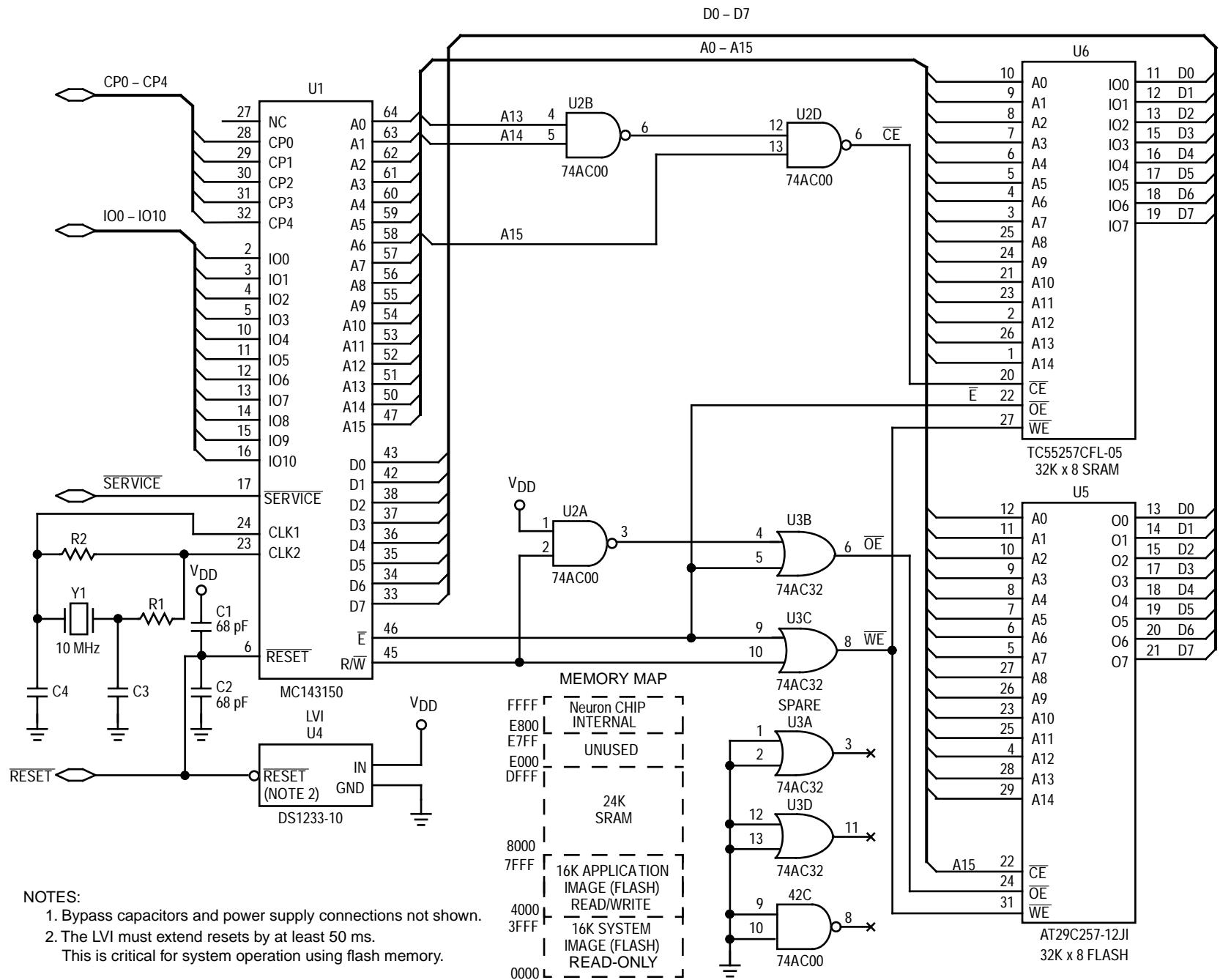


Figure C-4. 32K Flash/24K SRAM



SUITABLE MEMORY DEVICES

The following tables list vendors whose product lines include parts meeting the interface specifications of the Neuron 3150 Chip. This list does not cover all possible parts or semiconductor manufacturers. Motorola can not recommend one supplier over another. Contact the individual manufacturers or their representatives for price and availability.

PROM Devices (32K x 8)

Microchip	27HC256-12
Catalyst	CAT27HC256L-12
Wafer Scale Integration	WS27C256L-12J
Toshiba	TC57H256D-120
AMD	AM27C256-120JC
Atmel	AT27C256R-12JC

Static RAM Devices

Toshiba	TC55257-70L
---------	-------------

Flash Memory Devices

Atmel	AT29C256-90, AT29C256-12
	AT29C257-90, AT29C257-12
	AT29C512-90, AT29C512-12

APPENDIX D

DESIGN AND HANDLING GUIDELINES

D.1 APPLICATION CONSIDERATIONS

D.1.1 Termination of Unused Pins

Because the Neuron Chip is a CMOS device, unused input pins *including undeclared/unconnected I/O pins configured as inputs or three-state* must be terminated to assure proper operation and reliability. Figure D-1 shows a CMOS inverter representative of circuitry found on CMOS input pins. When the input is logic 0, the P-channel transistor is on (conducts), and the N-channel transistor is off. When the input is a logic 1, the P-channel transistor is off, and the N-channel transistor is on. These transistors are linear devices with relatively broad switch points. As the input transitions through the mid-supply region, there is a duration of time when both transistors are conducting. With fast rise time digital signals at the input, this duration is very short. Once the inverter is out of the linear region (it has switched high or low) there is very little current flow. This effect is the reason that the overall current drain of a CMOS device is directly proportional to the switching speed. Almost all the current consumption is by transistors passing through the linear region and charging and discharging of internal capacitances. If a pin is configured as an input or three-state, then the input can oscillate due to supply noise or float to the mid-supply region, resulting in higher current consumption. Current design techniques have made latch-up due to floating input unlikely, but it is good design practice to terminate unused I/O pins that are not configured (high impedance) or are configured as inputs. On the Neuron Chip, the only pin other than the I/O pins that could be configured as an unterminated input, is the $\overline{\text{SERVICE}}$ pin if the optional pull-up is disabled. If the optional pull-up devices are disabled on IO4 – IO7, then termination is necessary for those pins. Pull-ups are enabled in Neuron C with a pragma statement (`#pragma enable_io_pullups`).

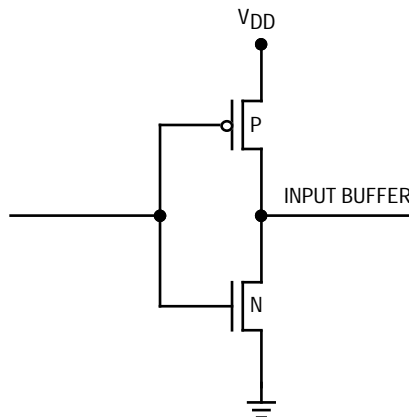


Figure D-1. CMOS Inverter

The best method to terminate unused I/O pins is with an individual pull-up or pull-down resistor for each unused pin. Unused input pins can be connected to each other and then to a common termination point. This cost/space effective method has the disadvantage of not allowing individual pin configuration later

and the possibility of contention in the event pins are declared as outputs. Individual unused I/O pins may be connected directly to V_{SS} or V_{DD} , but this is not recommended in case of software error and the possibility of output declaration to an opposing state. Unused pins may be declared as outputs, but this consumes application code space which is particularly valuable in 3120 applications. A pin capable of being configured as an output should never be connected to another such pin or directly to V_{SS} or V_{DD} .

D.1.2 Avoidance of Damaging Conditions

All integrated circuit devices can be damaged or destroyed by exceeding specified voltage and environmental limits. These limits are conservative to ensure reliable operation within the conditions specified.

Most potentially destructive ac waveforms fall into one of two categories. One type is high-voltage (10 kV – 25 kV), low-energy spikes usually under 100 ns in duration due to ESD discharge. ESD modeling has shown that the human body can generate and discharge electrostatic voltages of up to 12 kV. The second type is lower-voltage, higher-energy transients that can last for several hundred microseconds or more and can be caused by capacitive coupling of lightning or inductive load sources. Different protection devices must be implemented, depending on what is anticipated in the operating environment. Failure modes can be quantified and protective precautions taken to avoid product malfunction. This may be PC board layout-related or may involve the use of external protection devices. All pins on the Neuron Chip have internal diode protection that will protect ESD type transients up to 2 kV. External protection is required in products subject to human contact or where interfaces to other equipment may be encountered.

Many factors, including ambient temperature and semiconductor lot-to-lot processing variations, will influence the effect of illegal conditions on the Neuron Chip. The V_{SS} pins are internally connected to the substrate of the silicon die and are the reference point for all voltages. The Neuron Chip is guaranteed to function for a V_{DD} connected to the positive supply pin(s). In limited temperature range environments, the device may operate over a wider V_{DD} with timing, drive, and other specifications not met. It can survive with V_{DD} up to 7 V without damage, but timing and drive levels will differ from those specified. There may also be some adverse effects on gate oxides from long-term exposures to V_{DD} equal to or greater than 7 V.

Zap and latch-up refer to two damage mechanisms resident in CMOS ICs. Zap refers to damage caused by very-high-voltage, static-electricity exposure. This damage usually appears as breakdown of the relatively thin oxide layers that causes leakage or shorts. Often secondary damage occurs after an initial zap failure causes a short.

Latch-up refers to a usually catastrophic condition caused by turning on a parasitic, bipolar, silicon-controlled rectifier (SCR). A latch-up is formed by N and P regions in the layout of the integrated circuit, which act as the collector, base, and emitter of parasitic transistors. Bulk resistance of silicon in the wells and substrate acts as resistors in the SCR circuit. Application of voltages to pins above $V_{DD} + 0.3$ V or below $V_{SS} - 0.3$ V in conjunction with enough current to develop voltage drops across the parasitic resistors can cause the SCR to turn on. Once on, the SCR can be turned off only by removal of all power and applied voltages. The low on impedance of the SCR circuit can overheat and destroy the IC.

Figure D-2 shows the MOS circuitry for a digital input-only pin. The gates of the input buffer are very high impedance for all voltages that would ever be applied to the pin. Protection is implemented with a P-channel transistor acting as a diode to V_{DD} and an N-channel transistor acting as a diode to V_{SS} . Allowing a pin to float or be driven to a mid-supply level can result in both the N- and P-channel devices in the input buffer simultaneously being partially on, which causes excess current and noise on the V_{DD}/V_{SS} power supply. If a digital input is driven above V_{DD} , the pseudo-diode will conduct, protecting the input. As the current is increased to high levels (100 mA), damage can result. Figure D-3 shows the CMOS circuitry for a digital input/output-only pin. Pins CP0 – CP3 are designed so that if power is removed from the Neuron Chip, they will not load down the network.

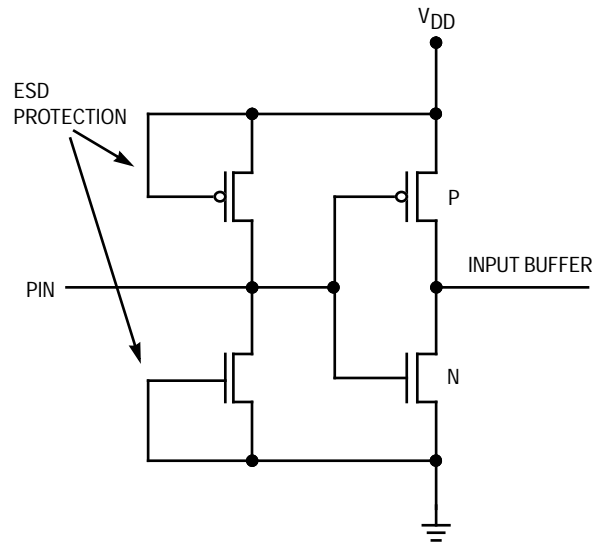


Figure D-2. Digital Input

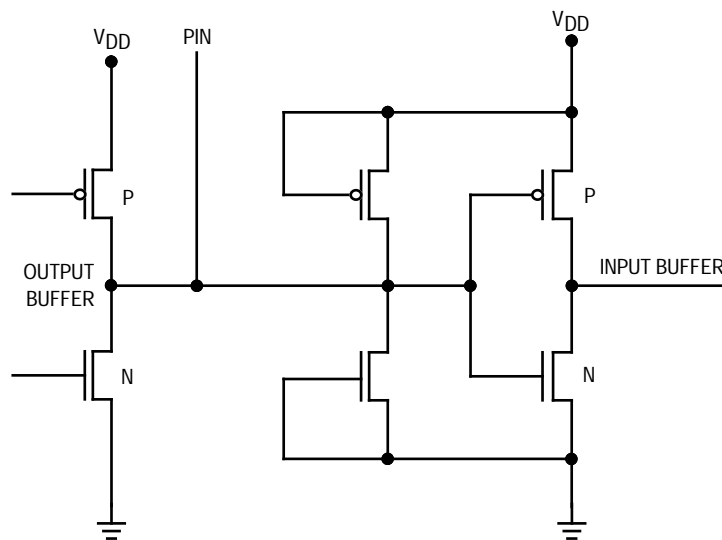


Figure D-3. Digital I/O

D.1.3 Power Supply, Ground, and Noise Considerations

This device may be used in digital equipment applications which require plugging the PC board into a rack with power applied. This is referred to as “hot-rack insertion.” In these applications, care should be taken to limit the voltage on any pin from going positive relative to the V_{DD} pins, or negative relative to the V_{SS} pins. One method to accomplish this is to extend the ground and power contacts of the PCB connector so that power is applied prior to any other pins having voltage applied. The device has input protection on all pins and may source or sink a limited amount of current without damage. See Section 6.2.1, Absolute Maximum Ratings, for more information concerning the current into or out of the device pins. Current limiting may be accomplished by series resistors between the signal pins and the connector contacts.

The most important considerations for PCB layout deal with noise. This includes noise on the power supply, noise generated by the digital circuitry on the device, and coupling digital signals into the analog signals. The best PCB layout methods to prevent noise induced problems are:

1. Keep digital signals as far away from analog signals as possible.

2. Use short, low-inductance traces for the analog circuitry to reduce inductive, capacitive, and radio frequency noise sensitivities.
3. Use short, low-inductance traces for the digital circuitry to reduce inductive, capacitive, and radio frequency radiated noise.
4. Bypass capacitors should be connected between the V_{DD} and V_{SS} pairs with minimal trace length. These capacitors help supply the instantaneous currents of the digital circuitry in addition to decoupling the noise that may be generated by other sections of the device or other circuitry on the power supply.
5. Use short, wide, low-inductance traces to connect all of the V_{SS} ground pins together and, with one trace, connect all of the V_{SS} ground pins to the power supply ground. Depending on the application, a double-sided PCB with a V_{SS} ground plane under the device connecting all of the digital and analog V_{SS} pins together would be a good grounding method. A multilayer PCB with a ground plane connecting all of the digital and analog V_{SS} pins together would be the optimal ground configuration. These methods will result in the lowest resistance and the lowest inductance in the ground circuit. This is important to reduce voltage spikes in the ground circuit resulting from the high-speed digital current spikes. Suppressing these voltage spikes on the integrated circuit is the reason for multiple V_{SS} ground leads.
6. Use short, wide, low-inductance traces to connect all of the V_{DD} power supply pins together and, with one trace, connect all of the V_{DD} power supply pins to the 5 V power supply. Depending on the application, a double-sided PCB with V_{DD} bypass capacitors to the V_{SS} ground plane under the device may complete the low-impedance coupling for the power supply. For a multilayer PCB with a power plane, connecting all of the digital and analog V_{DD} pins to the power plane would be the optimal power distribution method. The integrated circuit layout and packaging considerations for the 5 V V_{DD} power circuit are essentially the same as for the ground circuit.
7. Motorola recommends that a 4-layer board be used. It is possible to use a 2-layer board, but special care must be taken.

D.1.4 Transmission Line Termination

When data is transmitted over long distances, the line on which the data travels can be considered a transmission line. (Long distance is relative to the data rate being transmitted.) Examples of transmission lines include high-speed buses, long PCB lines, and coaxial and ribbon cables. All transmission lines should be properly terminated into a low-impedance termination. A low-impedance termination helps eliminate noise, ringing, overshoot, and crosstalk problems. Also, a low-impedance termination reduces signal degradation because the small values of parasitic line capacitance and inductance have lesser effect on a low-impedance line.

The value of the termination resistor becomes a trade-off between power consumption, data rate speeds, and transmission line distance. The lower the resistor value, the faster data can be presented to the receiving device, but the more power the resistor consumes. The higher the resistor value, the longer it will take to charge and discharge the transmission line through the termination resistor ($T = R \cdot C$).

Transmission line distance becomes more critical as data rates increase. As data rates increase, incident (and reflective) waves begin to resemble that of RF transmission line theory. However, due to the nonlinearity of CMOS digital logic, conventional RF transmission theory is not applicable.

By increasing the termination resistance value, the CMOS advantage of low-power consumption can be realized. Motorola recommends a minimum termination resistor value as shown in Figure D-4. The termination resistor should be as close to the receiving unit as possible. Another method of terminating the line driver, as well as the receiving unit, is shown in Figure D-5. Note that the resistor values in Figure D-5 are twice the resistor value of Figure D-4; this gives a net equivalent termination value of Figure D-4. Even higher values of resistors may be used for either termination method. This reduces power consumption, but at the expense of speed and possible signal degradation.

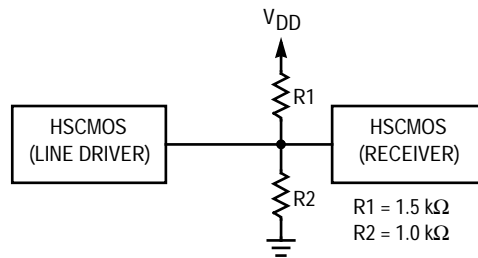


Figure D-4. Termination Resistors at the Receiver

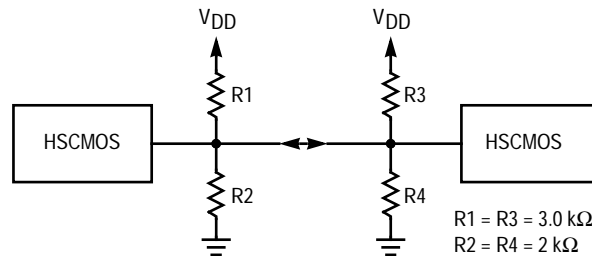


Figure D-5. Termination Resistors at Both the Line Driver and Receiver

D.1.5 Decoupling Capacitors

The switching waveforms shown in Figures D-6 and D-7 show the current spikes introduced to the power supply and ground lines. This effect is shown for a load capacitance of less than 5 pF and for 50 pF. For ideal power supply lines with no series impedance, the spikes would pose no problem. However, actual power supply and ground lines do possess series impedance, giving rise to noise problems. For this reason, care should be taken in board layouts, ensuring low-impedance paths to and from logic devices.

To absorb switching spikes, the HSCMOS devices should be bypassed with good quality 0.022 μF to 0.33 μF decoupling capacitors:

1. Bypass every device driving a bus with all outputs switching simultaneously.
2. Bypass all synchronous counters.
3. Bypass devices used as oscillator elements.
4. Bypass Schmitt-trigger devices with slow input rise and fall times. The slower the rise and fall time, the larger the bypass capacitor. Lab experimentation is suggested.

Bypass capacitors should be distributed over the circuit board. In addition, boards could be decoupled with a 1 μF capacitor.

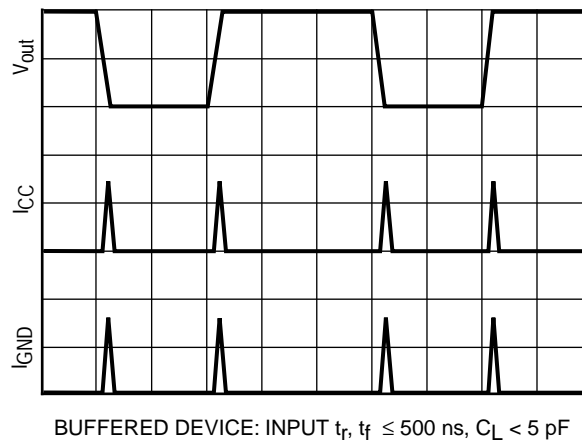


Figure D-6. Switching Currents for $C_L < 5 \text{ pF}$

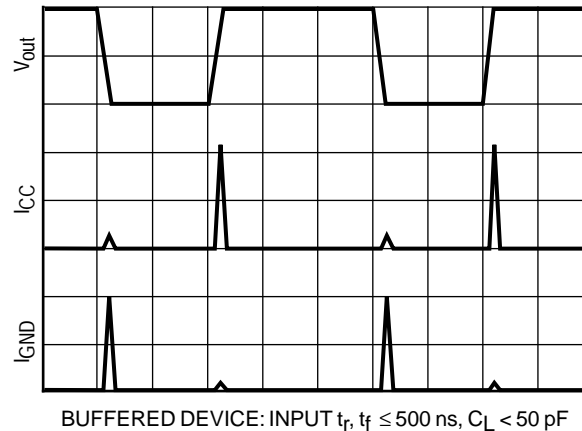


Figure D-7. Switching Currents for $C_L = 50 \text{ pF}$

D.2 BOARD SOLDERING CONSIDERATIONS

Dry pack is a process which slowly bakes moisture from the surface mount technology (SMT) package and then seals it into a dry pack bag to shield the unit from moisture in the atmosphere. The exterior of the bag will be marked with a label that indicates the devices are moisture sensitive and is marked with the date the bag was sealed (there is a one-year shelf life for these devices). There is a limited amount of time to use surface-mount devices once they are removed from the dry pack. It is recommended that before surface mounting, packages should not be out of the dry pack longer than 168 hours at $< 60\%$ relative humidity and $< 30^\circ\text{C}$. If the units have not been shipped dry pack or have been unpacked for too long, then units must be baked at 125°C for 6 hours prior to board soldering. (The old recommendation was 24 hours at 125°C .) If this is not done, some percentage of the units will exhibit destructive failures or latent failures after the soldering process.

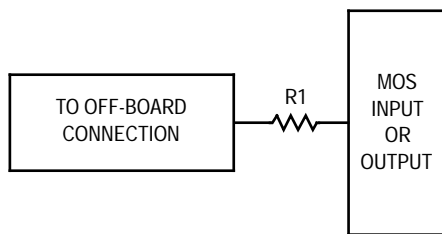
D.3 HANDLING PRECAUTIONS AND ELECTROSTATIC DISCHARGE

All CMOS devices have an insulated gate that is subject to voltage breakdown. The gate oxide for the Neuron Chip breaks down at a gate-source potential of about 10 V. The high-impedance gates on the devices are protected by on-chip networks. However, these on-chip networks do not make the IC immune to ESD. Laboratory tests show that devices may fail after one very high voltage discharge. They may also fail due to the cumulative effect of several discharges of lower potential.

Static-damaged devices behave in various ways, depending on the severity of the damage. The most severely damaged are the easiest to detect because the input or output has been completely destroyed and is either shorted to V_{DD} , shorted to V_{SS} , or is open-circuited. The effect is that the device no longer functions. Less severe cases are more difficult to detect because they appear as intermittent failures or degraded performance. Static damage can often increase leakage currents.

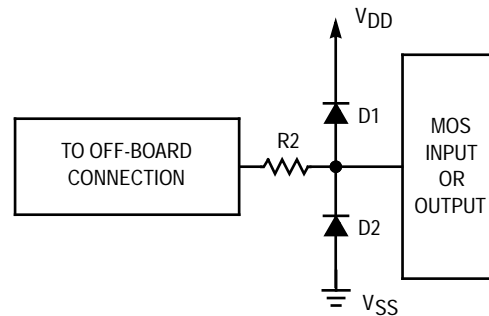
CMOS devices are not immune to large static voltage discharges that can be generated while handling. For example, static voltages generated by a person walking across a waxed floor have been measured in the 4 kV – 15 kV range (depending on humidity, surface conditions, etc.). Therefore, the following precautions should be observed.

1. Do not exceed the maximum ratings specified by the data sheet.
2. All unused device inputs should be connected to V_{DD} or V_{SS} .
3. All low-impedance equipment (pulse generators, etc.) should be connected to CMOS inputs only after the device is powered up. Similarly, this type of equipment should be disconnected before power is turned off.
4. A circuit board containing CMOS or devices is merely an extension of the device and the same handling precautions apply. Contacting connectors wired directly to devices can cause damage. Plastic wrapping should be avoided. When external connections to a PC board address pins of CMOS integrated circuits, a resistor should be used in series with the inputs or outputs. The limiting factor for the series resistor is the added delay caused by the time constant formed by the series resistor and input capacitance. This resistor will help limit accidental damage if the PC board is removed and is brought into contact with static-generating materials. For convenience, equations for added propagation delay and rise-time effects due to series-resistance size are given in Figure D-8.
5. All CMOS devices should be stored or transported in materials that are antistatic. Devices must not be inserted into conventional plastic “snow,” Styrofoam[®], or plastic trays, but should be left in their original container until ready for use.
6. All CMOS devices should be placed on a grounded bench surface and operators should ground themselves prior to handling devices, since a worker can be statically charged with respect to the bench surface. Wrist straps in contact with skin are strongly recommended. See Figure D-9.
7. Nylon or other static-generating materials should not come in contact with CMOS circuits.
8. If automatic handling is being used, high levels of static electricity may be generated by the movement of devices, belts, or boards. Reduce static build-up by using ionized air blowers or room humidifiers. All parts of machines which come into contact with the top, bottom, and sides of IC packages must be grounded metal or other conductive material.
9. Cold chambers using CO₂ for cooling should be equipped with baffles, and devices must be contained on or in conductive material.
10. When lead-straightening or hand-soldering is necessary, provide ground straps for the apparatus used and be sure that soldering ties are grounded.
11. The following steps should be observed during wave-solder operations.
 - a. The solder pot and conductive conveyor system of the wave-soldering machine must be grounded to an earth ground.
 - b. The loading and unloading work benches should have conductive tops which are grounded to an earth ground.
 - c. Operators must comply with precautions previously explained.
 - d. Completed assemblies should be placed in antistatic containers prior to being moved to subsequent stations.



Advantage: Requires minimal board area.

Disadvantage: $R1 > R2$ for the same level of protection, therefore rise and fall times, propagation delays, and output drives are severely affected.



Advantage: $R2 < R1$ for the same level of protection. Impact on ac and dc characteristics is minimized.

Disadvantage: More board area, higher initial cost.

NOTE: These networks are useful for protecting the following:

- a. digital inputs and outputs
- b. analog inputs and outputs
- c. three-state outputs
- d. bidirectional (I/O) ports

Equation 1 – Propagation Delay vs Series Resistance

$$R \approx \frac{t}{C \cdot k}$$

where:

- R = the maximum allowable series resistance in ohms
- t = the maximum tolerable propagation delay in seconds
- C = the board capacitance plus the driven device's input capacitance in farads
- k = 0.33 for the TTL input levels (switch point = 1.3 V)

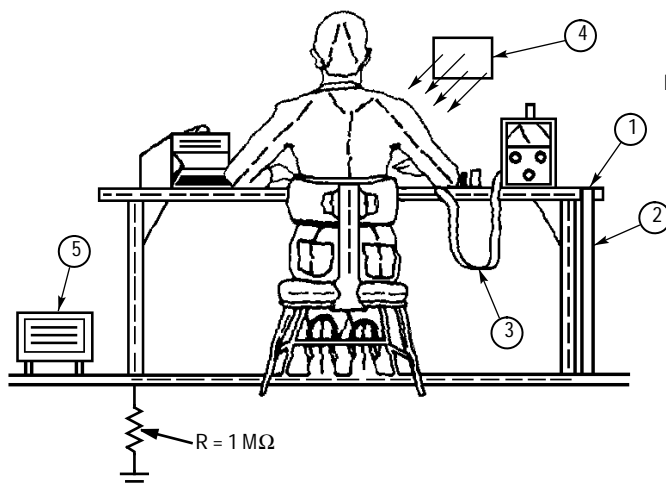
Equation 2 – Rise Time vs Series Resistance

$$R \approx \frac{t}{C \cdot k}$$

where:

- R = the maximum allowable series resistance in ohms
- t = the maximum rise time per data sheet in seconds
- C = the board capacitance plus the driven device's input capacitance in farads
- k = 2.3 for other devices

Figure D-8. Networks for Minimizing ESD and Reducing CMOS Latch-Up Susceptibility



NOTES:

- 1. 1/16 inch conductive sheet stock covering bench-top work area.
- 2. Ground strap.
- 3. Wrist strap in contact with skin.
- 4. Static neutralizer. (Ionized air blower directed at work.) Primarily for use in areas where direct grounding is impractical.
- 5. Room humidifier. Primarily for use in areas where the relative humidity is less than 45%. Caution: building heating and cooling systems usually dry the air causing the relative humidity inside a building to be less than outside humidity.

Figure D-9. Typical Manufacturing Work Station

12. The following steps should be observed during board cleaning operation.
 - a. Vapor degreasers and baskets must be grounded to an earth ground. Operators must likewise be grounded.
 - b. Brush or spray cleaning should not be used.
 - c. Assemblies should be placed into the vapor degreaser immediately upon removal from the antistatic container.
 - d. Cleaned assemblies should be placed in antistatic containers immediately after removal from the cleaning basket.
 - e. High-velocity air movement or application of solvents and coatings should be employed only when module circuits are grounded and a static eliminator is directed at the module.
13. The use of static-detection meters for line surveillance is highly recommended.
14. Equipment specifications should alert users to the presence of CMOS devices and require familiarization with this specification prior to performing any kind of maintenance or replacement of devices or modules.
15. Do not insert or remove CMOS devices from test sockets with power applied. Check all power supplies to be used for testing devices to be certain there are no voltage transients present.
16. Double-check the equipment setup for proper polarity of voltage before conducting parametric or functional testing.
17. Do not recycle shipping rails. Continuous use causes deterioration of their antistatic coating.
18. Wrist straps and equipment logs should be maintained and audited on a regular basis. Wrist straps malfunction and may go unnoticed. Also, equipment gets moved from time to time and grounds may not be reconnected properly.

ELECTROSTATIC DISCHARGE

There are many ways to deal with ESD, including:

- Divert or limit energy from points of contact to circuitry.
- Start with a series of electromagnetic interference (EMI) ferrites or resistors for high frequency filtering.
- Use diodes, transient voltage suppressors (ex: MOSorbs, transorbs, ...) for high-speed clamping.
- Use capacitors to protect critical inputs.
- Use good power distribution.
- Use a separate, low-impedance ESD ground path to divert energy from electronics (ex: "star" ground strategy).

NOTE: The impedance of a wire at 300 MHz is approximately 20 Ω /cm.

Use a conductor with less than or equal to 3:1 length:width ratio.

RECOMMENDED READING

Total Control of the Static in Your Business

Available by writing to:

Static Control Systems Div.
Box ELB-3, 225-4S
3M Center
St. Paul, MN 55144

Or calling:

1-800-328-1368
1-612-733-9420 (in Minnesota)

D.4 REDUCTION OF ELECTROMAGNETIC INTERFERENCE

Electromagnetic interference and radio frequency interference (RFI) are phenomena inherent in all electrical systems covering the entire frequency spectrum. Although the characteristics have been well documented, EMI remains difficult to deal with due to numerous variables. EMI should be considered at the beginning of a design, and taken into account during all stages, including production and beyond.

These entities must be present for EMI to be a factor: (1) a source of EMI, (2) a transmission medium for EMI, and (3) a receiver of EMI. Several sources include relays, FM transmitters, local oscillators in receivers, power lines, engine ignitions, arc welders, and lighting. EMI transmission paths include ground connections, cables, and the space between conductors. Some receivers of EMI are radar receivers, computers, and television receivers.

For microprocessor-based equipment, the source of emissions is usually a current loop on a PC board. The chips and their associated loop areas also function as receivers of EMI. The fact is that PC boards which radiate high levels of EMI are also more likely to act as receivers of EMI.

All logic gates are potential transmitters and receivers of emissions. Noise immunity and noise margin are two criterion which measure a gate's immunity to noise which could be caused by EMI. CMOS technology, as opposed to the other commonly used logic families, offers the best value for noise margin and is therefore, an excellent choice when considering EMI.

The electric and magnetic fields associated with ICs are proportional to the current used, the current loop area, and the switching transition times. CMOS technology is preferred due to smaller currents. Also, the current loop area can be reduced by the use of surface-mount packages.

In a system where several pieces of equipment are connected by cables, at least five coupling paths should be taken into account to reduce EMI. They are: (1) common ground impedance coupling (a common impedance is shared between an EMI source and receiver), (2) common-mode, field-to-cable coupling (electromagnetic fields enter the loop formed by two pieces of equipment, the cable connecting them, and the ground plane), (3) differential-mode, field-to-cable coupling (electromagnetic fields enter the loop formed by two pieces of equipment and the cable connecting them), (4) crosstalk coupling (signals in one transmission line are coupled into another transmission line), and (5) a conductive path through power lines.

Shielding is a means of reducing EMI. Some of the more commonly used shields against EMI and RFI contain stainless steel fiber-filled polycarbonate, aluminum flake-filled polycarbonate/ABS coated with nickel and copper electrolysis plating or cathode sputtering, nickel coated graphite fiber, and polyester SMC with carbon-fiber veil. Several manufacturers who make conductive compounds and additives are listed below.

SHIELDING MANUFACTURERS

General Electric Co., Plastics Group, Pittsfield, MA
Mobay Chemical Corp., Pittsburgh, PA
Wilson-Fiberfil International, Evansville, IN
American Cyanamid Co., Wayne, NJ
Fillite U.S.A., Inc., Huntington, WV
Transnet Corp., Columbus, OH

Motorola does not recommend, or in any way warrant, the manufacturers listed here. Additionally, no claim is made that this list is by any means complete.

RECOMMENDED READING

- D. White, K. Atkinson, and J. Osburn, "Taming EMI in Microprocessor Systems", *IEEE Spectrum*, Vol. 22, Number 12, Dec. 1985.
- D. White and M. Mardiguian, *EMI Control Methodology and Procedures*, 1985.
- H. Denny, *Grounding for the Control of EMI*.
- M. Mardiguian, *How to Control Electrical Noise*.
- D. White, *Shielding Design Methodology and Procedures*.

For more information on this subject, contact:

Interference Control Technologies
Don White Consultants, Inc., Subsidiary
State Route 625
P.O. Box D
Gainesville, VA 22065

D.5 HARDWARE DESIGN

INTRODUCTION

In the ideal digital world, signal voltages and timing margins are well within specifications. The environment is benign.

However, in the real world:

- Power distribution is imperfect; switching transients create noise and corrupt signal margins.
- Impedance discontinuities distort signals, corrupting signal margins.
- Radiated or conducted noise from external sources corrupts signal margins.
- Internal noise radiates or conducts to the outside world, creating electromagnetic interference.
- Electrostatic discharge and lightning surges interrupt or destroy systems.
- Supply voltage drops below operating range due to power interruption, resulting in logic malfunction.
- Thermal stress, mechanical stress, and component aging cause failures.

Many potential hardware problems may be averted by planning carefully. This section covers the following topics:

- Power Distribution
- EMI
- ESD
- Latch-Up
- Power Interruptions
- Testing
- Board Layout Issues and Guidelines

D.5.1 Power Distribution

Inductance in the power distribution creates noise during switching transients.

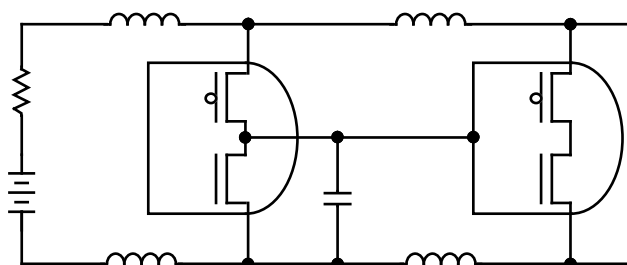


Figure D-10. Inductance Creates Noise

For example, a 200 nH inductor with 25 mA and 5 ns surge characteristic can generate 1 V in noise:

$$V_{\text{noise}} = L \, di/dt$$

$$L = 200 \, \text{nH}, \, di = 25 \, \text{mA}, \, dt = 5 \, \text{ns}$$

$$\Rightarrow V_{\text{noise}} = 200 \, \text{nH} * 25 \, \text{mA} / 5 \, \text{ns} = 1 \, \text{V}$$

Figure D-11 shows what the voltage looks like on an MC143150 Neuron Chip across V_{DD} and V_{SS} , with and without bypass capacitors. The MC143150 is on a two-layer board running at 5 MHz. The power cable is 0.5 m long. The 2.2 μF bypass capacitor brought the noise down from 1.1 V (using no bypass capacitor) to 0.4 V.

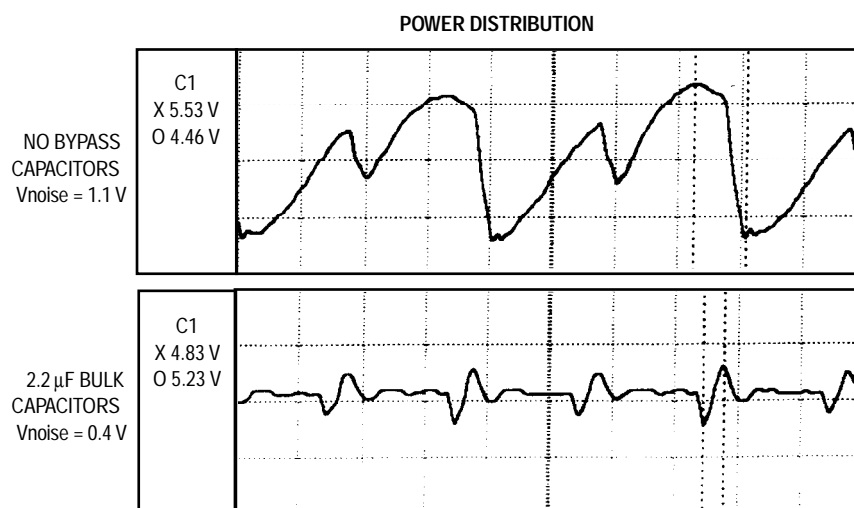


Figure D-11. Noise With and Without Bypass Capacitors

Figure D-12 shows a similar test, this time comparing a 2.2 μF bulk + 0.1 μF disc capacitors to a 2.2 μF bulk + 0.1 μF surface-mount bypass capacitor. The later case brought the noise down from 0.18 V to 0.06 V.

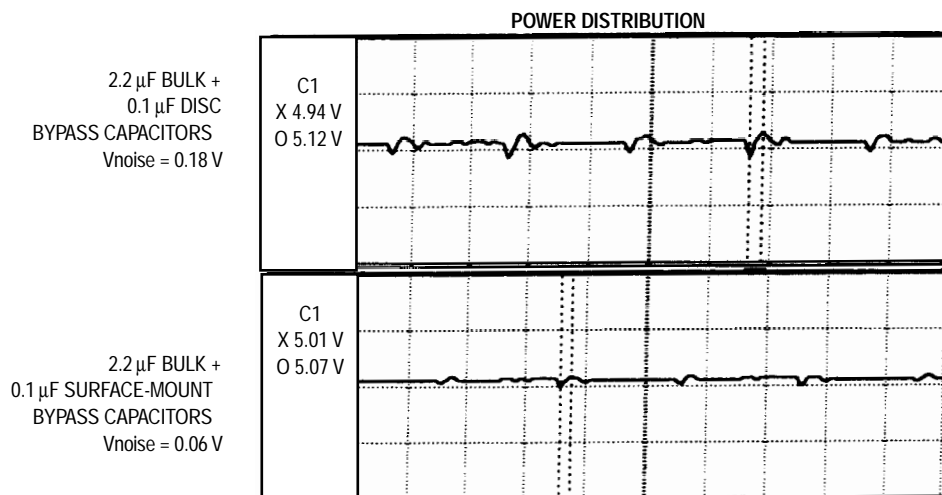


Figure D-12. Reducing Noise by Adding Additional Capacitors

Following are calculations to determine what size bulk capacitor to use:

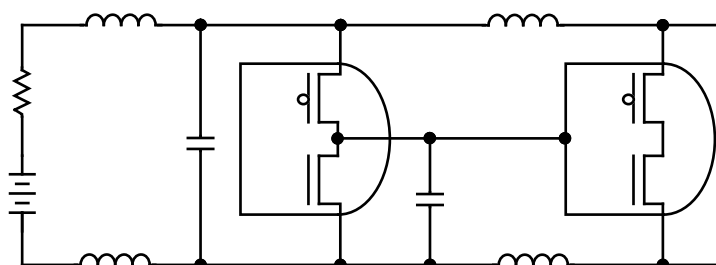


Figure D-13. Choosing a Bulk Capacitor

Set $X_{C_{bulk}} = X_{L_{pwr}} = \Delta V / \Delta I$ at common frequency ($X_{C_{bulk}} = 1/2 \pi f C$, $X_{L_{pwr}} = 1/2 \pi f L$)

$$\Rightarrow C_{bulk} = L_{pwr} * (\Delta I / \Delta V)^2$$

and for power distribution on a wire pair,

$$L_{pwr} = 400 * \text{wire length in meters} * \ln(2 * \text{wire separation} / \text{wire diameter})$$

Example:

- Power-supply wiring is 0.5 m long, 2.1 mm separation, 1.3 mm diameter

$$\Rightarrow L_{pwr} = 400 * 0.5 * \ln(2 * 2.1 / 1.3) = 234 \text{ nH}$$

- 50 outputs switching 10 pF through 5 V in 5 ns

$$\Rightarrow \Delta I = 50 * C * dV/dt = 50 * 10 \text{ pF} * 5 \text{ V} / 5 \text{ ns} = 0.5 \text{ A}$$

- Noise budget for bypassing is 0.1 V

$$\Rightarrow \Delta V = 0.1 \text{ V}$$

$$\Rightarrow C_{bulk} = 234 \text{ nH} * (0.5 \text{ A} / 0.1 \text{ V})^2 = 5.8 \text{ } \mu\text{F}$$

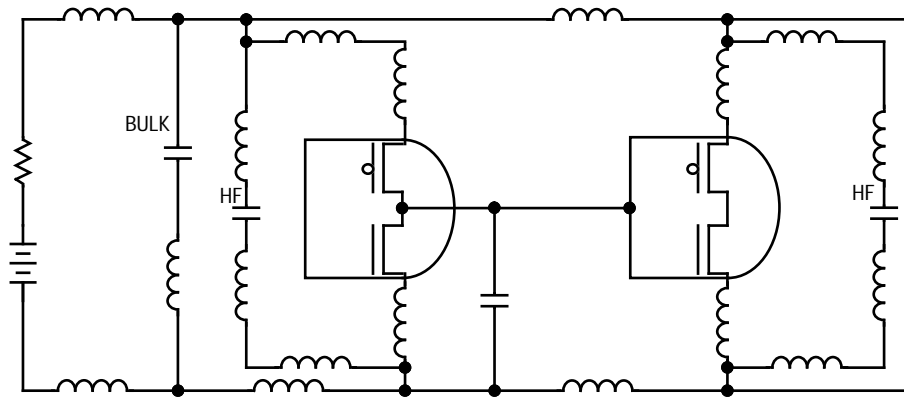


Figure D-14. Choosing a High-Frequency Bypass Capacitor

The IC lead frame, board trace (including vias), and capacitor leads all contribute to inductance.

$L_{\text{lead}} \approx 3 \text{ nH to } 5 \text{ nH per pin}$

$L_{\text{trace}} \approx 4 \text{ nH per cm for a trace above a ground plane (where } h/d = 1)$

$L_{\text{trace}} \approx 6 \text{ nH per cm for a trace with return on opposite side of board}$

$L_{\text{trace}} \approx 10 \text{ nH per cm for a trace with no nearby return}$

$L_{\text{cap}} \approx 1 \text{ nH for surface-mount (smt) capacitor}$

$L_{\text{cap}} \approx 5 \text{ nH for typical disk capacitor}$

Example 1: $L = 3 * 2 \text{ pins} + 1.5 * 2 \text{ traces} + 1 * 1 \text{ smt cap} = 10 \text{ nH}$

Example 2: $L = 5 * 2 \text{ pins} + 20 * 2 \text{ traces} + 5 * 1 \text{ smt cap} = 55 \text{ nH}$

D.5.2 EMI

Some guidelines to effectively control radiated EMI and susceptibility are:

- Start with an effective power-distribution scheme.
- Minimize capacitive coupling of noisy/power paths to external transceiver and I/O lines.
- Minimize loop area of any signal/power path which has high-frequency or high-current content.

For a small loop of area (A), carrying current (I) of frequency (f), the magnitude of the electric field (E) at distance (R) is:

$$E \propto f^2 * A * I * (1/R) (\text{far field, free space, } \phi = 90^\circ)$$

Figure D-15 depicts a larger loop area compared to a much smaller loop area. In affect, a larger loop area radiates more EMI than a smaller loop area.

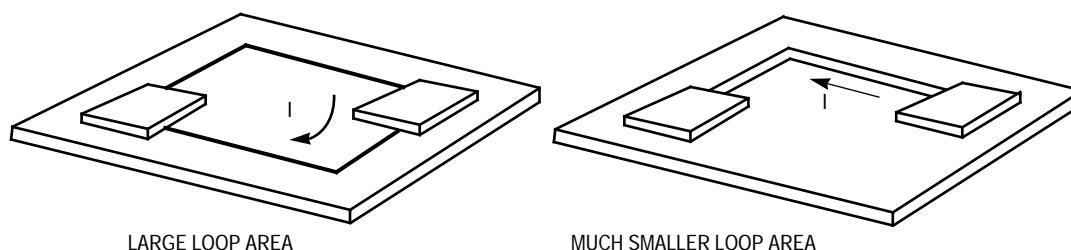


Figure D-15. Radiated EMI of a Large Loop Area versus a Smaller Loop Area

To deal with EMI:

- Separate or guard Neuron Chip/memory/clock traces from transceiver and external I/O connections.
- Consider filtering or buffering external I/O connections.
- Use high-frequency V_{DD} and V_{SS} bypassing with small surface-mount capacitors close to IC pins and at board power input.
- Use parallel V_{DD} and V_{SS} traces on 2-layer boards.
- Use parallel signal and return traces on 2-layer boards, especially for traces carrying substantial current or having a fast rise time.
- Consider a 2-layer board, especially if board space is limited or the input clock is 10 MHz.
- Dedicate 2 planes of a 4-layer board exclusively for V_{DD} and V_{SS} (do not use for signal routing unless planar integrity can be restored).
- Ensure integrity of V_{DD} and V_{SS} planes across through-hole arrays of connectors, ICs, vias, etc.

Refer to Appendix H for an example of laying out the crystal circuit for the 3150 and 3120 Neuron Chips.

D.5.3 Power Interruptions

Protect nonvolatile memory, including on-chip EEPROM and external nonvolatile devices, from logic malfunctions when V_{DD} is less than the recommended operating voltage. A standard method to protect the Neuron Chip's internal EEPROM is to use an external LVI connected to "reset." This LVI can also be tied to other nonvolatile memories. It should be noted that if internal EEPROM is being written to while a sudden drop in voltage occurs, the EEPROM value may still be corrupted.

The LVI trip point must allow for noise and power supply fluctuations. In regards to external memory, consider leaving some timing margins to account for slower propagation delays at the LVI trip point.

D.5.4 Testing

A robust system design starts with good design practice, and benefits from experience and lots of testing. Testing areas include functional (board-level) and system-level testing. Voltage, timing margins, and power cycling tests should be done. Other tests include EMC testing:

Testing Description	Standard
Radiated Emissions	VDE A/B, FCC A/B
Electrostatic Discharge	IEC 801-2 Levels 1 – 4, MIL883
Radiated Susceptibility	IEC 801-3 Levels 1 – 3
Burst Noise Susceptibility	IEC 801-4 Levels 1 – 4
Surge Susceptibility	IEC 801-5 Classes 0 – 5

Safety tests are not done on all products. Environmental testing vary temperature, humidity, altitude, and strife. Strife testing stresses hardware to emulate thermo-mechanical problems and aging. Figure D-16 shows one example of temperature cycling the product.

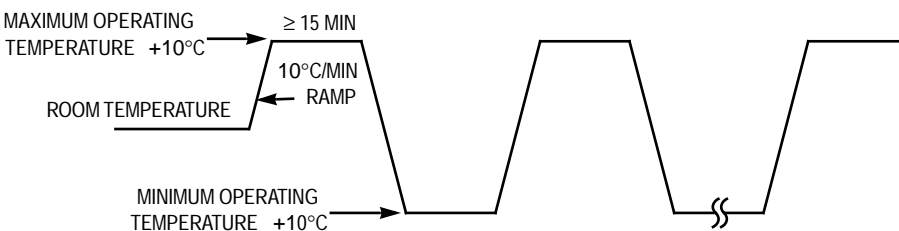


Figure D-16. Stress Hardware to Emulate Thermo-Mechanical Problems and Aging

D.5.5 Product Testing and Design Validation

IEC 801-2, ESD Testing, finds the majority of problems, but the following tests are also recommended.

IEC 801-2:	ESD	Level 4
IEC 801-3:	RF Susceptibility	Level 3
IEC 801-4:	Burst	Level 4
IEC 801-5:	Surge	Level 3
EMI Testing		

IEC 801-2, ESD Testing, is performed on a metal test table using an ESD transient generator. Level 4 testing involves injecting up to ± 8 kV contact discharges and up to ± 15 kV air discharges into the product under test. Depending on the product design, discharges may be injected at the network connector, power connector, and other user-accessible areas. Normal product operation should not be interrupted by the discharges, although one packet may be corrupted on the network by each static discharge to the network connector or cable.

IEC 801-3, RF Susceptibility Testing, is generally performed in an RF-shielded anechoic chamber. The product under test is placed on a non-conducting table in the chamber, and antennae are used to subject the product to intense radio frequency fields. Normal product operation should not be interrupted by the RF field, and normal network communication should continue. Level 2 testing is performed with a field of 3 V/m, which is classified by the test standard as a “moderate electromagnetic radiation environment.” Level 3 testing is performed with a field of 10 V/m, which is classified by the standard as a “severe electromagnetic radiation environment.”

IEC 801-4, Burst Testing, is performed on a non-conducting table, with 1 m of the network cable clamped in a high-voltage burst generation apparatus. Normal product operation should not be interrupted by the bursts, although one packet may be corrupted on the network by each burst. There are three bursts injected onto the network cable each second. Level 3 testing is performed with ± 1 kV bursts, which are classified by the test standard as representative of a “typical industrial environment.” Level 4 testing is performed with ± 2 kV bursts, which are representative of a “severe industrial environment.”

IEC 801-5, Surge Testing, is performed on a non-conducting table using specialized surge generation equipment. The surges are injected directly into the network wiring via a coupling circuit. Normal product operation should not be interrupted by the surges, although one packet may be corrupted on the network by each surge. Level 2 testing is performed with up to ± 1 kV surges, and Level 3 testing is performed with up to ± 2 kV surges.

D.5.6 Board Layout Issues and Guidelines

ESD and EMI are two of the most important design considerations when laying out the PCB for a node.

Tolerance of ESD and other types of network transients requires good layout of the power, ground, and reset circuitry. In general, an ESD discharge current will return to earth ground or other nearby metal structures. The node's ground scheme must be able to pass this ESD current between the network connection and the node's external ground connection without generating significant voltage gradients across the node's PCB. The low-inductance star ground scheme (discussed below) accomplishes this task. The reset traces on the node's PCB are sensitive to noise pickup, so small decoupling capacitors are used on the RESET pins at the Neuron Chip to ensure that ESD does not induce false resets.

In summary, the following general rules applies to EMI and EMC:

- The faster the Neuron Chip clock speed (1.25 MHz to 20 MHz), the higher the level of EMI.
- Better V_{DD} decoupling quiets RF noise at the sources (the digital ICs) and lowers EMI.
- The 3120 Neuron Chip will generate less EMI than the 3150 Neuron Chip, since the 3120 has no external memory interface lines.
- A 4-layer PCB will generate less EMI than a 2-layer PCB, since the extra layers facilitate better V_{DD} decoupling and more effective logic ground guarding.
- Ferrite beads in series with the network traces at the network connector, and ferrite chokes in series with the power input traces at the power connector, can be used to help meet EMC requirements for nodes that have noisy application circuitry or special circuit requirements. See Figure D-17 for an example of using ferrite beads at the power connector.

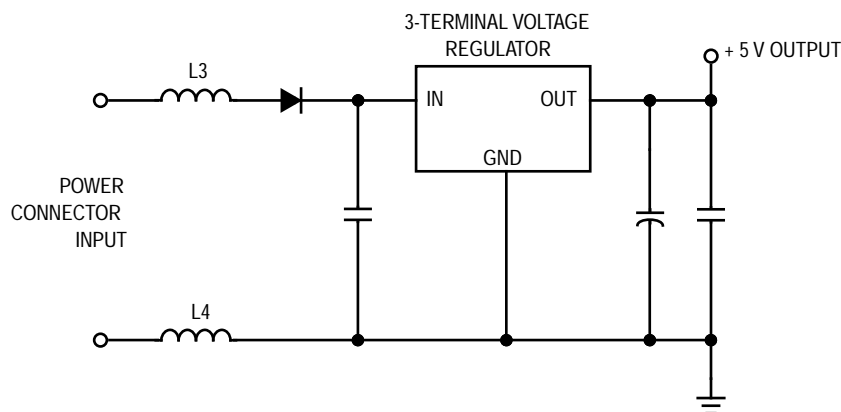


Figure D-17. Using Ferrite Beads at the Power Connector Input

Early testing of prototype circuits at an outdoor EMI range should be used to determine the effectiveness of these EMC techniques in a particular application.

Star Ground Configuration: The distribution of ground on the PCB should be in the form of a star, with the power connector, network connector, and any “chassis ground” connection all located as close as practical to the center of the star. This star ground distribution serves to separate the ground returns for the various functional blocks in a node so that transients are conducted out of the node with a minimum of disruption to other function blocks. The “chassis ground” connection is at the very center of the star. If the node has a metal chassis, the ESD and other transients will generally return to that chassis. If the node’s logic ground is connected to this chassis ground, then connection should be made at this single point only. If a node is housed in a plastic enclosure and it is powered with an isolating transformer, then there may not be any explicit earth ground or “chassis” ground available. In this case, it is still important for the network connector and power supply connector to be located at the center of the star.

Ground Planes: As ground is routed from the center of the star out to the function blocks on the board, planes or very wide traces should be used to lower the inductance of the ground distribution system. In order to ensure that the Neuron Chip has low-impedance grounding, a ground plane can be designed under the Neuron Chip. The Neuron Chip is typically on the component side of the printed circuit board. Since SMT parts on the component side do not routinely feed through to the solder side of the board, this puts the low-impedance ground plane on the same side as the Neuron Chip.

+ 5 V Power Distribution and Decoupling: In general, V_{DD} should be distributed through low-inductance traces and planes in the same manner as ground.

Reset Routing and Decoupling: The reset line is monitored by the Neuron Chip. The V_{DD} inputs at the Neuron Chip must be well decoupled to the ground plane through low-inductance capacitors. Additional V_{DD} decoupling capacitors can be placed around the Neuron Chip for improved EMI performance. For a 3120 Neuron Chip, the V_{DD} input at pin 32 is used for the reset signal comparison, so it is especially important for that V_{DD} pin to have an adjacent decoupling capacitor. For the 3150 Neuron Chip, the V_{DD} input to pin 7 is used for this reset comparison.

Ground Guarding of Clock Signals: The Neuron Chip and any other fast digital circuitry should be kept away from the network traces. If noisy digital circuitry is located too close to the network connector, RF noise may couple out onto the network cable and cause EMI problems.

ESD Keepout Area: No traces or other metal should be placed on the PCB in the area of the network connect, or except the circuitry needed for the network. In general, keep all other circuitry and metal patterns at least 0.175 in. (4.45 mm) away from the network connector and protection components. Do not place stick-on labels in this area.

EMI Keepout Area: It is important to keep RF noise from coupling from the Neuron Chip and digital node circuitry out onto the network wires.

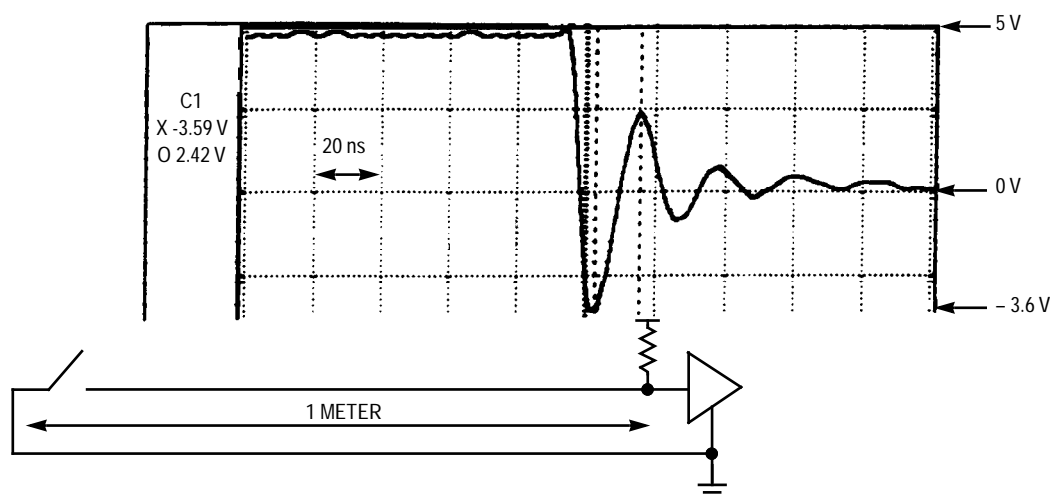
D.6 CMOS LATCH-UP

Latch-up will not be a problem for most designs, but the designer should be aware of it, what causes it, and how to prevent it.

Figure D-18 shows the layout of a typical CMOS inverter and Figure D-19 shows the parasitic bipolar devices that are formed. The circuit formed by the parasitic transistors and resistors is the basic configuration of a SCR. In the latch-up condition, transistors Q1 and Q2 are turned on, each providing the base current necessary for the other to remain in saturation, thereby latching the devices on. Unlike a conventional SCR, where the device is turned on by applying a voltage to the base of the NPN transistor, the parasitic SCR is turned on by applying a voltage to the emitter of either transistor. The two emitters that

The diagram illustrates a cross-section of a CMOS inverter. It features an N-SUBSTRATE with a P-WELL region. The P-channel MOSFET (P-MOS) is located on the N-SUBSTRATE, with its gate connected to the INPUT and its source to VDD. The N-channel MOSFET (N-MOS) is located within the P-WELL, with its gate also connected to the INPUT and its source to VSS. Both MOSFETs share a common DRAIN connection, which is the OUTPUT. The diagram also shows FIELD OXIDE regions and N+ and P+ doped areas for contacts and gates.

The pnpn structure works like a silicon-controlled rectifier. Once on, power must be removed to turn it off. Figure D-20 shows an example of a closing switch producing a negative spike, possibly latching up the CMOS device.



9-81

Once a CMOS device is latched up, if the supply current is not limited, the device will be destroyed. Ways to prevent such occurrences are listed below.

1. Ensure that inputs and outputs are limited to the maximum rated values, as follows:
 - $-0.3 \leq V_{in} \leq V_{DD} + 0.3 \text{ Vdc}$ referenced to V_{SS}
 - $-0.3 \leq V_{out} \leq V_{DD} + 0.3 \text{ Vdc}$ referenced to V_{SS}
 - $|I_{in}| \leq 10 \text{ mA}$
 - $|I_{out}| \leq 10 \text{ mA}$ when transients or dc levels exceed the supply voltages.
2. If voltage transients of sufficient energy to latch up the device are expected on the outputs, external protection diodes can be used to clamp the voltage. Another method of protection is to use a series resistor to limit the expected worst case current to the Maximum Ratings values. See Figure D-4.
3. If voltage transients are expected on the inputs, protection diodes may be used to clamp the voltage, or a series resistor may be used to limit the current to a level less than the maximum rating of $I_{in} = 10 \text{ mA}$. See Figure D-4.
4. Sequence power supplies, so that the inputs or outputs of CMOS devices are not powered up first (e.g., recessed edge connectors may be used in plug-in board applications and/or series resistors).
5. Power supply lines should be free of excessive noise. Care in board layout and filtering should be used.
6. Limit the available power supply current to the devices that are subject to latch-up conditions. This can be accomplished with the power-supply filtering network or with a current-limiting regulator.

Using industrial controllers for driving relays or motors creates an environment in which latch-up is a potential problem. Also, ringing due to inductance of long transmission lines in an industrial setting could provide enough energy to latch up CMOS devices.

RECOMMENDED READING

Paul Mannone, "Careful Design Methods Prevent CMOS Latch-Up", EDN, January 26, 1984.

D.7 RECOMMENDED BYPASS CAPACITOR PLACEMENT

Proper decoupling is required to ensure proper operation of the Neuron Chip. Table D-1 and Figure D-21 show the recommended capacitor placement for the MC143150 and MC143120. Whenever possible, match V_{DD} names with V_{SS} names. For example, in the MC143150 table: V_{DD} -Reset with V_{SS} -Reset. In all cases, this may not be possible, such as in the MC143150 table: V_{SS} -Core with V_{DD} -Core. In the case where there are two adjacent V_{SS} or V_{DD} pins, they can be tied together and only one capacitor used.

When connecting capacitors to Neuron Chips, make the leads as short as possible. All V_{DD} pins must be tied to + 5 V, and all V_{SS} pins to ground. Keep the crystal circuit close to the Neuron Chip and isolated from communication lines.

Bypass capacitors should be 0.1 μF or 0.33 μF ceramic or dipped-mica capacitors and should be placed as close to V_{DD} pins as possible. V_{DD} and GND loops should be avoided. Minimum recommended configurations are:

- MC143150: 0.1 μF bypass capacitor between pins: 7:8, 21:22, 25:26, 39:40.
- MC143120: 0.1 μF bypass capacitor between pins: 10:11, 12:13, 16:18, 31:32.

Figure D-22 shows suggested bypass capacitor placement and crystal circuit trace outlines.

Table D-1. Recommended Capacitor Placement

Pin	Function	Recommendation
MC143150 (64-Pin PQFP)		
7	V _{DD} — Reset	Mandatory
8	V _{SS} — Reset	Mandatory
9	V _{SS} — Core/CPU/Memory	Mandatory
19	V _{SS}	Desired
20	V _{DD}	Desired
21	V _{SS} — Clock Buffer Internal	Mandatory
22	V _{DD} — Clock Buffer Internal	Mandatory
25	V _{SS} — Oscillator/Transceiver	Mandatory
26	V _{DD} — Oscillator/Transceiver	Mandatory
39	V _{SS} — I/O	Mandatory
40	V _{DD} — I/O	Mandatory
41	V _{DD} — Core/CPU/Memory	Mandatory
44	V _{DD}	Mandatory
MC143120 (32-Pin SOG)		
2	V _{DD}	Desired
9	V _{SS} — Core/CPU/Memory	Mandatory
10	V _{SS} — I/O	Mandatory
11	V _{DD} — I/O	Mandatory
12	V _{DD} — Core/CPU/Memory	Mandatory
13	V _{SS} — Core/CPU/Memory	Mandatory
16	V _{SS} — Oscillator/Transceiver	Mandatory
18	V _{DD} — Oscillator/Transceiver	Mandatory
23	V _{SS}	Desired
25	V _{DD} — Core/CPU/Memory	Mandatory
31	V _{SS} — I/O	Mandatory
32	V _{DD} — I/O	Mandatory
MC143120 (44-Pin QFP)		
7	V _{SS} — Core/CPU/Memory	Mandatory
8	V _{SS} — I/O	Mandatory
9	V _{DD} — I/O	Mandatory
10	V _{DD} — Core/CPU/Memory	Mandatory
13	V _{SS} — Core/CPU/Memory	Mandatory
16	V _{SS} — Oscillator/Transceiver	Mandatory
19	V _{DD} — Oscillator/Transceiver	Mandatory
26	V _{SS}	Desired
29	V _{DD} — Core/CPU/Memory	Mandatory
37	V _{SS} — I/O	Mandatory
38	V _{DD} — I/O	Mandatory
41	V _{DD}	Desired

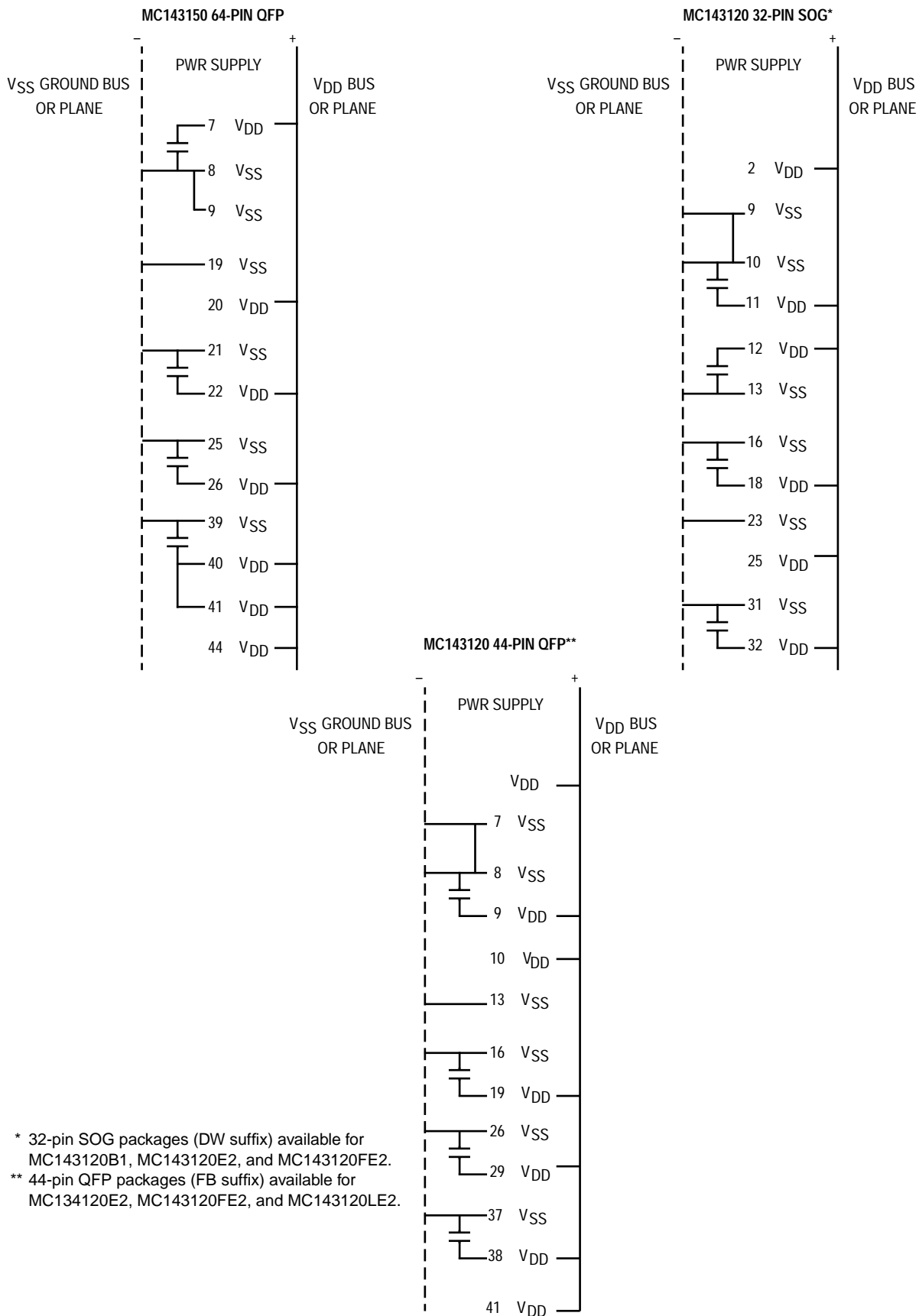
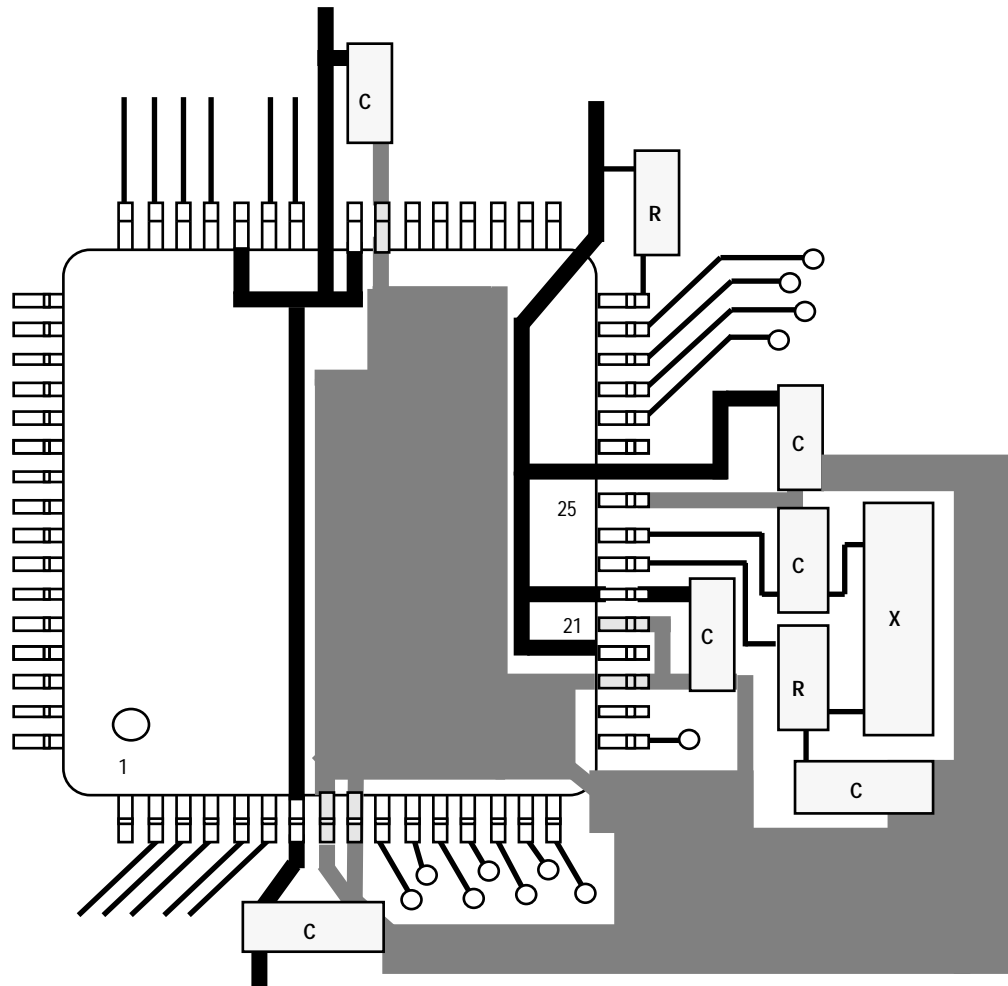


Figure D-21. MC143150 and MC143120 Recommended Bypass Capacitor Configuration

MC143150
64-LEAD QUAD FLAT-PACK



KEY LAYOUT RULES

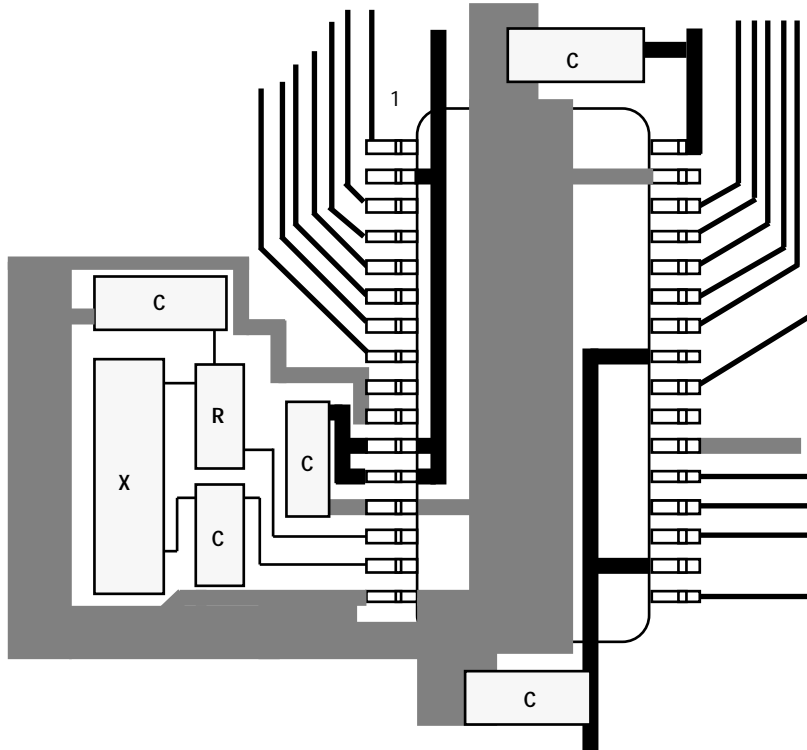
1. If possible, use 4-layer (or more) boards. This would greatly simplify the layout and reduce grounding and noise-related problems.
2. For 2-layer boards, the four bypass capacitors must be close to the Neuron IC. V_{DD} and ground must be large traces to reduce inductance and noise.
3. The crystal must be isolated from any digital signal. If clock 2 is being used for other circuit signals, keep the trace short or buffer it. The added board capacitance and input capacitance of other devices being driven will skew the crystal frequency.
4. On 2-layer boards, avoid running high-frequency digital signal traces under crystal circuit or input pins of the communications signals, on opposite sides of the board.
5. Ensure that power supply and ground traces are large enough to handle the peak surge switching currents. Otherwise there will be power supply dips on the V_{DD} pins, which may cause errors in the checksum calculation, resulting in the Neuron IC going applicationless.

NOTE:

C = Capacitor, surface mount.
R = Resistor.
X = Crystal.

Figure D-22. Minimum Recommended Capacitor Placement (Sheet 1 of 2)

MC143120
32-LEAD SOG



MC143120
44-LEAD QFP

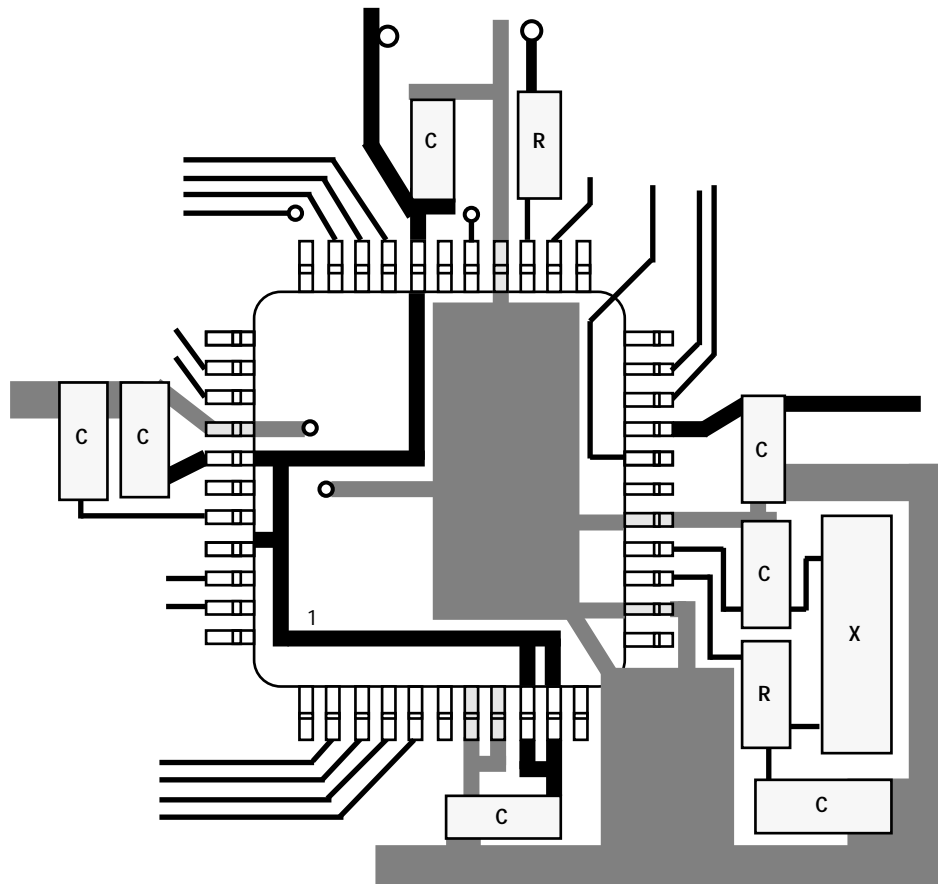
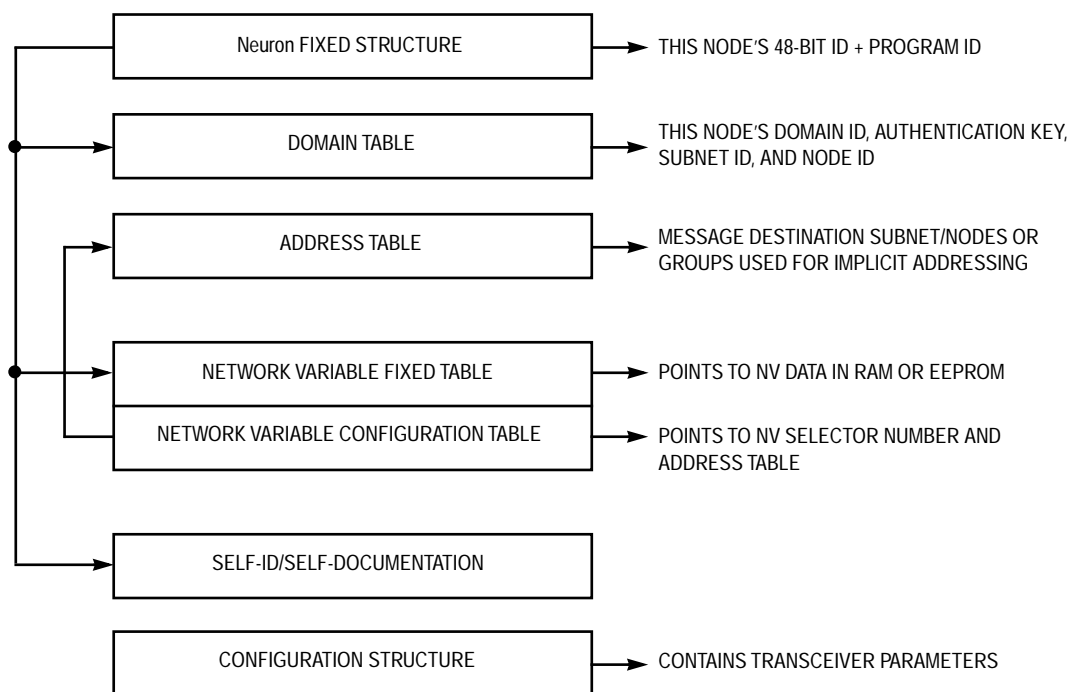


Figure D-22. Minimum Recommended Capacitor Placement (Sheet 2 of 2)

APPENDIX E

Neuron IC MEMORY MAPS

This appendix contains the memory structures of a Neuron Chip that govern addressing and network access, and graphical representations of most of these structures.



Structure	Location	
	MC143120B1	MC143120E1/E2, MC143150B1/B2
Fixed Read-Only Data Structure	0xF000	0xF000
Configuration Data Structure	0xF024	0xF029
Boot ID	0xF1FE (Neuron 3150 Chip only)	0xF1FE (Neuron 3150 Chip only)
Domain Table	0xF03D	0xF042
Address Table	0xF03D + 15 (decimal) bytes per domain	0xF042 + 15 (decimal) bytes per domain
Network Variable Configuration Table	Address table + 5 bytes per address	Address table + 5 bytes per address

NOTE: MC143120B1 contains version 4 firmware. MC143120 contains version 3 firmware. MC143150 firmware version depends on the software which exported it.

Figure E-1. Six Major Memory Structures

The following memory maps are not in the order placed in the Neuron Chip.

The Neuron fixed structure is always found in on-chip EEPROM.

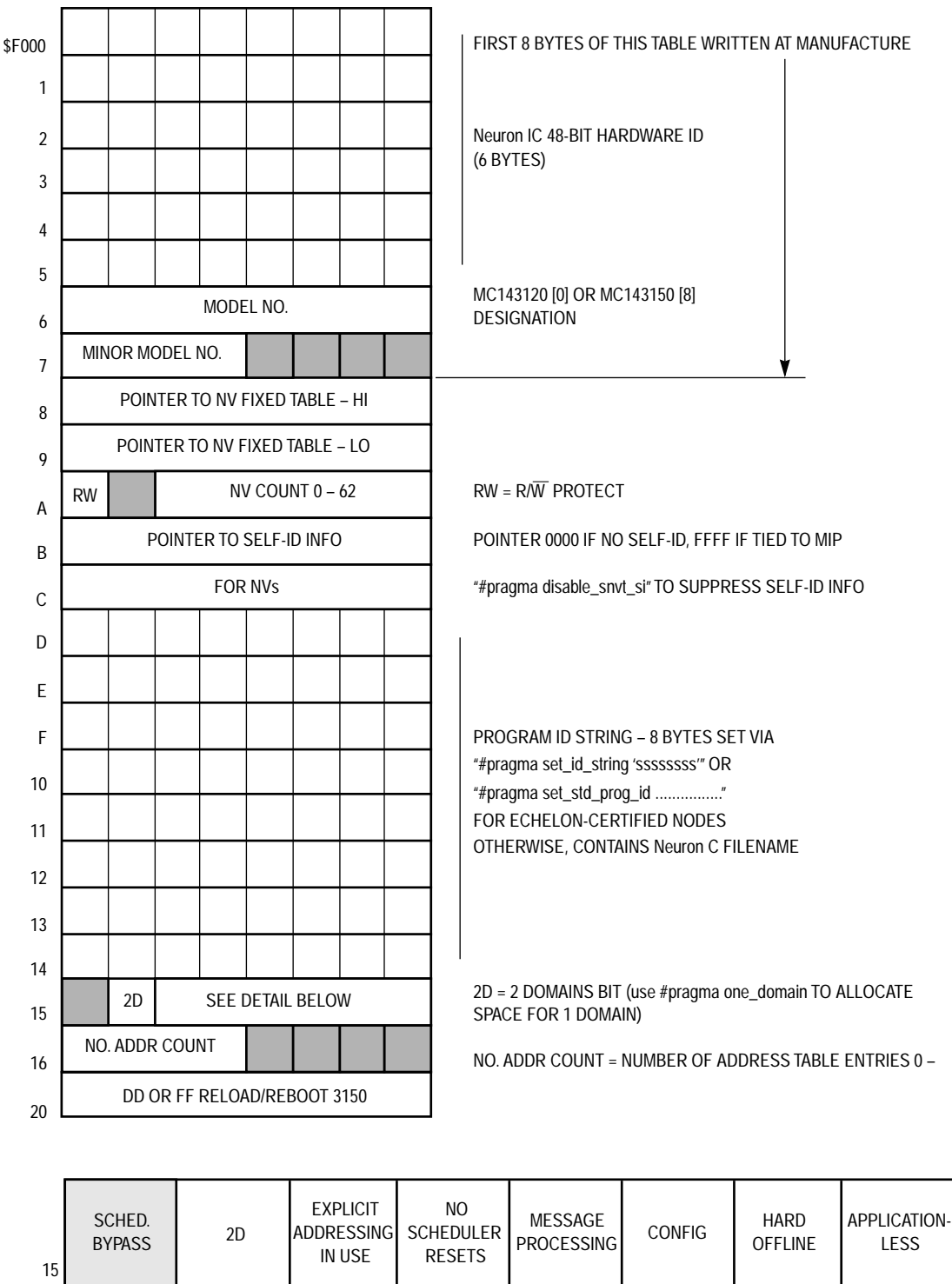


Figure E-2. Neuron Fixed Structure

The Neuron Chip domain table is always found in on-chip EEPROM. There are two tables if there are two domains.

- Read this table with “access_domain” NMgmt messages.
- Write with “update_domain” NMgmt messages.
- Configuration checksum is rewritten when this table is updated.

OFFSET \$00											
1											
2											
3											
4											
5											
6	SUBNET ID 1 – 255										
7		NODE ID 1 – 127									
8	DOMAIN LENGTH										
9											
A											
B											
C											
D											
E											

6-BYTE DOMAIN ID (IF SMALLER 0, 1, AND 3 JUSTIFIED TO THE TOP OF THIS FIELD)

0...1...3...6 BYTES \$FF = DOMAIN NOT IN USE

6-BYTE AUTHENTICATION KEY

Figure E-3. Neuron Chip Domain Table

The Neuron Chip address table is always found in on-chip EEPROM, and is up to 15 entries, each 5 bytes long.

- There are five different formats: Subnet/Node, Group, Broadcast, Turnaround, and Not-In-Use.
- A Neuron C program can write its own node's address table; i.e., Self-Installation.
- Explicit message tags use the first entries, then NVs.

NOTES: \$F03D + 15 (decimal) bytes per domain (MC143150FU/FU1, MC143120DW/B1).

\$F042 + 15 (decimal) bytes per domain (MC143120E1/E2, MC143150B1FU).

The default setting provides two domains. The pragma statement `#pragma one_domain` sets one domain.

SUBNET/NODE			
OFFSET \$00	FORMAT OF ADDRESS TABLE		FORMAT = 1 DEFINES SUBNET/NODE ADDRESS TABLE FORMAT
1	DI	NODE ID	DI = DOMAIN INDEX
2	RPT TIMER	RETRY	TABLE A-2 HAS 4-BIT CODES FOR EACH OF THESE SETTINGS*
3	RCV TIMER	Tx TIMER	
4	SUBNET NUMBER 1 – 255		
GROUP			
OFFSET \$00	FMT	GROUP SIZE	FORMAT = 1 FOR GROUP; GROUP SIZE = 2 – 64; 0 = UNLIMITED
1	DI	GROUP MEMBER ID	DI = DOMAIN INDEX; MEMBER ID IN A GROUP 0 – 63
2	RPT TIMER	RETRY	TABLE A-2 HAS 4-BIT CODES FOR EACH OF THESE SETTINGS*
3	RCV TIMER	Tx TIMER	
4	GROUP ID		GROUP NO. WITHIN THIS DOMAIN
BROADCAST			
OFFSET \$00	FORMAT OF ADDRESS TABLE		FORMAT = 3 DEFINES BROADCAST ADDRESS TABLE FORMAT
1	DI		DI = DOMAIN INDEX
2	RPT TIMER	RETRY	TABLE A-2 HAS 4-BIT CODES FOR EACH OF THESE SETTINGS*
3	RCV TIMER	Tx TIMER	
4	SUBNET NUMBER 1 – 255		SUBNET = 0 MEANS DOMAIN-WIDE BROADCAST. FIRST ACK OR RESPONSE RECEIVED FROM BROADCAST MESSAGE COMPLETES THE TRANSACTION
TURNAROUND/UNUSED			
OFFSET \$00	FORMAT OF ADDRESS TABLE		FORMAT = 0 DEFINES TURNAROUND OR UNUSED
1	1 IF TURNAROUND; 0 IF UNUSED		ADDRESS TABLE FORMAT
2			READ THIS TABLE WITH "access_address"
3			WRITE WITH "update_address"
4			CONFIGURATION CHECKSUM IS REWRITTEN WHEN TURNAROUND TABLE IS UPDATED

* RPT TIMER = Interval between unacknowledged repeat tries.

RETRY = Retry count.

RCV TIMER = If node receives a group message, it will wait this time interval allowing same transaction IDs to be seen as retries versus new messages.

Tx TIMER = Delay before retry is attempted.

Figure E-4. Neuron Chip Address Table

The network variable tables create both a fixed-structure entry and a configured-structure entry.

- Use “config_data_nv_fixed” call to find the top of the NV fixed-structure table’s location.
- Use “nv_table_index” call to find a specific NVs configured-structure table entry.
Read this table with “access_NV”.
Write with “update_NV”.
- These tables move to the host MPU when the MIP is used.
- Network variables are “BOUND” when:
 - The sender has the destination node’s address in its address table.
 - The NV selector numbers match between sender and destination nodes.

The network variable table(s) start after the address table(s).

Address Start: Address table + 5 bytes per address table entry.

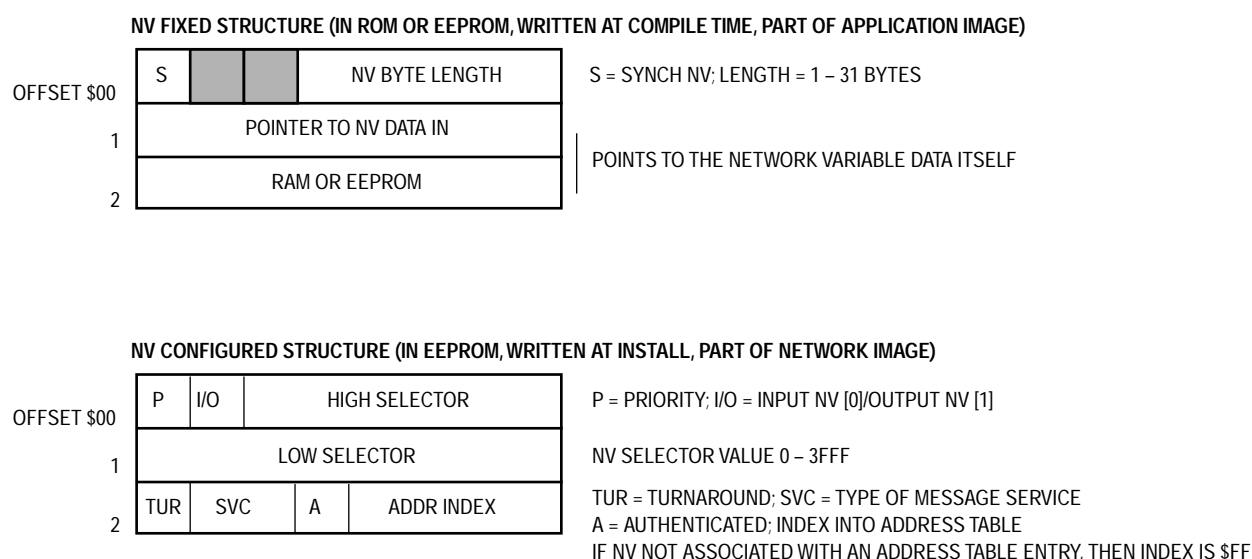
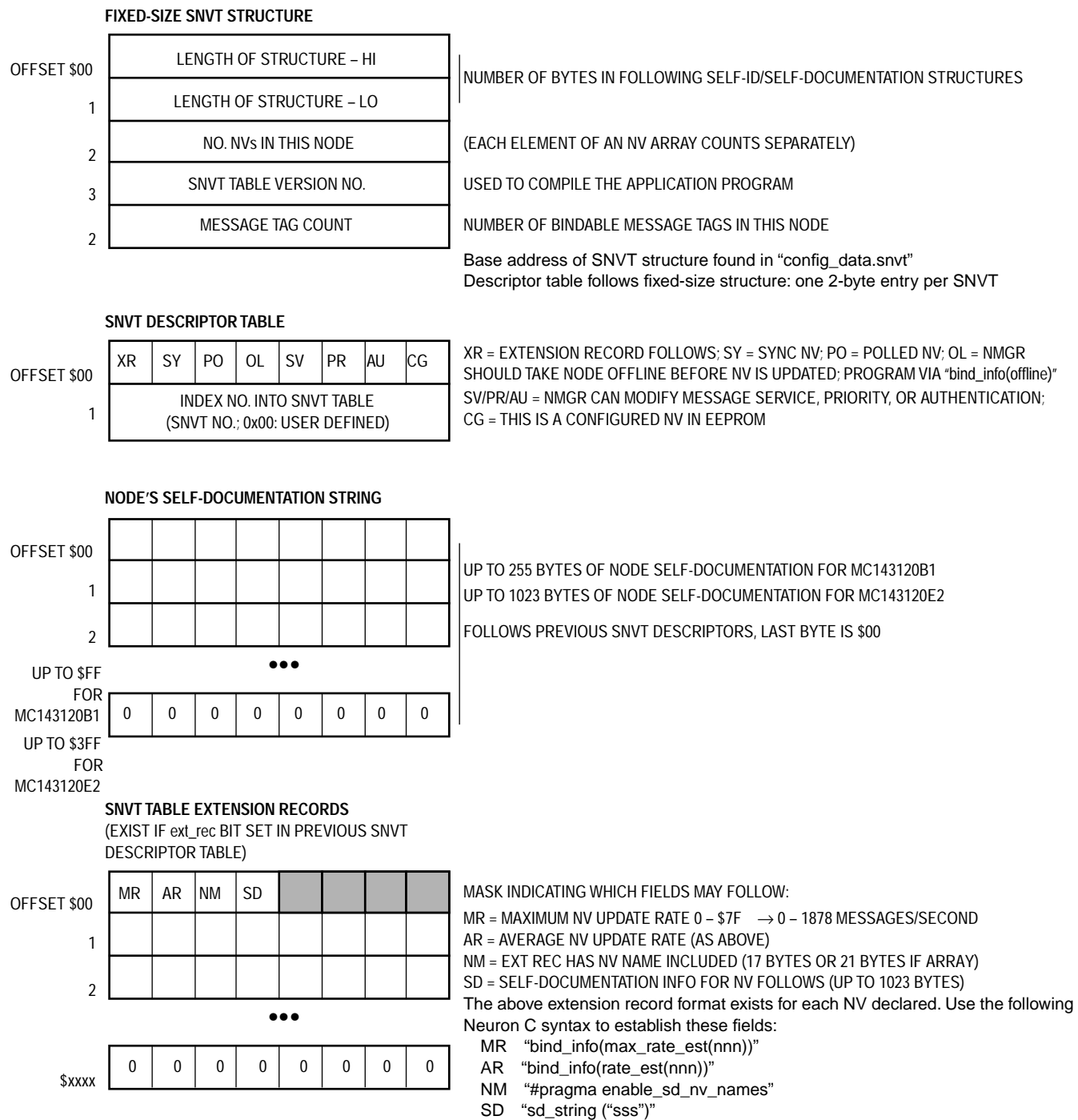


Figure E-5. Network Variable Tables

The SNVT structures consist of four self-ID and self-documentation structures which are part of the application image.



NOTE: “#pragma disable_snvt_si” will prevent these tables from being allocated.

Figure E-6. SNVT Structures

The Neuron configuration structure is always found in on-chip EEPROM.

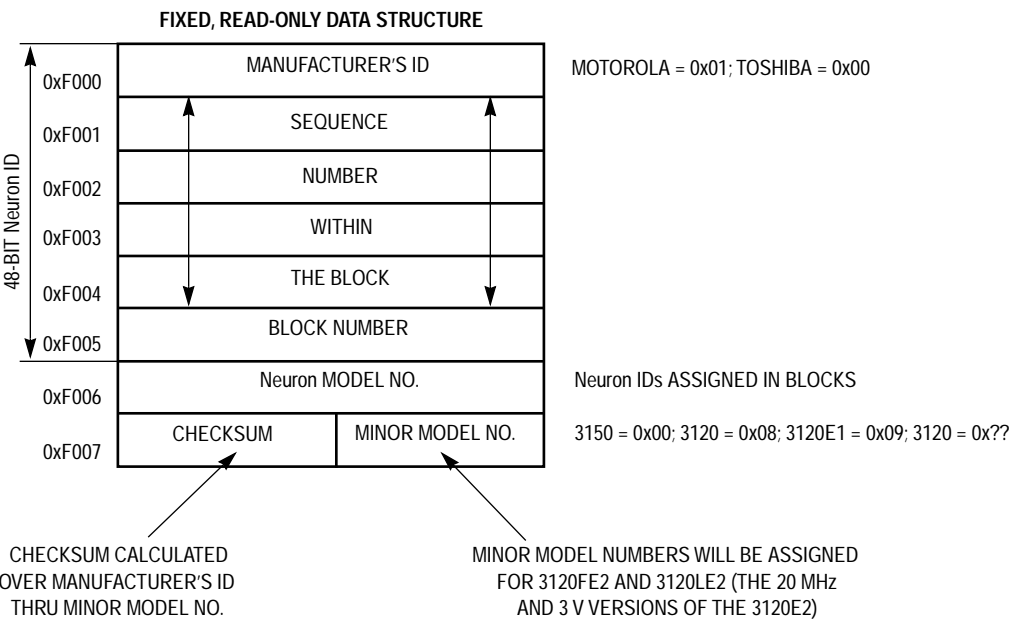
OFFSET \$00									CHANNEL ID – LonBuilder TOOL CREATES (NOT USED BY Neuron FIRMWARE – FOR NETWORK DATABASE RECONSTRUCTION)
1									6-BYTE NODE PHYSICAL-LOCATION STRING
2									
3									
4									
5									
6									
7									
8	COMM CLOCK				INPUT CLK				INPUT = OSC FREQ, COMM = DIV RATIO FOR NETWORK XCVR TYPE; DIRECTION OF COMM PINS BIT 0 = CP0
9	COMM TYPE		COMM PIN DIR						
A	PREAMBLE LENGTH PAD								NUMBER OF PROCESSOR CYCLES FOR THESE XCVR PARAMETERS SET ON LonBuilder TOOL AS RAW DATA IN CHANNEL SCREEN
B	PACKET CYCLE PAD								
C	BETA PAD								
D	TRANSMIT BETA 1 PAD								
E	RECEIVE BETA 1 PAD								
F	NODE PRIORITY								PRIORITY SLOT NUMBER (0 – 255) THAT A NODE USES ON A CHANNEL
10	CHANNEL PRIORITIES								
11	CD	BST		FILTER		HYSTER		7-BYTE FIELD FOR SPECIAL PURPOSE MODE XCVRs	
12							TL PR		
13								(FIRST 2 BYTES ARE OVERLAID ON THIS FIELD IF DIRECT-MODE XCVR IS USED AS SHOWN HERE) CD = COLLISION DETECT; BST = NO. OF BITS INTERPRETED AS BIT SYNC FILTER = GLITCH FILTER (TABLE 4-4) HYSTER = HYSTERESIS FILTER (TABLE 4-3) TL/PR = COLLISION DETECT AT END OF PACKET OR PREAMBLE	
14									
15									
16									
17									
18	NON-GP TIMER			AM	PREEMP TIME				UNICAST RECEIVE TIMER; AUTHENTICATE NETWORK MANAGEMENT; PREEMP TIME = WAIT TIME FOR A FREE BUFFER

NOTE: Application program can read (only) this data with “config_data” declaration.
Network management messages can read memory or write memory to access.

Figure E-7. Neuron Configuration Structure

Miscellaneous

0XF1FD Contains the error log of the last error found. These do not include network errors. Error log messages can be found in the “error-log” enumeration found in Appendix B.2.



0XF1FE Boot ID high byte

0XF1FF Boot ID low byte

A typical MC143120B1 EEPROM fresh from the factory may contain the following memory values:

F000: 01 00 0f 68 a7 00 09 51 00 00 00 00 00 00 00 00
F010: 00 00 00 00 00 03 f0 02 bb bb 00 22 24 00 00 00
F020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
F030: 00 05 ac 03 01 00 00 00 00 00 00 00 00 00 00 00

rest of bytes until and including \$F1FF contain 0s.

APPENDIX F SUPPORT TOOLS

LONWORKS[®] Support Tools

Motorola's LonBuilder[®] Support tools offer the user a quick and flexible means to demonstrate or test a LONWORKS-based product which has been developed and debugged on the LonBuilder Developer's Workbench. The family of tools consists of Neuron[®] Chip-based development boards, I/O application boards, a Differential Direct-Connect Transceiver board (for the LonBuilder Developer's Workbench), and a Neuron Chip Test/Programming board. These tools offer the unique advantages of:

- Capability to Test Products Over a Wide Range of Media and Data Rates
- All Boards (Except the LiteNodes) Have RJ-45 Connectors Allowing Ease of Connectivity
- Neuron Chip Boards Contain a 5 V Regulator Allowing for a Wider Range of Power Supply Voltages
- Common 2 x 10 Connector for Interface to the Neuron Chip I/O Pins (Does Not Apply to the LiteNode Family)
- Library of Application Functions Is Available from Motorola
- Inexpensive Means of Demonstrating LONWORKS-Based Products

This document covers a brief detail on each of the boards. For further information, contact Motorola's LONWORKS applications support team in Austin, TX, at 512-934-7610 or fax 512-934-7991. The Motorola LONWORKS web site address is <http://motorola.com/lonworks>.

The family consists of the following:

- M143120DWEVK Neuron Evaluation/Programming Kit With 3120[®] 32-Pin Socket
- M143120FBEVK Neuron Evaluation/Programming Kit With 3120[®] 44-Pin Socket
- M143150EVK Neuron Evaluation/Programming Kit With 3150[®] Socket
- M143204EVK LonBuilder Emulator Boards Direct Drive Transceiver
- M143206EVK Neuron I/O Application Kit, 5 LEDs, A/D, D/A, Clock
- M143208EVK I/O Test Board, 11 LEDs and 11 Switches
- M143221EVK RS-232 EVBU Interface Board for M143120EVK/M143150EVK
- M143232EVK Voice Development Kit, to Interface to LonBuilder or Neuron Board
- M143235EVK Neuron IC Evaluation Kit with MC143120E2, RS-485 Transceiver, Full 11 I/O Access, Two Isolated Input/Output Ports
- MC143238EVK LiteNode; Two-In, Two-Out; MC143120E2 Node
- MC143239EVK LiteNode; H-Bridge; MC143120E2 Node
- MC143240EVK LiteNode; Four-Channel A/D; MC143120E2 Node
- MC143245EVK LiteNode; Gateway/Programmer; MC143150B2 Node

All the boards are available from Motorola and our franchised distributors.

Echelon, LON, LonBuilder, LonManager, LonTalk, LonUsers, LONWORKS, Neuron, 3120, 3150, and NodeBuilder are registered trademarks of Echelon Corporation. LonLink, LonMaker, LONMARK, LONews, and LonSupport, are trademarks of Echelon Corporation.

ECHELON® SUPPORT TOOLS

LonBuilder Developer's Workbench

Thanks to Echelon's LonBuilder and NodeBuilder® tools, as well as Motorola's extensive technical support network, both system and device manufacturers can now develop control networks quickly and inexpensively. These tools provide developers with everything needed to begin building LONWORKS nodes.

The LonBuilder Developer's Workbench combines three development tools — a multi-node development system, a network manager, and a protocol analyzer — into an integrated hardware and software development environment. This development system provides the tools to create software applications and prototype hardware on a network ranging from two to hundreds of nodes. The network manager installs and configures nodes during development, making them easy to connect, define, and build. The protocol analyzer monitors the network and interprets its activity.

The LonBuilder Developer's Workbench includes a PC interface card, two LONWORKS transceivers, an expandable development station with two Neuron Chip emulator cards, DOS-based software for compiling, loading, integrating, and testing LONWORKS applications, and software for monitoring and controlling a LONWORKS application.

The LONWORKS NodeBuilder Development Tool is used to design LONWORKS nodes. The NodeBuilder tool does not include the system integration and test tools incorporated into the LonBuilder Developer's Workbench, but does include all the tools required to compile, load, and test code for a LONWORKS node. NodeBuilder includes Windows-based software, a PC interface card, a prototype LONWORKS node, and two LONWORKS transceivers that are used to develop and test LONWORKS nodes.

For further information on the Echelon LONWORKS Product Family call Echelon at 1-800-258-4566.

Transceivers

FTT-10A Free Topology Twisted-Pair Transceiver Models 50051, 50051-1, and 50050-01 (78 kbps)
LPT-10 Link Power Twisted-Pair Transceiver Model 50040 (78 kbps)
TPT/XF Twisted-Pair Transceiver Models 50010-10 (78 kbps) and 50020-10 (1.25 Mbps)
PLT-10A Power Line Transceiver Model 50080 (10 kbps)
PLT-21 Power Line Transceiver Model 50090-01 (5 kbps)
PLT-30 Power Line Transceiver Model 50100-01 (2 kbps)
PLA-21 Amplifier Model 53001-01

Control Modules

Twisted-Pair Control Modules Models, Flash Control Modules
55010-00 (TP/XF-78), 55020-01 (TP/FT-10), and 55030-00 (TP/XF-1250)
LPI-10 Link Power Interface Module and Developer's Kit Models 56210-01 and 58020-01
PLCA Power Line Communications Analyzers Models 57010, 57010-01, and 57010-03

Router Products

RTR-10 Router Core Module Model 61000-100
LONWORKS Router Model 71000-11

Network Interface Products

LonBuilder MIP/P20 and MIP/P50 Developer's Kit Models 23201 and 23205
LonBuilder MIP/DPS Developer's Kit Models 23211 and 23215
LTS-10 SLTA Core Module Model 65200-100
PSG-10 Serial Gateway Core Module and Evaluation Kit Model 65200-200
SLTA/2 Serial LonTalk® Adapter Model 73000-1
PSG/2 Programmable Serial Gateway Model 73000-3
PCLTA PC LonTalk Adapter Model 73100-1x, 73401
SLTA/2 and Router Accessories Model 73000-1
PCC-10 PCMCIA Card Models 73200, 7830x, and 7840x
PCLTA-10 Network Interface

Network Services Products

LonManager® LonMaker™ Starter Kit Models 32100-00, 32200-0x, and 32205
LonManager LonMaker Installation Tool Models 32100-00, 32200-0x, and 32205
LonManager DDE Server Models 33000, 33000-01, and 33005
LonManager ISA-Bus and PCC Protocol Analyzer Models 33100-00 and 33100-10
LonManager NSS-10 Model 34000-100
LonManager NSI-10 Module Model 35000-100
LonManager PCNSS Card Model 34100
LonManager PCNSI Card Model 35100

Network Services Products (Continued)

LonManager LNS for Microcontrollers and NSS-10 Upgrade Models 34001-00x, and 34001-005
LonManager LNS for Windows Model 34303
LonManager LNS FASTART Package Model 34304-00x

Development Tools

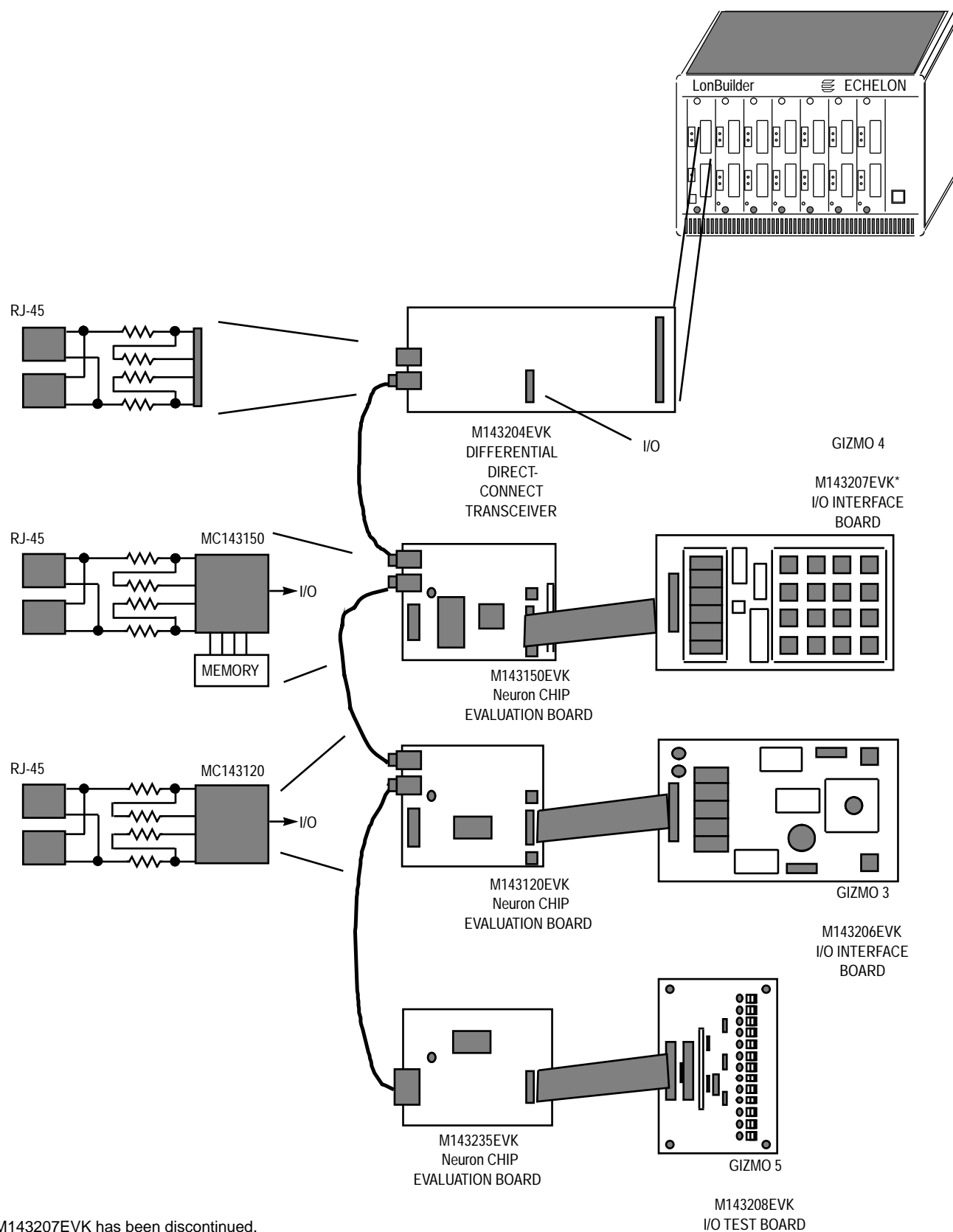
LonBuilder Starter Kit Model 20311
LonBuilder Control Processor Model 21200
LonBuilder Development Station Enclosure Model 21400
LonBuilder Neuron 3120 Programmer Model 21700
LonBuilder 3120 and 3150 Application Interface Pods Models 21820 and 21860
LonBuilder Neuron Emulator Model 25000
LonBuilder Router Model 25400
LonBuilder SMX Adapter Model 27100
LonBuilder Application Interface Kit Model 27810
LonBuilder Gizmo 3 Kit Model 28001
NodeBuilder Development Tools Model 10010
LonBuilder Application Interface Cable Model 21800
LonBuilder Module Application Interface Model 21860

SMX Transceivers

LONWORKS SMX Transceivers Models 77010, 77030, 77040, 77090, 77160, and 77180
Power Line Coupler Model 78200

Customer Support and Training Programs

LONWORKS Technology Intelligent Distributed Control Training Course (4 days)
LonSupport™ Program for LonBuilder Kits and NodeBuilder Tools
LNS Network Tools Development Course
Host-Based Node Development Course



*M143207EVK has been discontinued.

Figure F-1. Evaluation and I/O Interface Boards

M143120DWEVK

Neuron Chip Custom Node Development Board with 32-Pin Socket

The Motorola M143120DWEVK is intended for LONWORKS product developers that need to program and test application hardware and software in an actual networking environment. It provides the next step after emulation in application hardware-software debug before completing a final LONWORKS product design. A LonBuilder Developer's Workbench is required to develop application software (Neuron C), and it or some other Network Management Device can be used to install, configure, or reconfigure custom node boards over the communications network. Each board has a socket for an MC143120 Neuron IC which will contain its own unique serial number. Easy access to the Neuron Chip's 11 I/O pins and 5 communications port pins is provided via header pins and connectors. Application specific hardware can be easily bread-boarded and software-hardware interaction quickly evaluated, thereby eliminating much of the prototyping phase of product development.

The M143120DWEVK is shipped with a simple scanning application. After power up, the MC143120E2DW will sequence output pins IO_0 to IO_10 and back again. This can be used to verify that the Neuron IC is working.

The M143120DWEVK can be used to program MC143120 devices without a LonBuilder tool. This is explained in application note AN1251 in the *LONWORKS Technology Data Book*, DL159/D.

- 32-Pin Socket for Any of the 3120 Neuron IC Devices (DW Suffix)
- Can Be Used to Program 3120 Devices from Either a LonBuilder Tool or an M143150EVK
- 20 MHz Oscillator with Selectable Time Base: 20 MHz, 10 MHz, 5 MHz, 2.5 MHz, 1.25 MHz, and 625 kHz
- On-Board Linear Regulator for Wide Power Supply Input; 8 to 16 V
- Configurable Between Direct-Connect and EIA-485 Differential Communications
- Screw-Down Terminal Blocks and Two RJ-45 Connectors for Easy Wiring Connection
- Reset and Service Pin Push-Button Switches
- Power LED Indicator and Service Pin Status LED Indicator
- 2 x 10 Connector for Interface from Neuron IC I/O Pins to Standard I/O Cable
- Can Operate @ 20 MHz for Testing the MC143120FE2DW

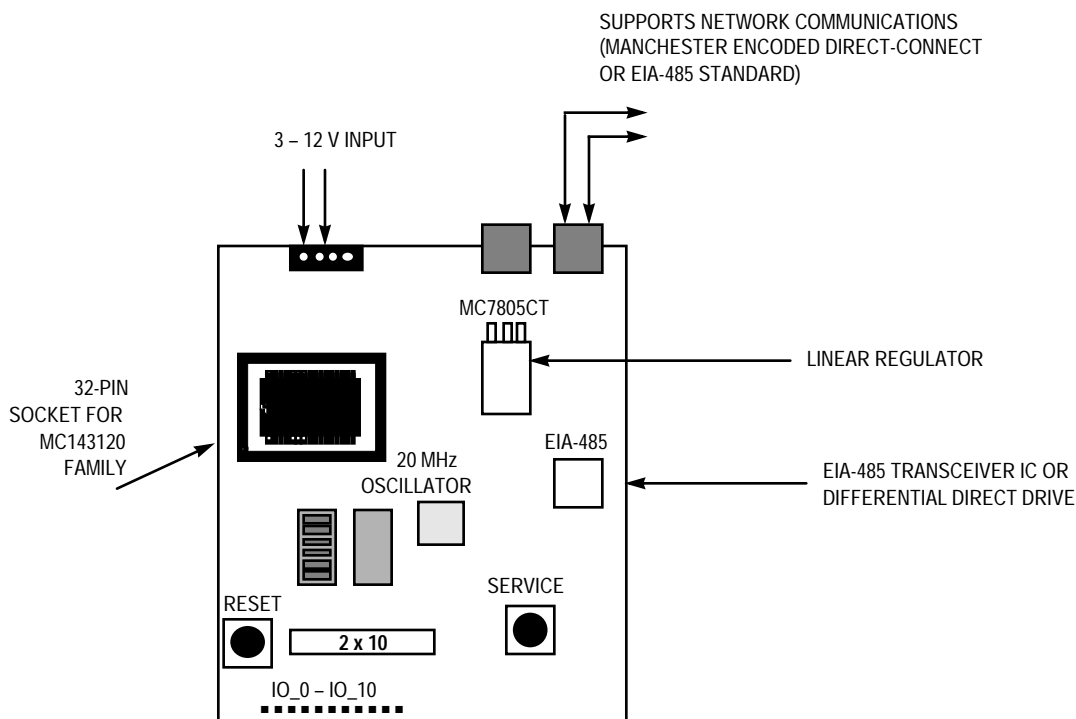
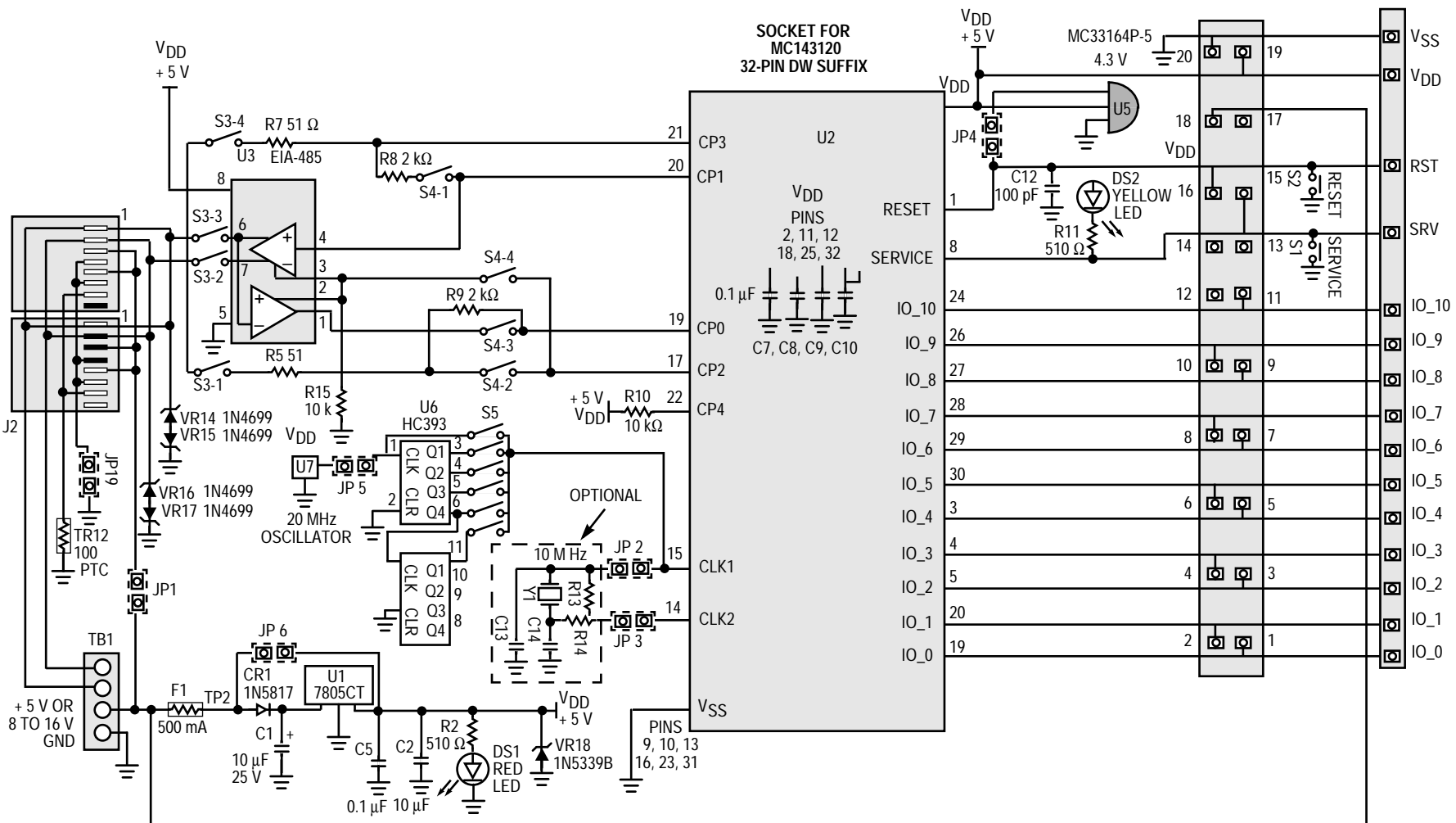


Figure F-2. M143120DWEVK Schematic Diagram



PCB layout diagram for the M143120DWEVK board, revision 2.1. The diagram shows the placement of various components including two RJ-45 ports (J2, J3), a 32-pin socket for the MC143120 (U1), a 625 MHz oscillator (U6), a 32-pin connector (J1), and various passive components like resistors (R1-R15), capacitors (C1-C15), and LEDs (DS1, DS2). It also includes a red LED (DS1), a yellow LED (DS2), and a reset button (S2). The board is labeled "BOARD REV 2.1 M143120DWEVK" and "OPTIONAL".

M143120FBEVK

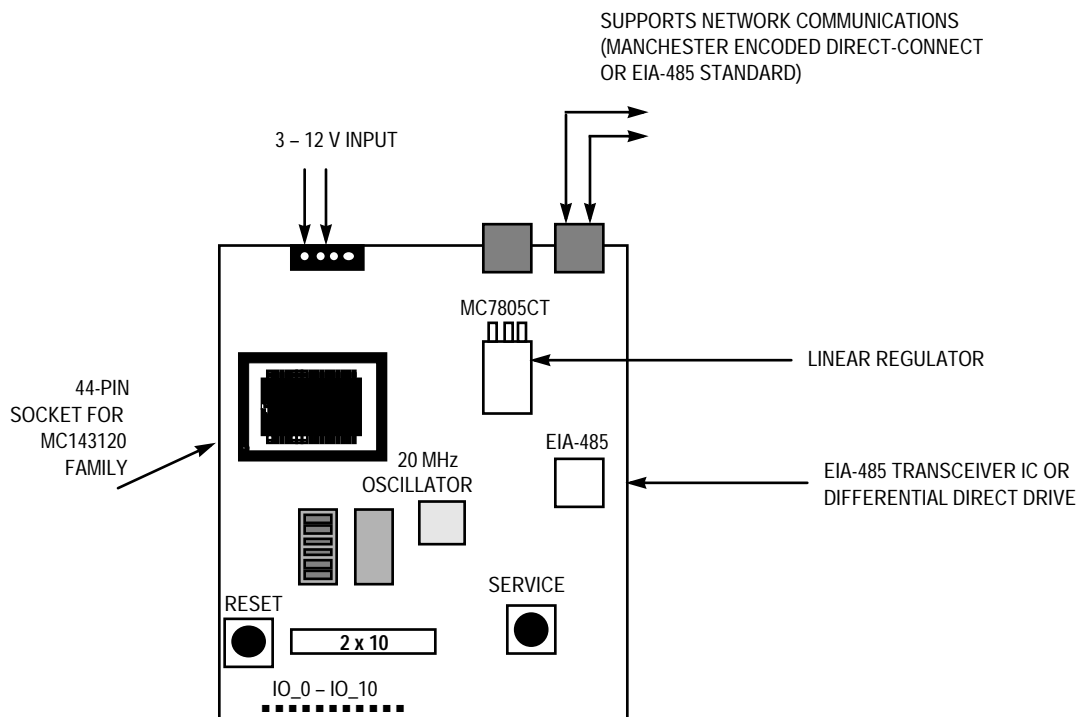
Neuron Chip Custom Node Development Board with 44-Pin Socket

The Motorola M143120FBEVK is intended for LONWORKS product developers that need to program and test application hardware and software in an actual networking environment. It provides the next step after emulation in application hardware-software debug before completing a final LONWORKS product design. A LonBuilder Developer's Workbench is required to develop application software (Neuron C), and it or some other Network Management Device can be used to install, configure, or reconfigure custom node boards over the communications network. Each board has a socket for an MC143120 Neuron IC which will contain its own unique serial number. Easy access to the Neuron Chip's 11 I/O pins and 5 communications port pins is provided via header pins and connectors. Application specific hardware can be easily bread-boarded and software-hardware interaction quickly evaluated, thereby eliminating much of the prototyping phase of product development.

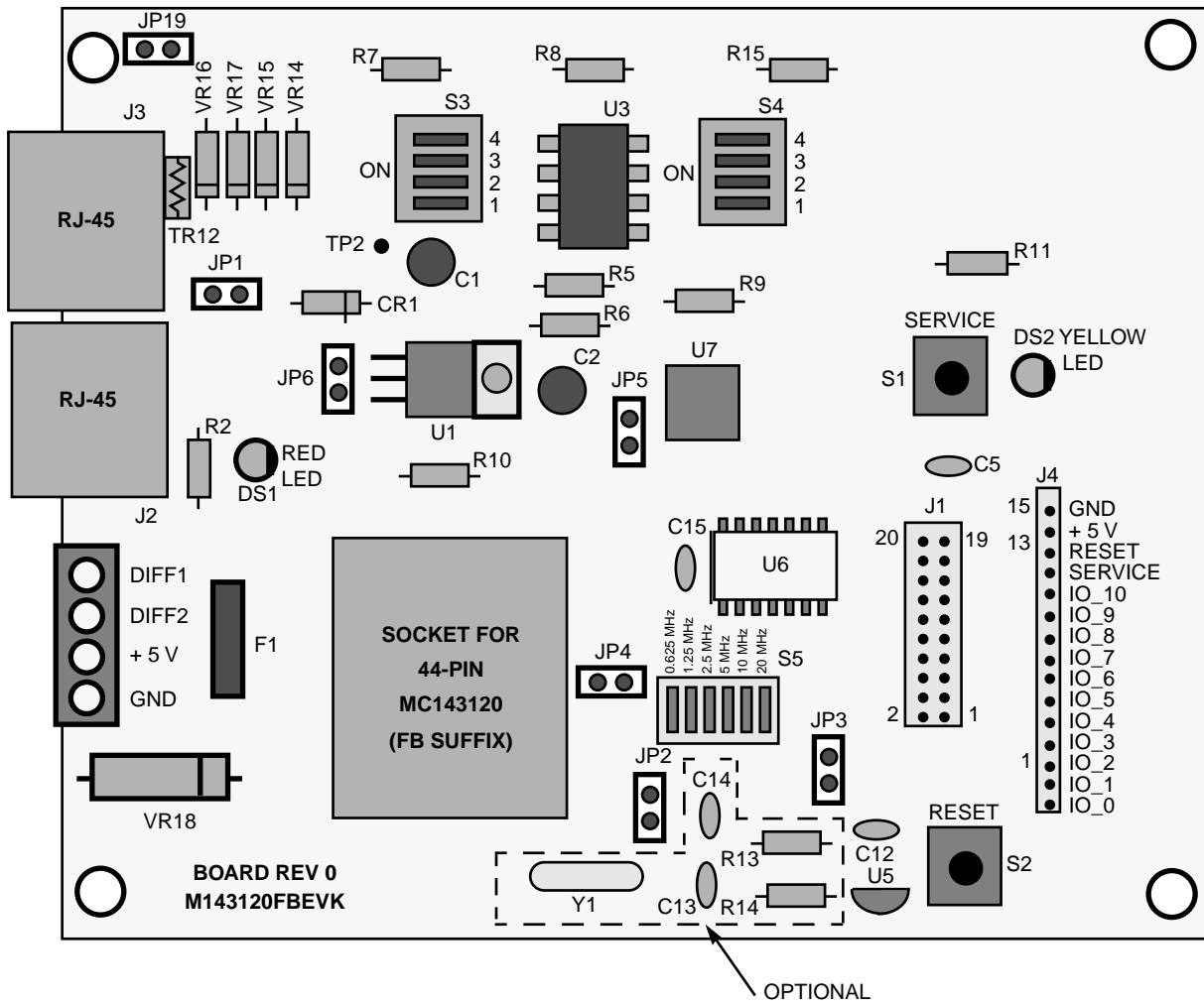
The M143120FBEVK is shipped with a simple scanning application. After power up, the MC143120E2DW will sequence output pins IO_0 to IO_10 and back again. This can be used to verify that the Neuron IC is working.

The M143120FBEVK can be used to program MC143120 devices without a LonBuilder tool. This is explained in application note AN1251 in the *LONWORKS Technology Data Book*, DL159/D.

- 44-Pin Socket for Any of the 3120 Neuron IC Devices (FB Suffix)
- Can Be Used to Program 3120 Devices from Either a LonBuilder Tool or an M143150EVK
- 20 MHz Oscillator with Selectable Time Base: 20 MHz, 10 MHz, 5 MHz, 2.5 MHz, 1.25 MHz, and 625 kHz
- On-Board Linear Regulator for Wide Power Supply Input; 8 to 16 V
- Configurable Between Direct-Connect and EIA-485 Differential Communications
- Screw-Down Terminal Blocks and Two RJ-45 Connectors for Easy Wiring Connection
- Reset and Service Pin Push-Button Switches
- Power LED Indicator and Service Pin Status LED Indicator
- 2 x 10 Connector for Interface from Neuron IC I/O Pins to Standard I/O Cable
- Can Operate Down to 2.7 V for Testing MC143120LE2
- Can Operate @ 20 MHz for Testing the MC143120FE2DW



M143120FBEVK ASSEMBLY DIAGRAM



M143150EVK

Neuron Chip Custom Node Development Board

The Motorola M143150EVK is intended for LONWORKS product developers that need to program and test application hardware and software in an actual networking environment. It provides the next step after emulation in application hardware-software debug before completing a final LONWORKS product design. A LonBuilder Developer's Workbench is required to develop application software (Neuron C) and it or some other Network Management Device can be used to install, configure, or reconfigure custom node boards over the communications network. Each board has a socket for an MC143150 Neuron IC which will contain its own unique serial number. Easy access to the Neuron Chip's 11 I/O pins and 5 communications port pins is provided via header pins and connectors. Application specific hardware can be easily bread-boarded and software-hardware interaction quickly evaluated, thereby eliminating much of the prototyping phase of product development.

The M143150EVK is shipped with a simple scanning application. After power up, the MC143150B1FU1 will sequence the output pins IO_0 to IO_10 and back again. This can be used to verify that the Neuron IC is working.

- 64-Pin Socket for Any of the 3150 Neuron IC Devices
- 29C256 32K Flash Memory for Ease of Programming
- Can Be Used to Program 3150 Devices from Either a LonBuilder or Other Network Manager Tool
- 20 MHz Oscillator with Selectable Time Base: 10 MHz, 5 MHz, 2.5 MHz, 1.25 MHz, and 625 kHz
- On-Board Linear Regulator for Wide Power Supply Input; 8 to 16 V
- Configurable Between Direct-Connect and EIA-485 Differential Communications
- Screw-Down Terminal Blocks and Two RJ-45 Connectors for Easy Wiring Connection
- Reset and Service Pin Push-Button Switches
- Power LED Indicator and Service Pin Status LED Indicator
- 2 x 10 Connector for Interface from Neuron I/O Pins to Standard I/O Cable

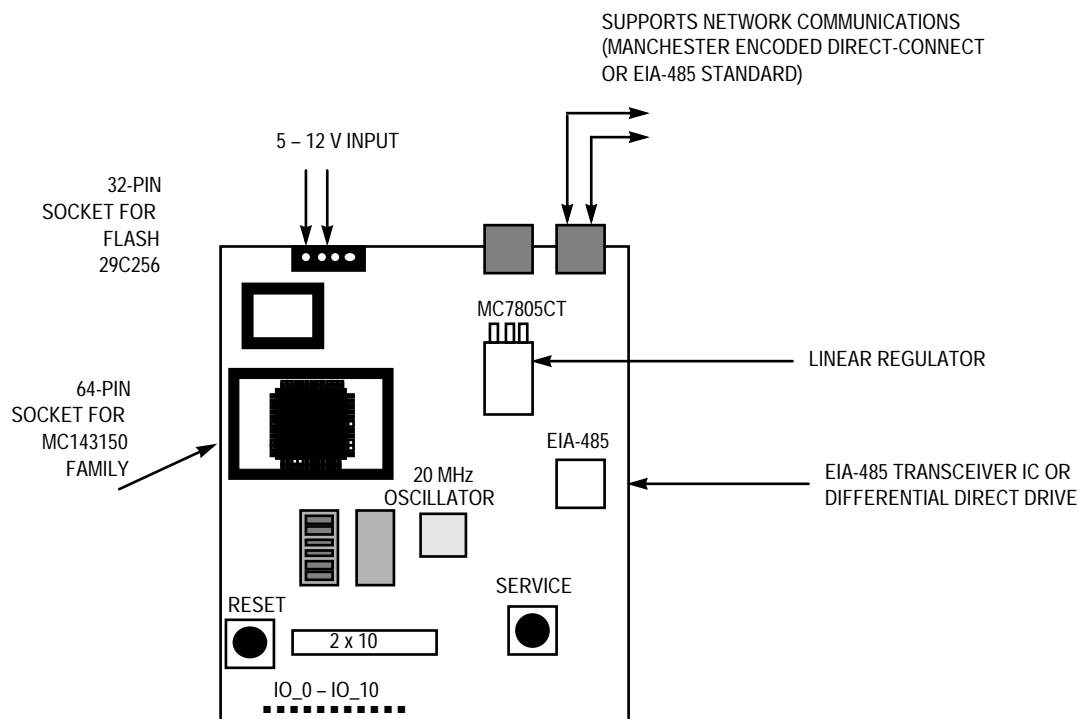
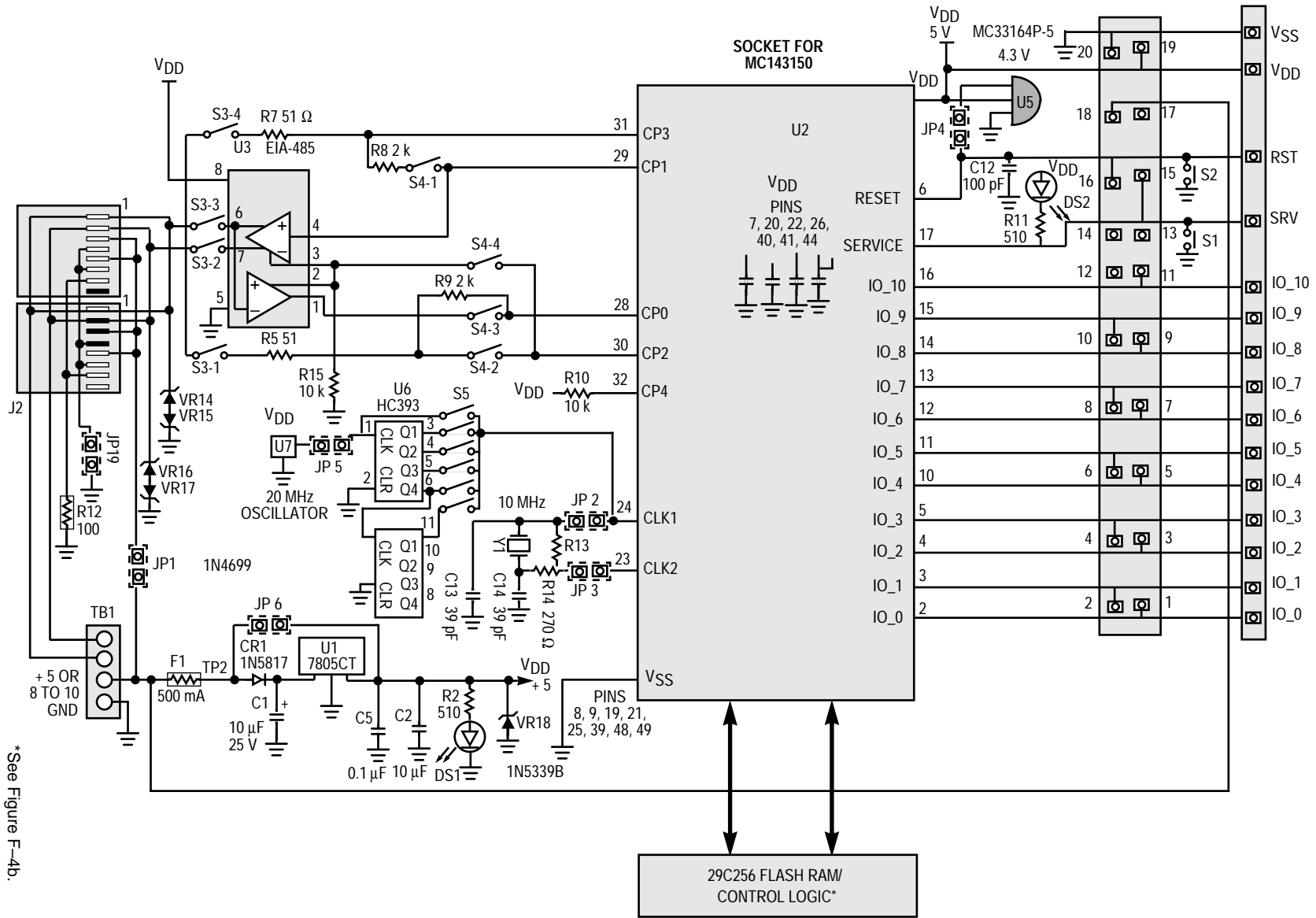


Figure F-4a. M143150EVK Schematic Diagram



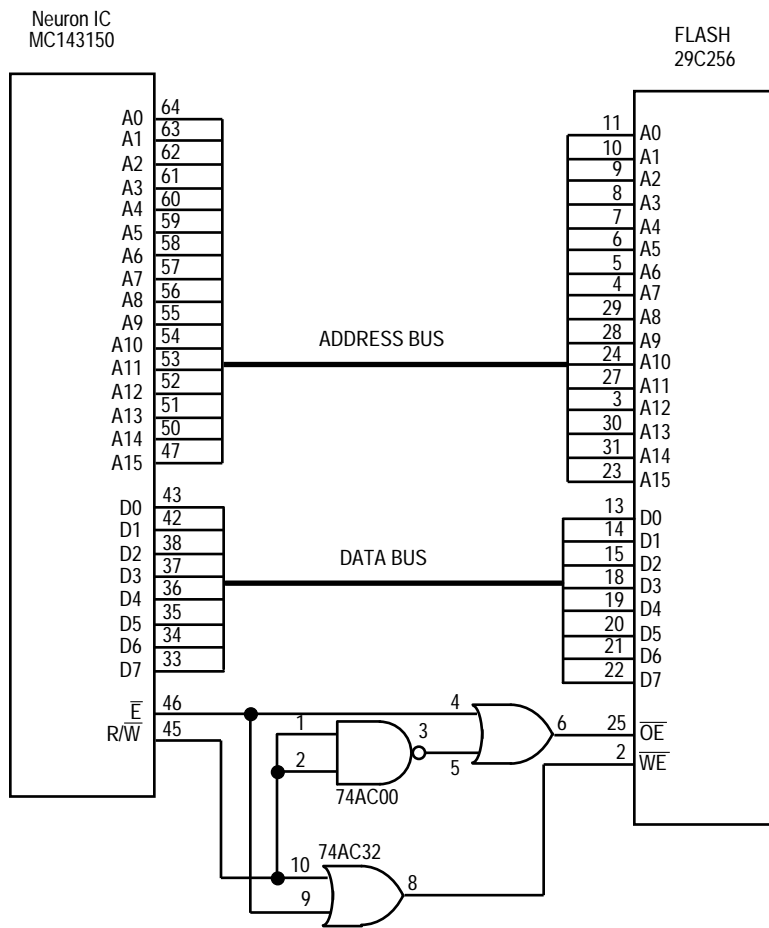
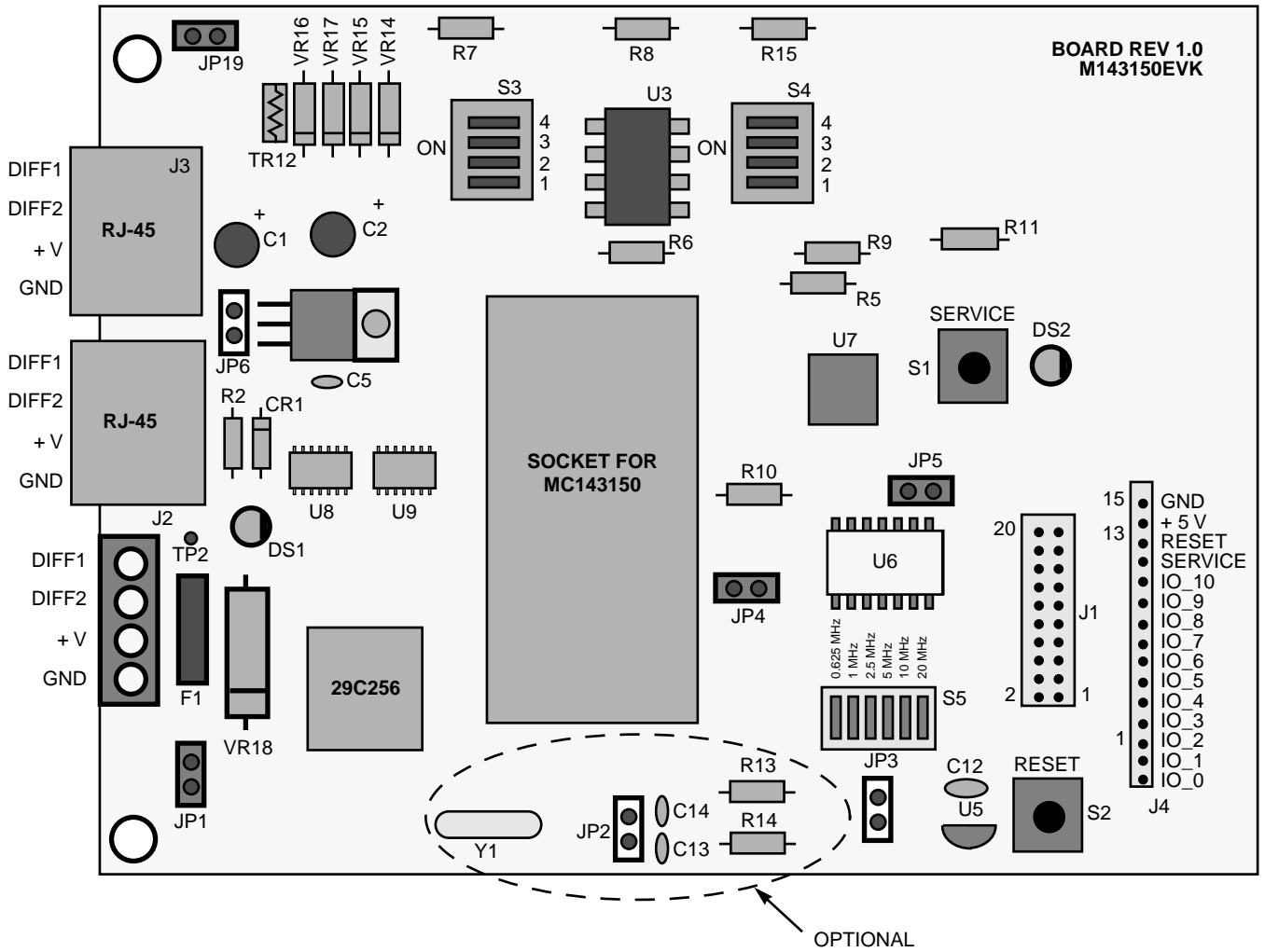


Figure F-4b.

M143150EVK ASSEMBLY DIAGRAM



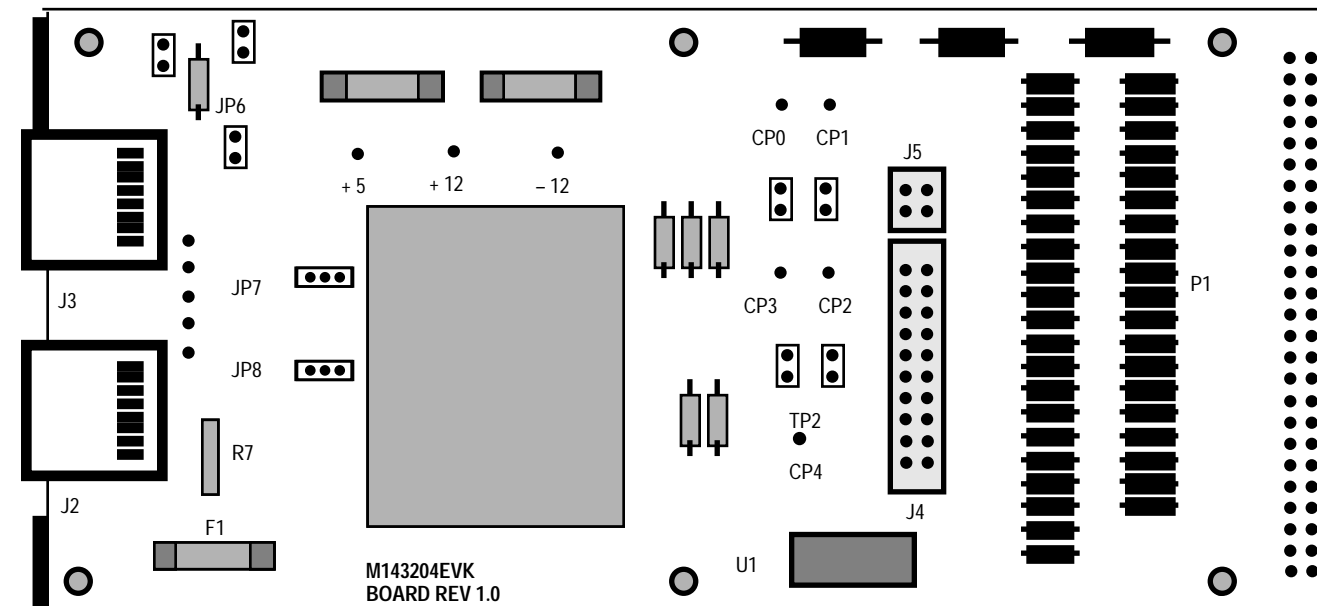
M143204EVK

LonBuilder Direct-Connect Transceiver Board

Motorola's Direct-Connect Transceiver Board (M143204EVK) is designed to be used as a standard transceiver in the available slot of a development board in the LonBuilder Developer's Workbench. It provides a differential direct connection to a network via two RJ-45 connectors. In the differential mode, the Neuron IC built-in transceiver is able to differentially drive and sense a twisted-pair transmission line. One direct-drive transceiver board can be used on a LonBuilder tool to bring out the LonBuilder backplane which connects all the Neuron IC Emulator boards together. The direct-connect board can then be interfaced to off-board nodes with either direct-drive, RS-485, or transformer-coupled transceivers. This is possible because all three media transceivers use two-wire manchester coding. With a direct drive, transceivers can easily test how nodes will behave with different network speeds.

The M143204EVK board can also be used to gain access to the Neuron IC I/O pins. This is achieved if the M143204EVK is placed in the lower slot on the LonBuilder Emulator card. A standard 2 x 10 connector gives access to all the pins.

- Two RJ-45 Connectors on the Front Panel Provide an Extension of the LonBuilder Developer's Workbench's Differential Backplane via Transceiver Slot (P2) of Any LonBuilder Development Board
- 2 x 10 Connector Allows Access to the Neuron Chip's I/O Pins (Compatible to Motorola's and Echelon's Gizmo Boxes) via Slots P2 or P3 on the LonBuilder's Emulator or SBC Boards
- Prototype Area, Jumpers, Test Points, and Connectors Allow for Convenient Modifications, Building of Prototypes, Running of Experiments, and Testing
- Fuses for + 5, + 12, and - 12 V Operation Through the RJ-45 Connectors (The LonBuilder Developer's Workstation Supports Currents at these Voltages up to 400, 35, and 35 mA, Respectively)
- Capability of Providing Power to Other Nodes through the RJ-45 Connector: a Jumper Selects + 5 or + 12 V (or None) to Pins 3 or 5 of the RJ-45 Connector
- Network Termination Resistor May be Jumped In or Out
- Support for Differential Mode Transceivers at Distances Less than 15 Meters



REV 3
2/97

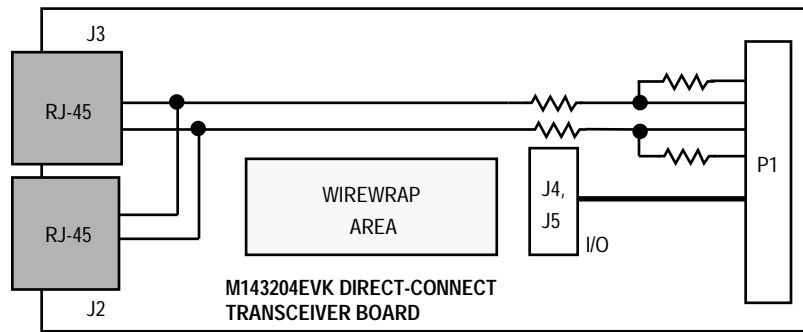


Figure F-5. Direct-Connect Transceiver Board

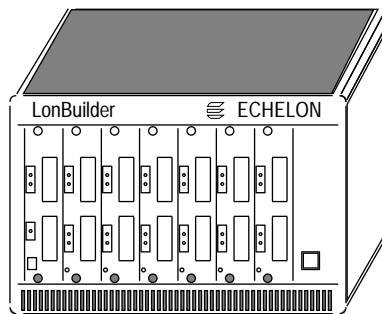


Figure F-6. LonBuilder Developer's Workbench

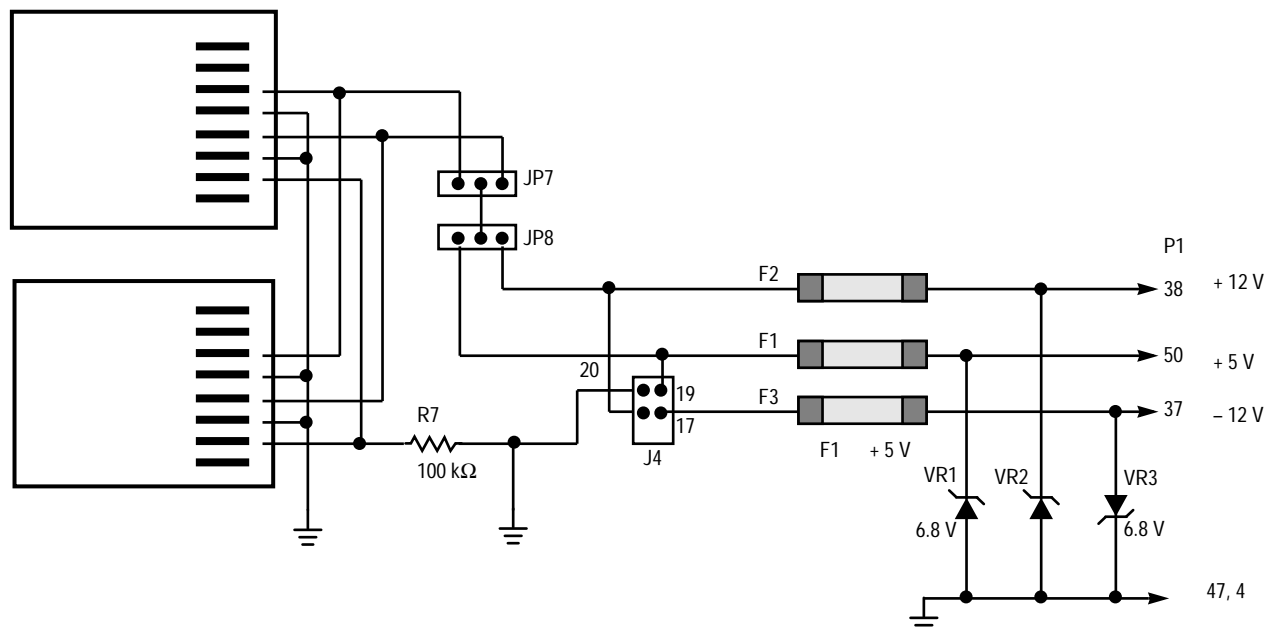
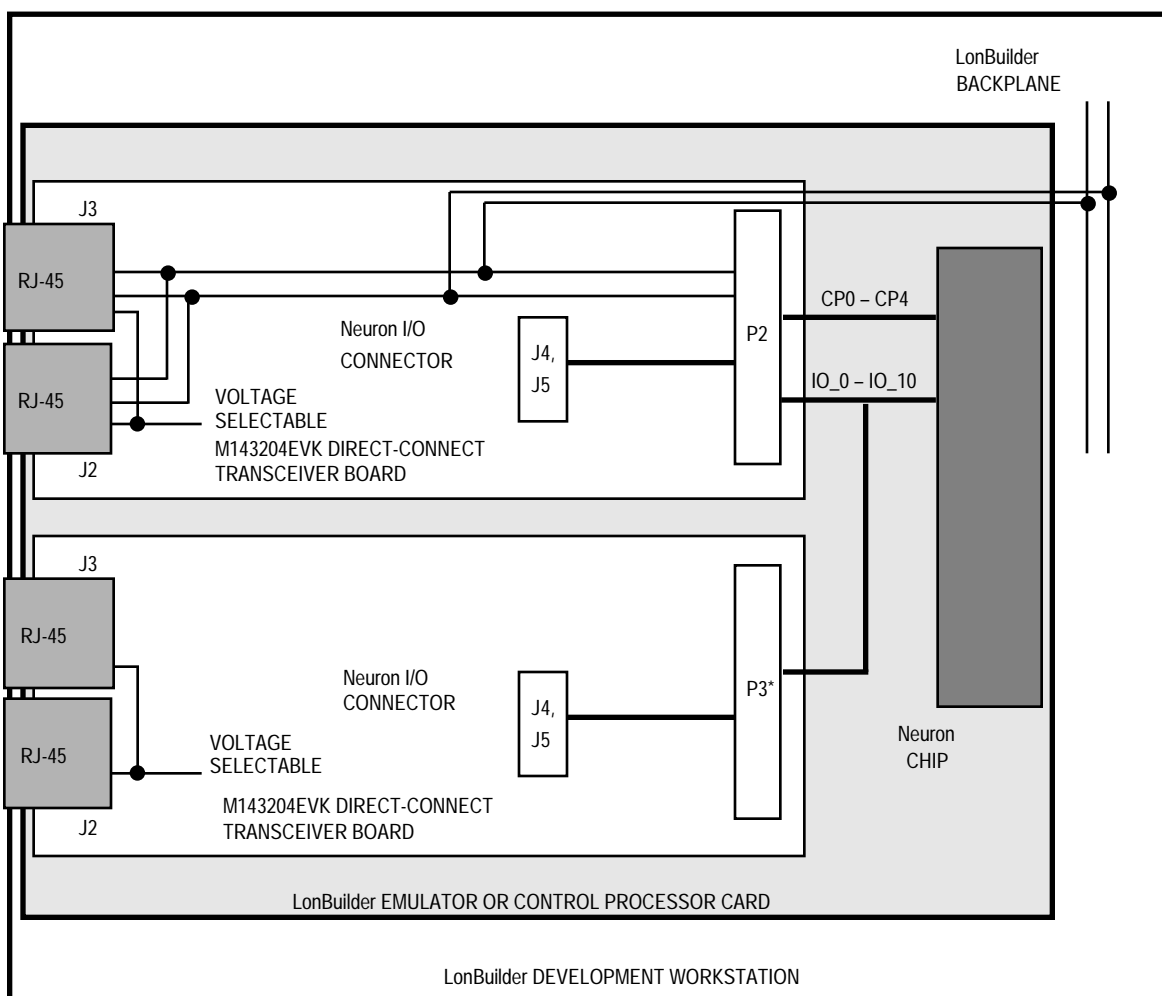
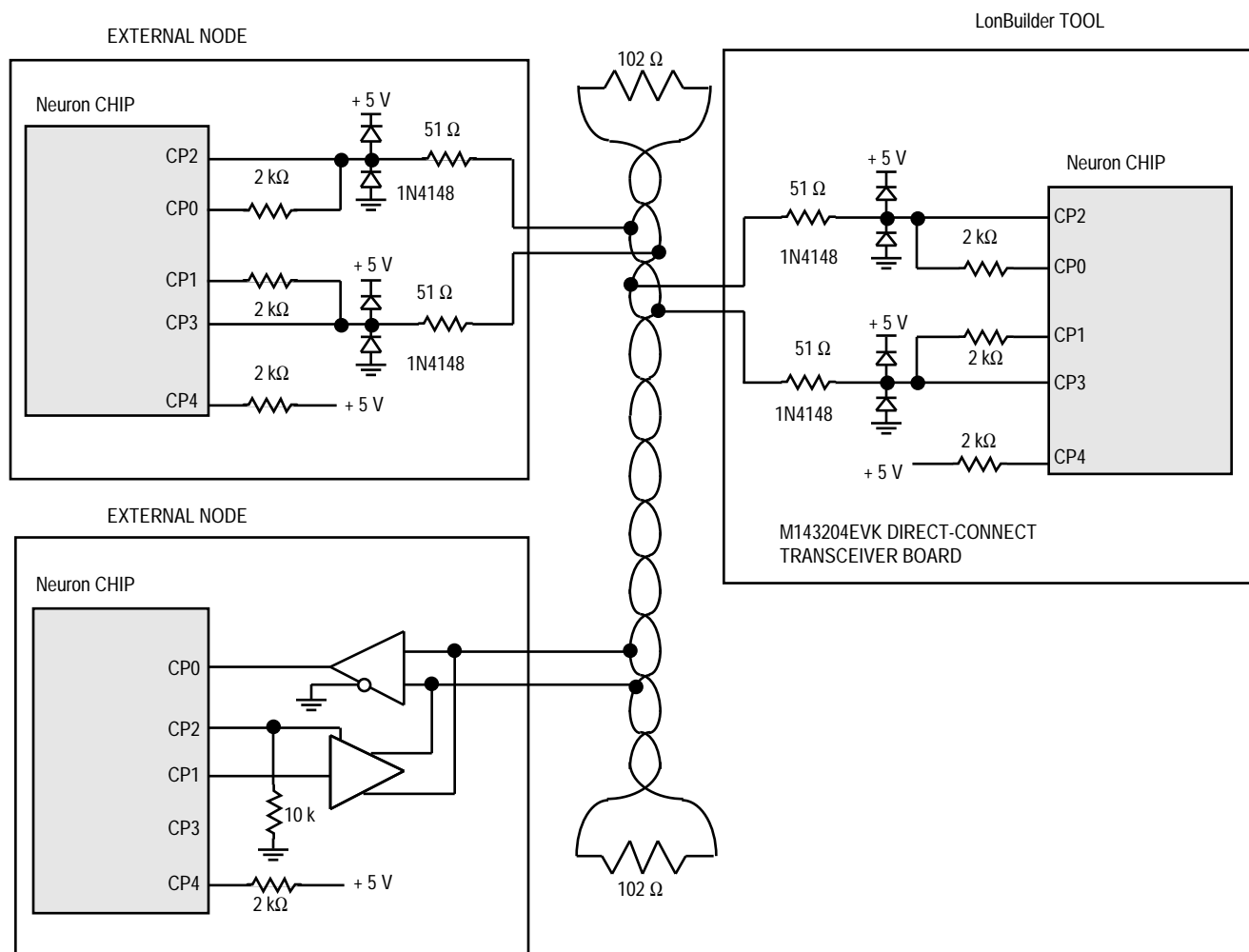


Figure F-7. Pinout



*No communication lines available from P3.

Figure F-8. Connecting to a LonBuilder Developer's Workbench



NOTE: Direct-Drive Card can interface to all twisted-pair transceivers.

Figure F–9. Connecting to a Custom Node

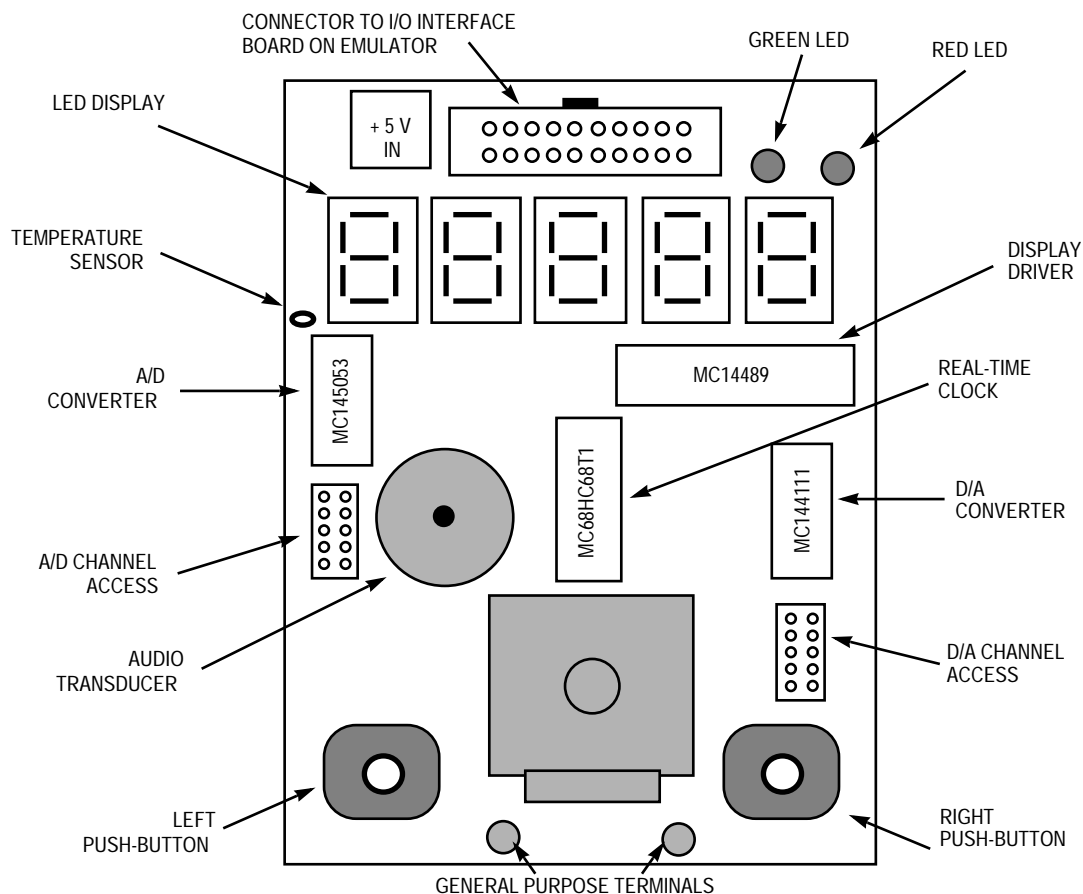
M143206EVK

Neuron Chip Gizmo 3 I/O Interface Board

The M143206EVK I/O Interface Application Board (Gizmo 3) can be used for experimenting with the Neuron IC, as part of a training course, or for quick development of a demonstration project. The board contains a variety of ICs that are accessible through a 2 x 10 connector that is pin compatible to Motorola's Neuron IC development boards or the LonBuilder Developer's Workbench I/O cards. Included with the kit is Neuron C application software for each of the ICs to aid in quick product development.

The M143206EVK (Gizmo 3) has an assortment of devices that can interface to the I/O port of a Neuron IC. Features include:

- MC144111 6-Bit, 4-Channel, Digital-to-Analog Converter IC
- MC145053 10-Bit, 5-Channel, Analog-to-Digital Converter IC
- MC68HC68T1 Real-Time Clock IC
- MC14489 Display Driver with a 5-Digit LED Display
- Digital Shaft Encoder
- Piezoelectric Buzzer
- LM34 Fahrenheit Temperature Sensor
- Two Push-Button Switches
- Two LEDs
- Pin-for-Pin Compatibility with Echelon's Gizmo Box
- Neuron C Software to Sense and Control the Hardware



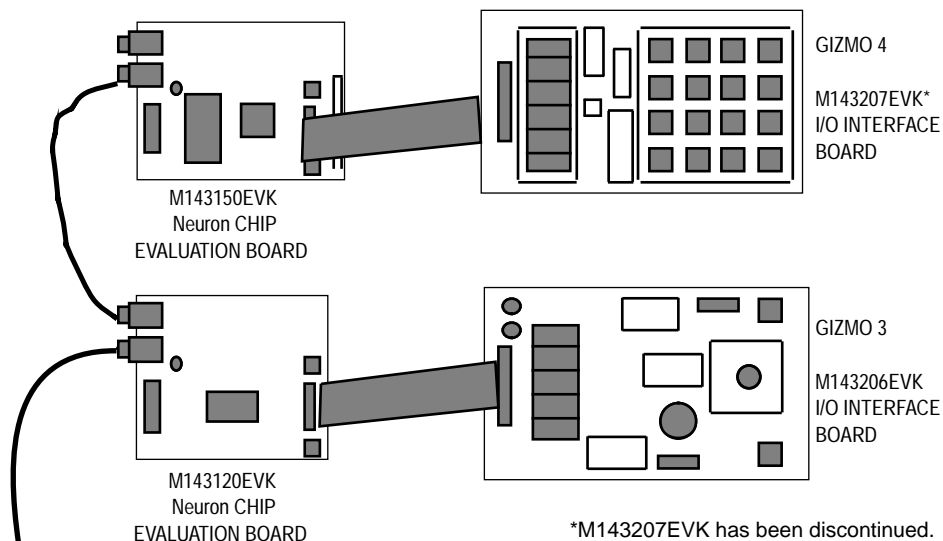


Figure F-10. LonWorks Development Tools

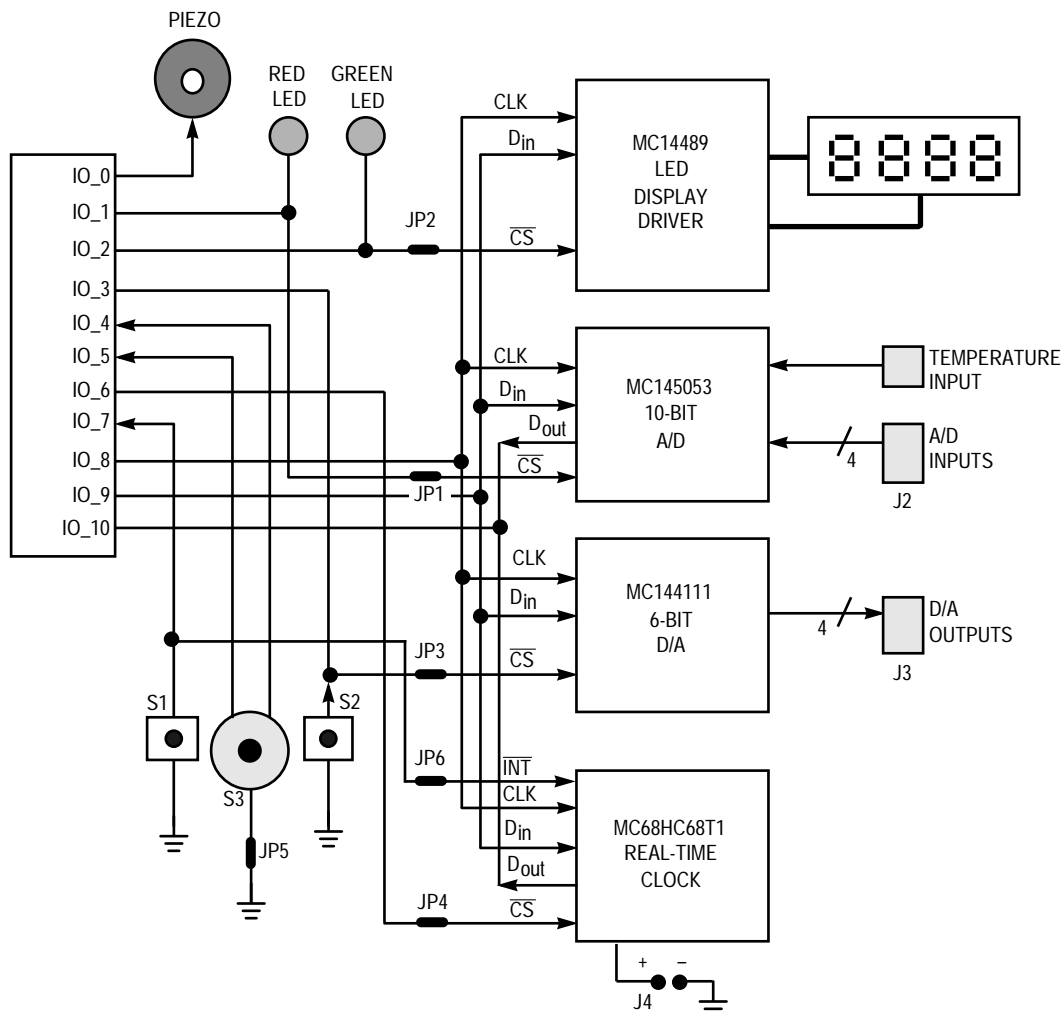


Figure F-11. M143206EVK Functional Diagram (Gizmo 3)

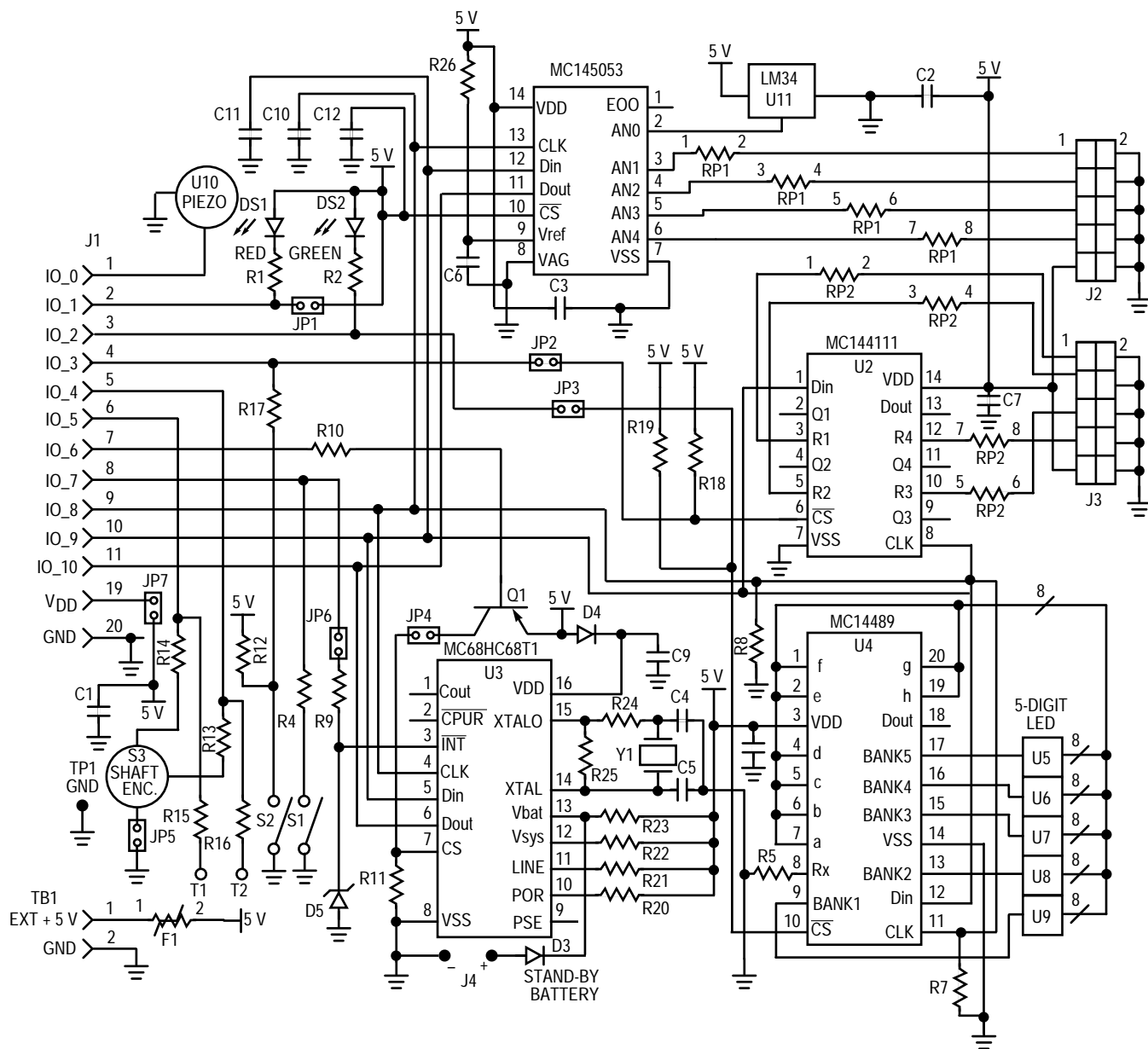


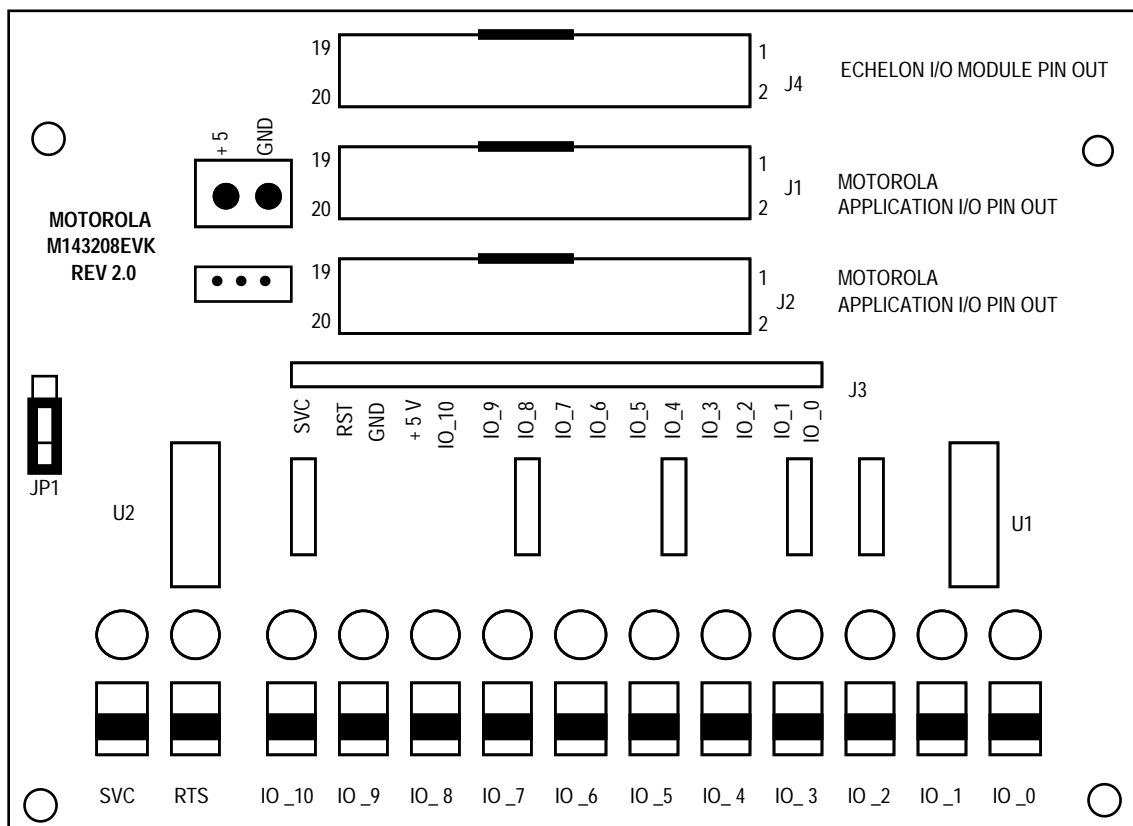
Figure F-12. Schematic Diagram of M143206EVK (Gizmo 3) – Rev. 2.5

M143208EVK

I/O Interface Test Board

The M143208EVK I/O Interface Test Board (Gizmo 5) offers the user the ability to test all I/O pins of the Neuron IC. The board can be configured as either an input or output interface. When configured as an input, Gizmo 5 can be used to visually monitor the I/O lines of the Neuron IC using the LEDs. Like the other Gizmo boards, the M143208EVK can be used for experimenting with the Neuron IC, as part of a training course, or for quick development of a demonstration project. The board contains two 2 x 10 connectors that are pin compatible to Motorola's Neuron IC development boards or the LonBuilder Developer's Workbench I/O cards. Features of Gizmo 5 include:

- Can Interface to Either Motorola's Neuron IC EVBUs or Echelon's Modules
- 2 x 10 Connector for Interface Through Ribbon Cable
- 11 LEDs for Monitoring the 11 Neuron IC I/O Pins
- 11 Slide Switches for Pull Down Use When Neuron I/O Pins Are Configured as Inputs
- Terminal Block for External + 5 V Supply
- Jumper to Disable LED
- LED and Slide Switch Connecting to the Neuron IC's Reset Pin
- LED and Slide Switch Connecting to the Neuron IC's Service Pin
- Two 2 x 10 Header Connectors for Single or Inline Connection to the Neuron IC



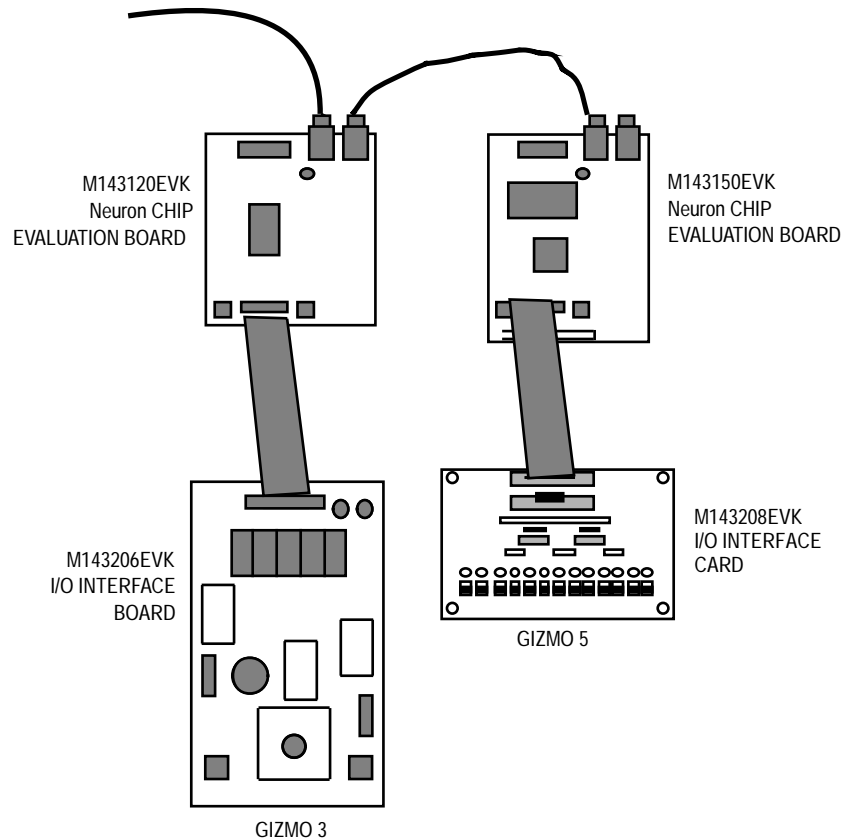
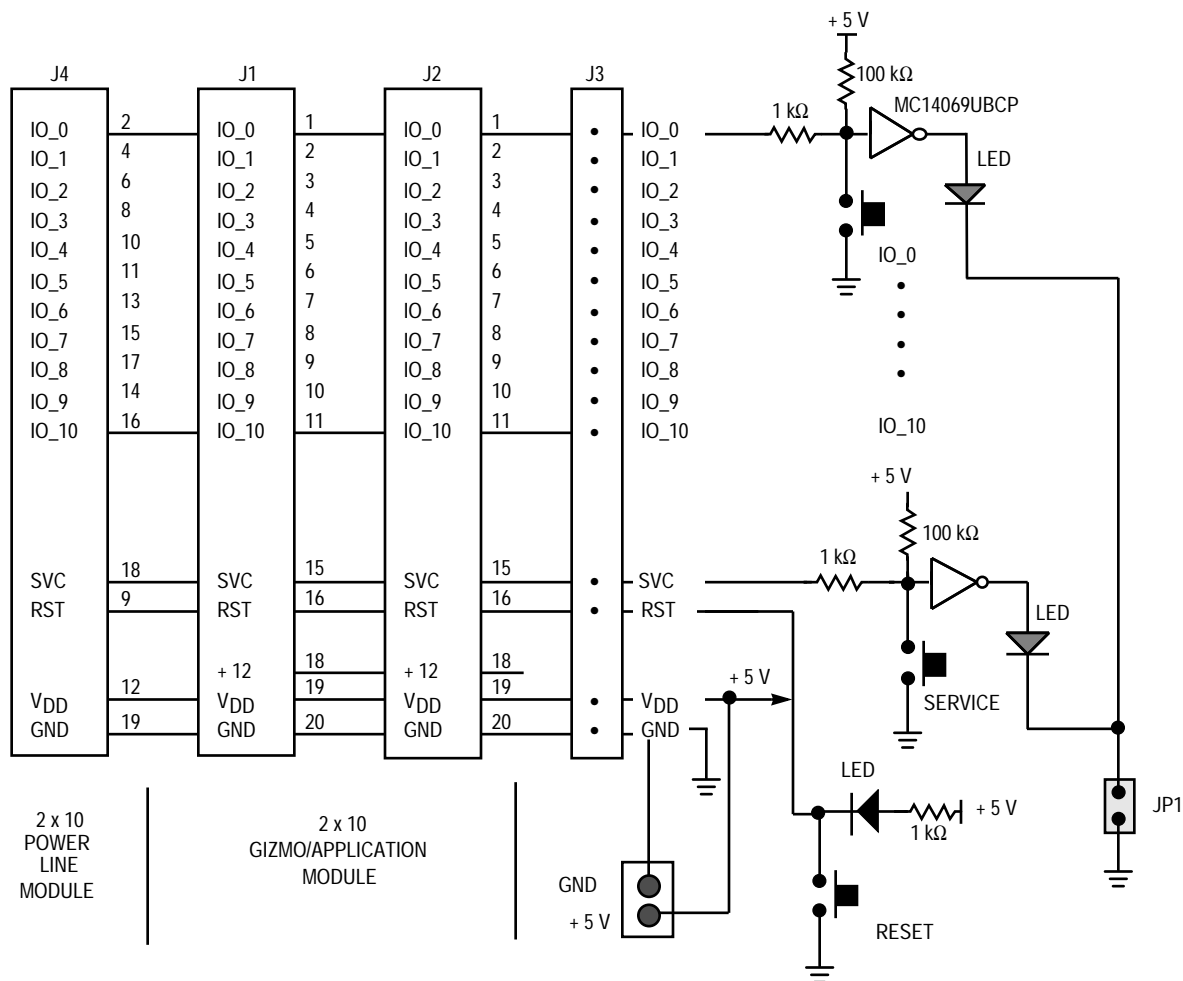


Figure F-13. LonWorks Development Tools



NOTES:

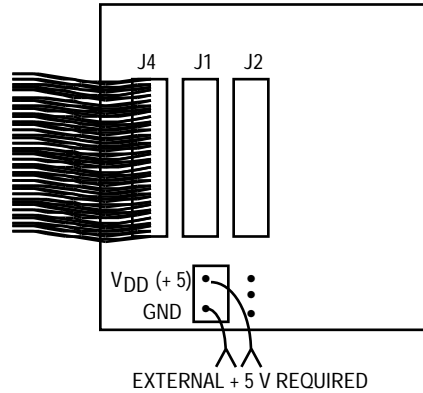
1. Connectors J1 and J2 are compatible with both Echelon's Gizmo application kit or any of Motorola's support tool kits. Connectors J1 and J2 make possible the monitoring of the I/O pins between a Neuron EVK and an application kit.
2. Connector J4 is capable of interfacing to Echelon's modules, but consideration should be taken relative to the power source. Echelon's power line module (PLC-10) has a compatible 2 x 10 connector, but does not have a + 5 V output. An external + 5 V supply should be connected to the terminal block on the M143208 board. When connecting to Echelon's TP or FT modules, an external + 5 V supply is also needed, but in this case the TP or FT module will be powered from J4 over the ribbon cable.

CABLE 2 x 9

TP/FT(78/1250)		J4	
IO_0	2	2	IO_0
IO_1	4	4	IO_1
IO_2	6	6	IO_2
IO_3	8	8	IO_3
IO_4	10	10	IO_4
IO_5	11	11	IO_5
IO_6	13	13	IO_6
IO_7	15	15	IO_7
IO_8	17	17	IO_8
IO_9	14	14	IO_9
IO_10	16	16	IO_10
SVC	18	18	SVC
RST	9	9	RST
V _{DD}	12	1	V _{DD}
GND	3, 5, 7	3, 5, 7, 19	GND

5 V @ 80 mA

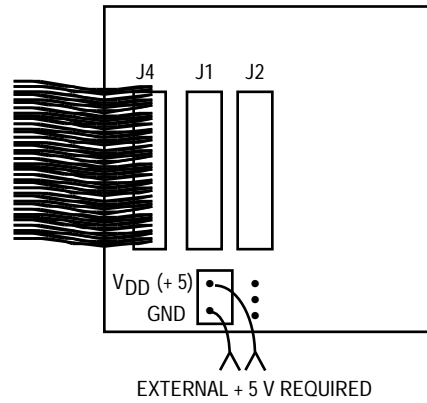
GND



CABLE 2 x 10

PLC10		J4	
IO_0	2	2	IO_0
IO_1	4	4	IO_1
IO_2	6	6	IO_2
IO_3	8	8	IO_3
IO_4	10	10	IO_4
IO_5	11	11	IO_5
IO_6	13	13	IO_6
IO_7	15	15	IO_7
IO_8	17	17	IO_8
IO_9	14	14	IO_9
IO_10	16	16	IO_10
SVC	18	18	SVC
RST	9	9	RST
+9	1	1	
NC	12	12	V _{DD}
GND	3, 5, 7	3, 5, 7, 19	GND

GND

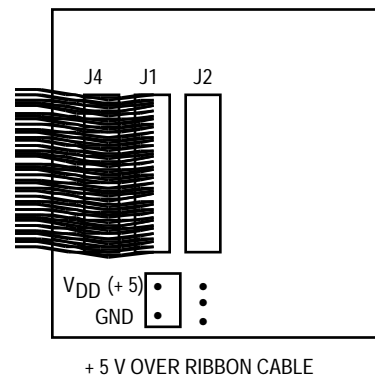


CABLE 2 x 10

MOTOROLA KITS		J1 OR J2	
IO_0	1	1	IO_0
IO_1	2	2	IO_1
IO_2	3	3	IO_2
IO_3	4	4	IO_3
IO_4	5	5	IO_4
IO_5	6	6	IO_5
IO_6	7	7	IO_6
IO_7	8	8	IO_7
IO_8	9	9	IO_8
IO_9	10	10	IO_9
IO_10	11	11	IO_10
SVC	15	15	SVC
RST	16	16	RST
+12	18	18	+12
V _{DD}	19	19	V _{DD}
GND	20	20	GND

5 V

GND

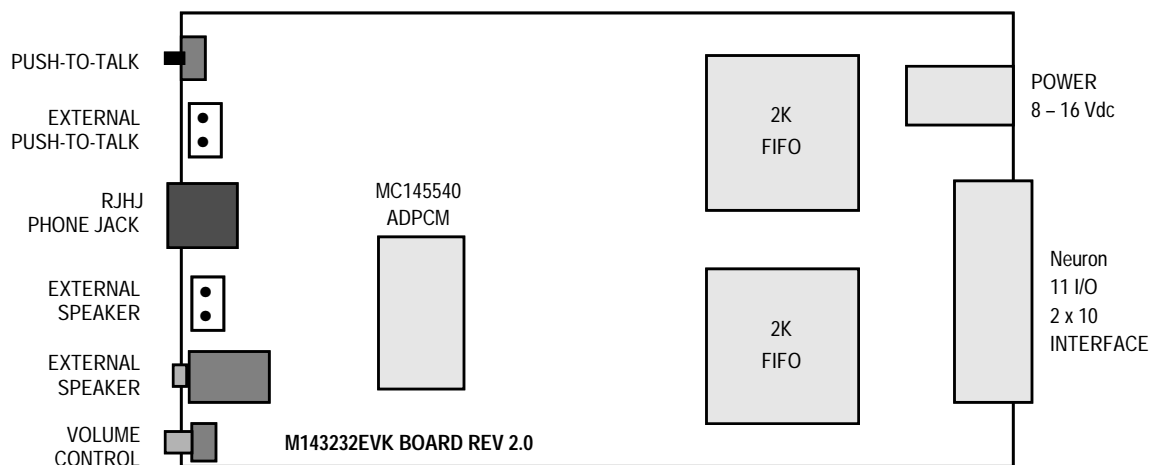


M143232EVK

Voice Compression Board for LonWorks Networks

The possibility of transmitting voice over a LONWORKS distributed control network lends itself to a variety of product opportunities. These range from intercoms, PA systems, voice storage playback systems, and simple phones (see Figure F-14).

- Provides for Digitized Voice Transmission and Storage on LONWORKS Networks
- On-Board 16 kbps ADPCM Codec with Handset and Amplifier Circuitry
- On-Board Linear Regulator for Wide Power Supply Input; 8 to 16 Vdc
- Built-In Self Test Mode
- 20-Pin Connector for Access to LonBuilder or Neuron Development Boards
- Push-to-Talk or Optional Voice-Activated-Switch (VOX) Circuitry
- Power LED Indicator



In a PA application, voice can be transmitted to any speaker on the network with the option of easily selecting which receiving speaker would receive the voice. In addition, a microphone could be installed anywhere on the network, lending itself to a very flexible system. In an intercom application, each intercom has a push-to-talk switch and a keyboard to easily select individual or groups of receiving intercoms. Speakers could also be part of this network. Prerecorded messages could be stored and played back to any one of the output speakers. For example, in an apartment building there could be numerous entry phones used to buzz the appropriate apartments. The applications are numerous.

The M143232EVK Voice Demo Board is designed for use with LonBuilder tools, NodeBuilder tools, or Neuron IC develop-

ment boards. The board provides the user with the means to implement voice communications using the LonTalk® communications protocol over a LONWORKS technology network. The M143232EVK has three modes of operation; two for normal operation and a third as a test mode.

The Voice Demo Board interfaces to either an MC143120 or MC143150 Neuron IC to perform the voice function. Voice is compressed at a rate of 16 kbps through a Motorola MC145540 ADPCM codec IC, then is output serially at a 64 kbps rate into a 2K x 1 FIFO, and then parallel loaded into the Neuron IC for transmission over the LONWORKS network. When operating the compressed voice system on a 78 kbps LONWORKS network, the voice packets take up only 30% of the bandwidth. On a 1.25 Mbps network, the packets' bandwidth is reduced to 5%.

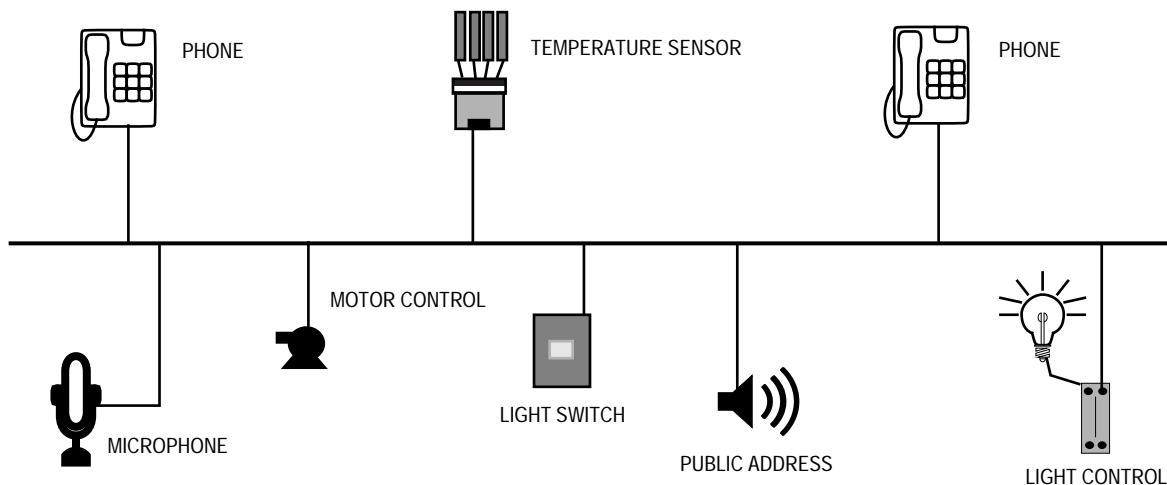


Figure F-14. Voice on a Distributed-Control Network in Homes, Offices, and Factories

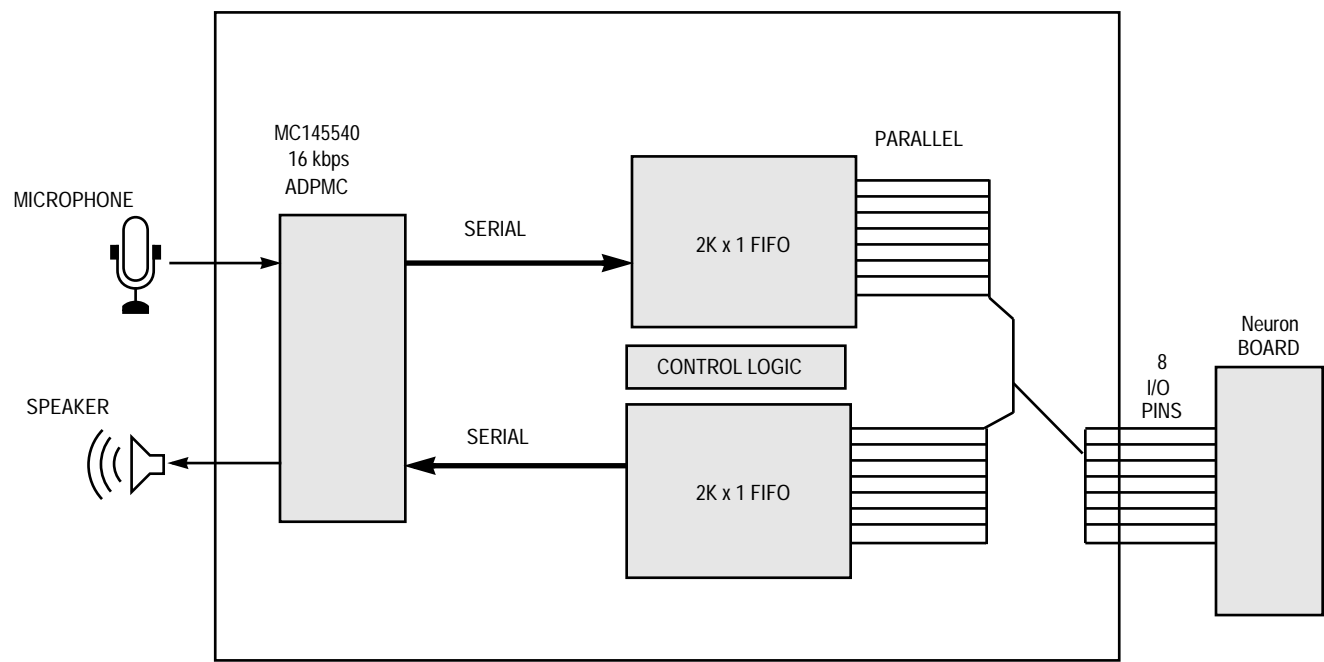


Figure F-15.

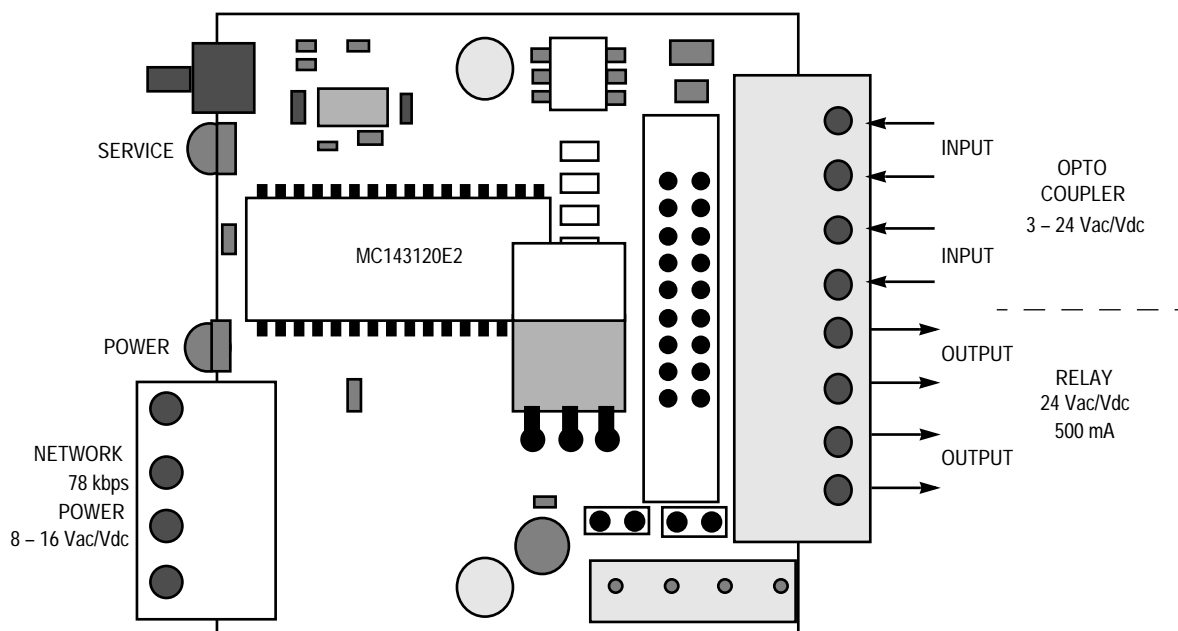
M143235EVK

Advance Information

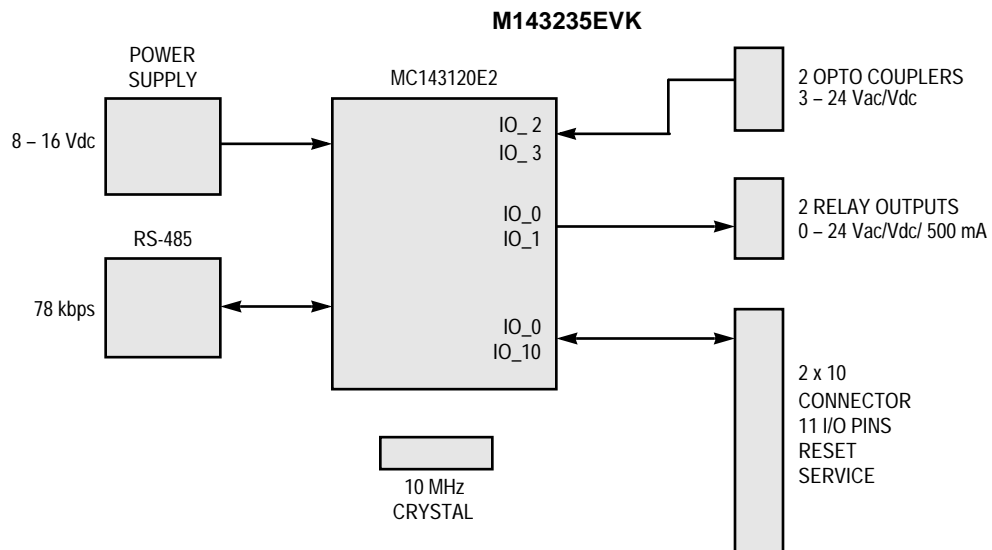
Neuron Chip Custom Node Development Board

The Motorola M143235EVK is intended for LONWORKS product developers that need to program and test application hardware and software in an actual networking environment. It provides the next step after emulation in application hardware-software debug before completing a final LONWORKS product design. A LonBuilder Developer's Workbench or NodeBuilder is required to develop application software (Neuron C), and it or some other Network Management Device can be used to install, configure, or reconfigure custom node boards over the communications network. Each board contains an MC143120E2DW Neuron IC which will contain its own unique serial number. Easy access to the Neuron Chip's 11 I/O pins and 5 communications port pins is provided via header pins and connectors. Application specific hardware can be easily bread-boarded and software-hardware interaction quickly evaluated, thereby eliminating much of the prototyping phase of product development.

- Contains an MC143120E2DW Neuron IC for Use in Developing Products
- On-Board 10 MHz Oscillator
- On-Board Linear Regulator for Wide Power Supply Input; 8 to 16 Vac/Vdc
- On-Board 78 kbps Transformer-Coupled Transceiver
- 20-Pin Connector for Access to All Neuron I/O, Service, Reset, and Power Pins
- Two Relay-Isolated Output Pins (IO_0, IO_1) 0 to 24 Vac/Vdc @ 500 mA
- Two Opto-Isolated Inputs Pins (IO_4, IO_5) 3 to 24 Vac/Vdc
- Service Push-Button and Indicator LED
- Power LED Indicator



This document contains information on a new product. Specifications and information herein are subject to change without notice.



Advance Information **Neuron[®] Chip LiteNode Development Boards**

MC143238EVK MC143239EVK MC143240EVK

The LiteNode family of products offers a low-cost entry into evaluating LONWORKS[®] technology. The initial product offering of I/O nodes and a network manager gateway make possible a variety of network configurations that can be used in many industry or market scenarios to aid in testing or evaluating LONWORKS technology. A library of downloadable application software is available on Motorola's web site (<http://motorola.com/lonworks>). These applications, or NEI files, are compiled specifically for the individual LiteNode products. Through the web page, Motorola provides a large variety of node-specific application files to choose from, depending on the system need. The NEI file can be downloaded from Motorola's web site to the PC, and then downloaded into the appropriate node through the network manager gateway. The list of application software will continue to grow because all users can upload NEI files for others to use.

When the appropriate application files are loaded into the different nodes in the system, they are then installed and bound to the PC. Using Visual Basic, a system application can be written to control the nodes. Different Visual Basic graphics illustrating home control, building control, industrial process control, and many others, will also be available for downloading from Motorola's web site. The network manager gateway does not offer a C compiler capability. For modifying an application file or creating new NEI files, there are numerous companies that offer C compilers, depending on the degree of development needs. Links to such companies can be found on both Motorola's and Echelon's web sites.

FEATURES

- Each Board Contains the 44-Pin MC143120E2 (FB Suffix) with 2K EEPROM and 2K RAM
- On-Board 10 MHz Oscillator
- On-Board Linear Regulator for Wide Power Supply Input; 10 to 16 Vdc
- On-Board EIA-485 Transceiver, Configured for 9.6 to 1250 kbps Data Rate
- Multi-Pin Connector for Access to I/O Pins (2 x 7)
- Service Push-Button and Indicator LED
- Family of Boards with a Variety of I/O Configurations
- Library of Applications Code on Motorola's Web Site

MC143238EVK (I/O NODE)

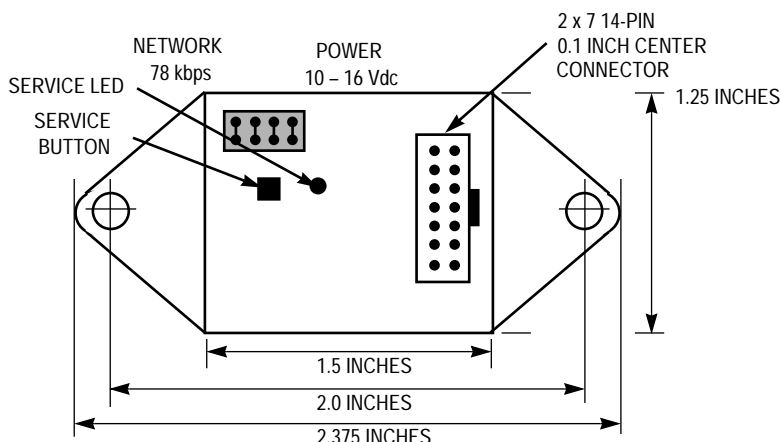
- Two Input Pins (IO_4, IO_5) 0 to 16 Vdc (On > 2.5 Vdc)
- Two Output Pins (IO_0, IO_1) 0 to 24 Vdc @ 200 mA

MC143239EVK (H-BRIDGE NODE)

- Two Input Pins (IO_4, IO_5) 0 to 16 Vdc (On > 2.5 Vdc)
- Two Output Pins from H-Bridge for Motor Drive 0 to 16 Vdc @ 500 mA

MC143240EVK (10-BIT A/D NODE)

- One Digital Input Pin (IO_4) 0 to 16 Vdc (On > 2.5 Vdc)
- One Digital Output Pin (IO_0) 0 to 24 Vdc @ 200 mA
- Four Analog Inputs (10-Bit A/D) 0 to 5 Vdc



REV 0.2
9/98

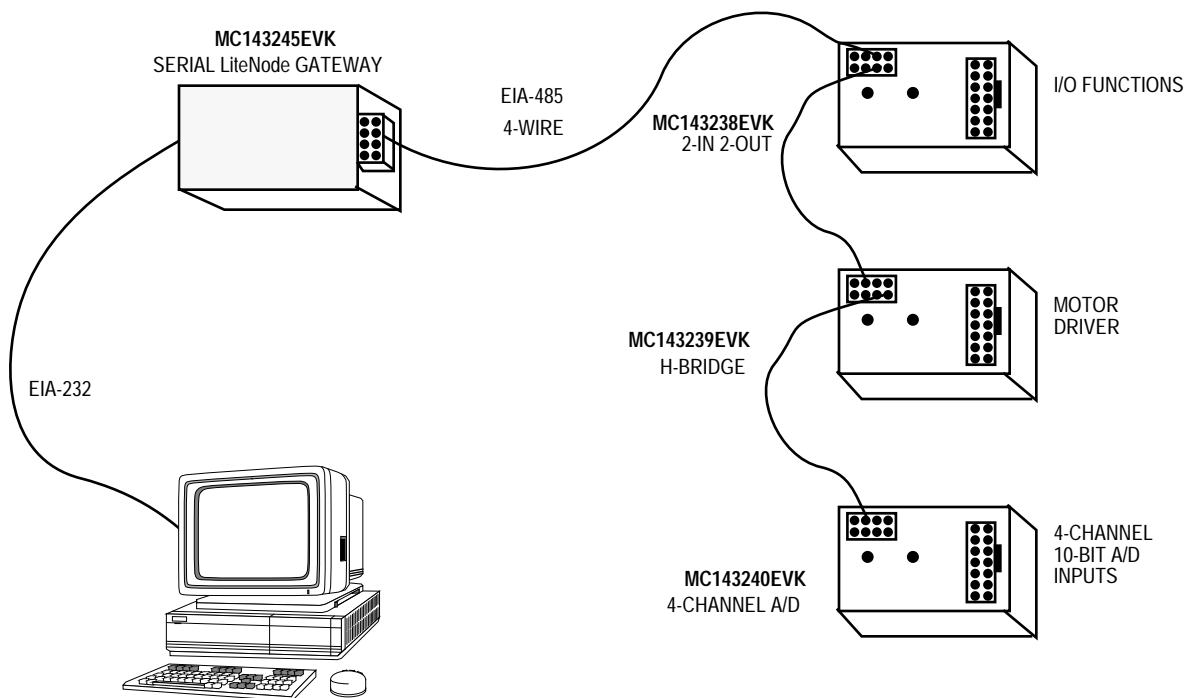


Figure F-16.

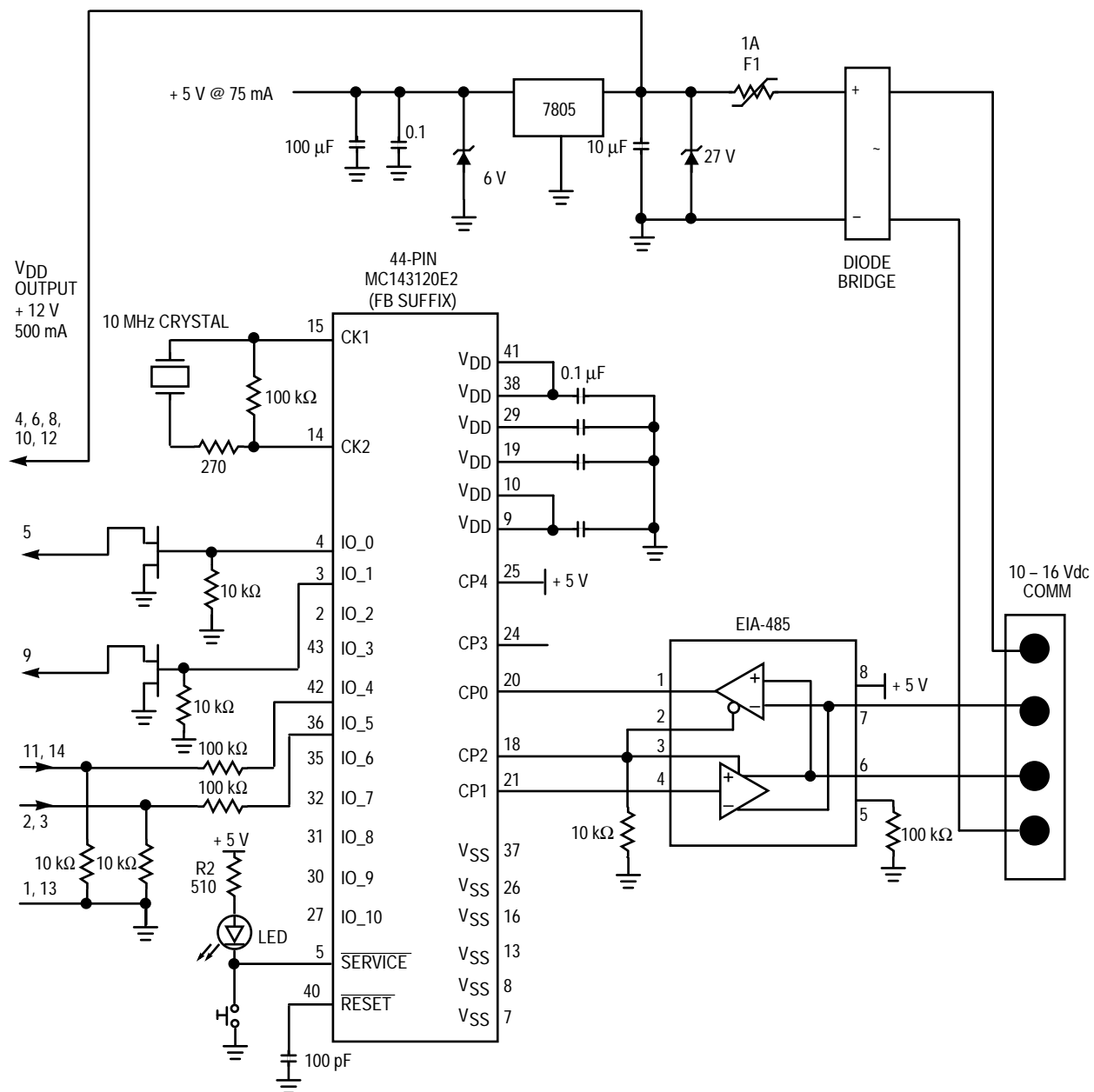
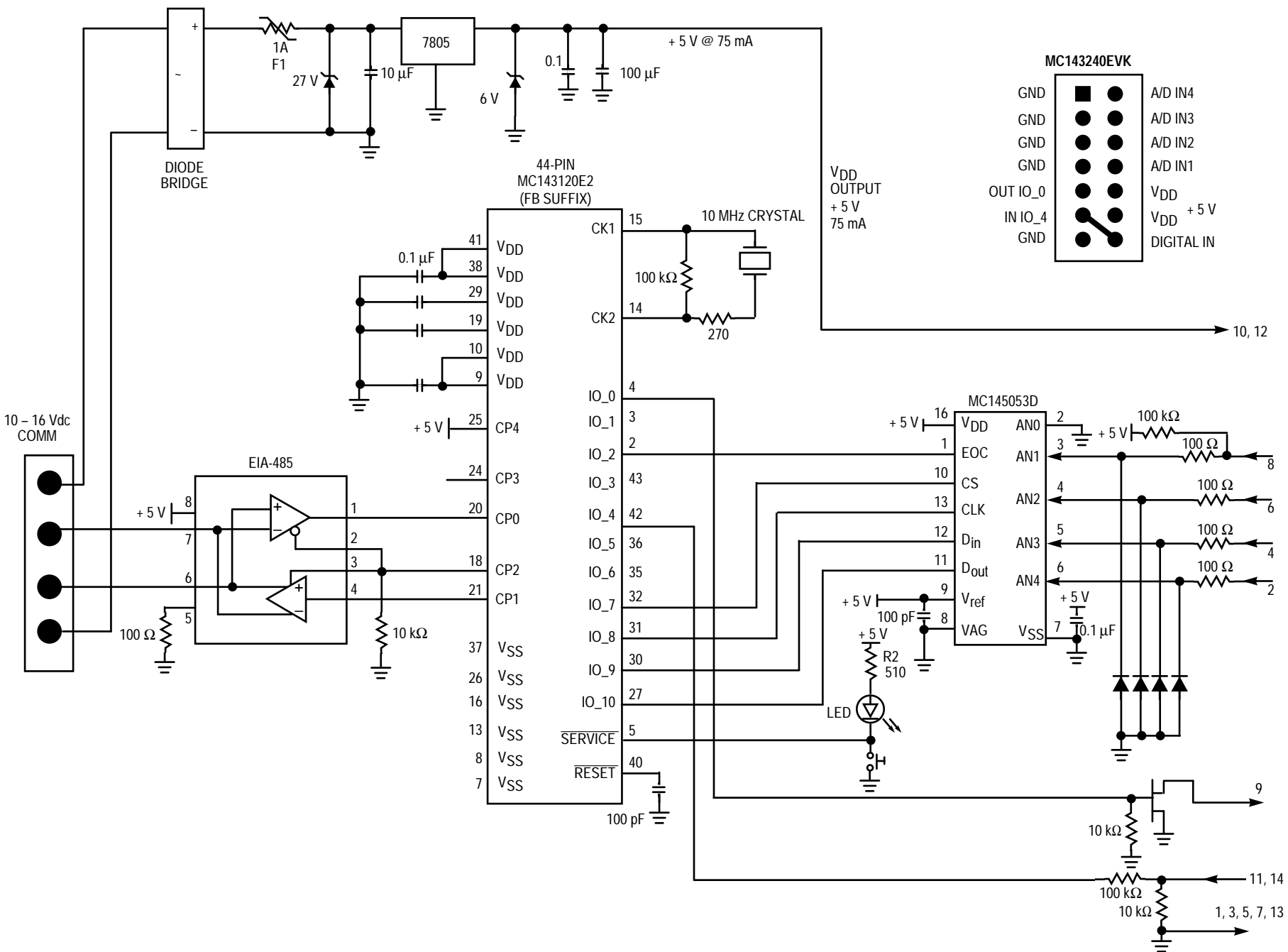


Figure F-17. MC143238EVK 2-Channel — I/O Node





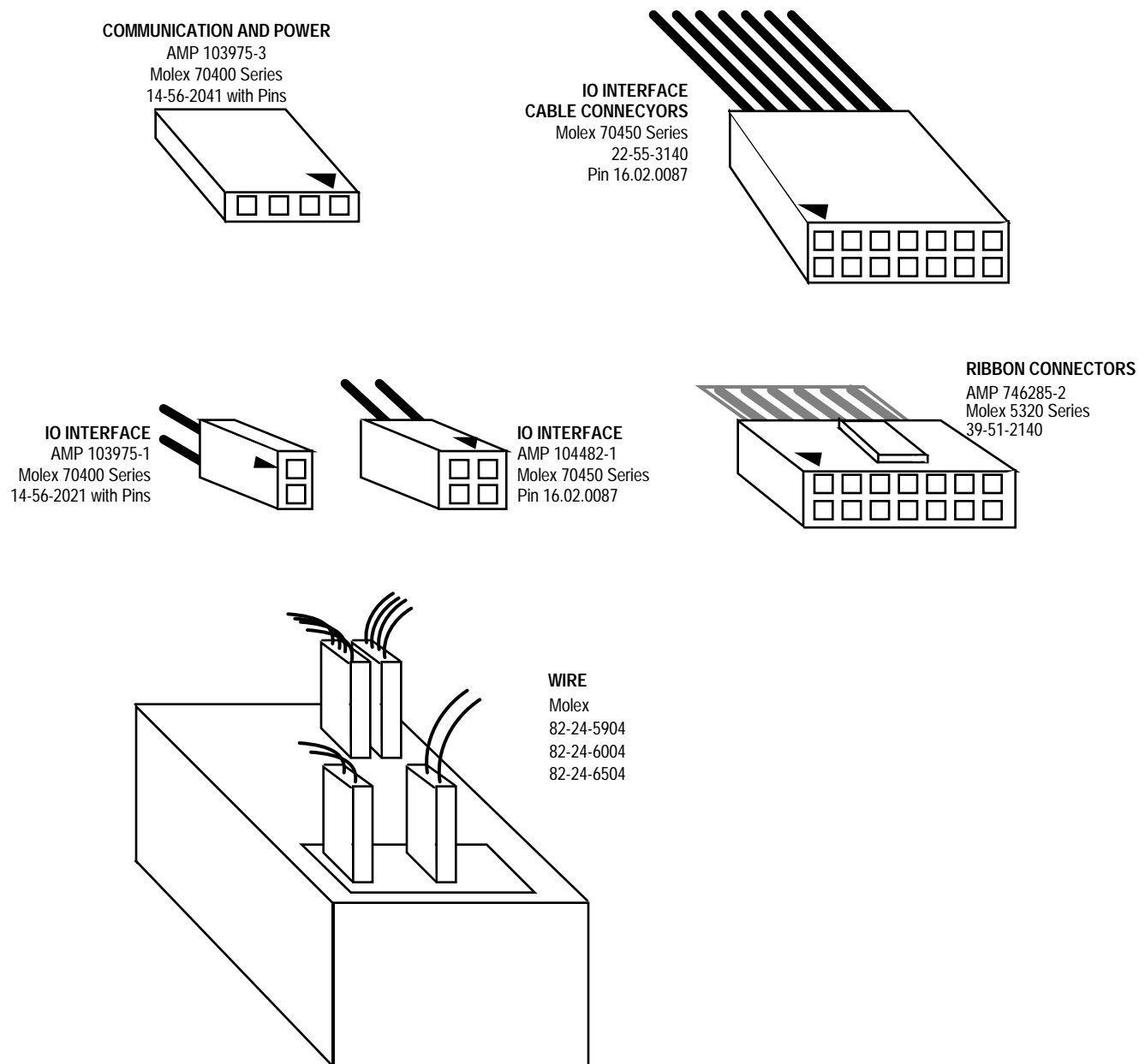


Figure F-20. LiteNode Kit Connectors

APPENDIX G CUSTOMER ALERTS

G.1 NEURON EEPROM PROTECTION

Introduction

A customer reported that the internal EEPROM of a Neuron Chip was becoming corrupted during reset and power up. The purpose of this alert is to remind the user of the need to protect the Neuron IC EEPROM through design practices, like good power and ground filtering, and isolation of noisy circuits. Figure 4-14 shows an LVI circuit that will help maintain the integrity of EEPROM during fluctuations of voltage from the power supply.

Background

Control systems with reprogrammable nonvolatile memory offer tremendous flexibility for many applications. The Neuron Chip includes both EEPROM memory and a built-in charge-pump that generates the required programming voltage, so that memory can be written over the network using the LonTalk protocol. The internal EEPROM is used to store system configuration data, application code and configuration data, and network configuration data. These data are changed infrequently and must be preserved during both reset and power-up events. This appendix describes certain design parameters that should be followed to maintain the integrity of the EEPROM during these events, whether the Neuron Chip is purchased directly from Motorola or Toshiba, or if a module containing the Neuron Chip is purchased from Echelon.

Fault Conditions

Protective measures should be taken to maintain the integrity of the EEPROM memory from corruption during reset or power up. A system designer using LONWORKS technology will choose the appropriate protective measures based on a careful assessment of the application and the environment.

A pending or on-going EEPROM write operation is not guaranteed if the Neuron Chip is not within the specified power supply voltage range. The total erase and write time is 20 ms. For example, consider a write operation which is under way when power is lost to the node. When power is reapplied, the data stored in EEPROM may either be old data, new data, or some totally different value. Also, stored data may change with time, due to inadequate erasure or programming time. *External voltage indicators or other means can ensure that the supply voltage is maintained within the specified operating range through the duration of the write cycle.*

The Neuron Chip computes checksums of any application and configuration data which are stored in EEPROM. Checksum failures result in the node going to either an "applicationless" or "unconfigured" state. *To minimize the chances of this occurring, it is best not to change node state or configuration frequently when power is likely to be lost; e.g., calling "go_unconfigured" from within an application after every power up is not advised.* This caution is based on the assumption that a power up can often be immediately followed by another power down. Were the application to be in the middle of the "go_unconfigured" call when the power down occurred, the node would likely end up in the applicationless state (the node state is considered to be part of the application).

One exception to the checksum failure rule stated above occurs when using self-installation functions. A power down while an application is modifying its own configuration (via self-installation routines, such as "update_domain") will not result in a checksum error. It is assumed that the application will reapply the configuration changes upon power up, thus correcting the configuration. Therefore, a faulty checksum is ignored when self-installation is interrupted by reset or power cycle.

In conclusion, EEPROM is a critical part of the Neuron Chip's flexibility in many applications and installation scenarios. In order to ensure reliable operation, Echelon recommends the guidelines discussed above to protect EEPROM data.

G.2 Neuron CHIP HANDLING PRECAUTIONS

Introduction

This alert describes handling precautions for Neuron Chips to prevent possible failures due to moisture-induced thermal stress and electrostatic discharge. These precautions are typical of those recommended for all CMOS VLSI devices.

Moisture-Induced Thermal Stress Precautions

Surface mount chips may crack when thermal stress is applied during surface mount assembly after exposure to atmospheric moisture. This is especially true of VLSI devices, such as the Neuron Chip. One customer has reported problems that were traced to this cause. It is important to note that such cracks may produce latent failures; therefore, final product testing is not a substitute for proper handling as described in Section D.2.

Damage to Lead Finish

Do not bake devices for more than 24 hours, since damage to lead finish may result. An exception is made if a nitrogen purge oven is used. It is acceptable to bake for more than 24 hours in a nitrogen purge oven without damage to lead IC finish.

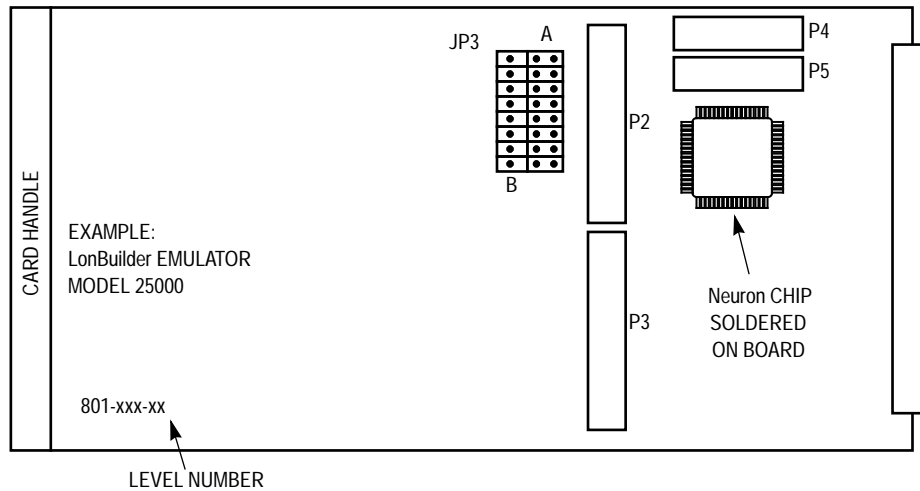
G.3 USING ECHELON'S EMULATOR BOARDS WITH MOTOROLA'S DIRECT-CONNECT BOARD (M143204EVK)

This alert applies to the LonBuilder cards where the Neuron Chip is soldered directly on the board. This board can also be identified by the -03- as the last 2 digits of the level number on the board.

The previous LonBuilder cards have the Neuron Chip soldered to a daughter board that plugs into the LonBuilder card and is referenced by -01- or -02- as the last 2 digits of the level number on the board.

The latest revision of the LonBuilder emulator boards have a backplane transceiver jumper labeled JP3 that enables or disables the built-in backplane transceiver.

Make certain that the JP3 jumpers are in the setting "A" position, indicating "EXTERNAL TRANSCEIVER" mode before the direct-connect transceiver board is installed into connector P2. This will allow the LonBuilder Developer's Workbench to provide voltage to the direct-connect transceiver board and allow the direct-connect board to provide voltage out the external RJ-45 connector, if needed.



Other references:

- LonBuilder Developer's Workbench Startup and Hardware Guide
- M143204EVK Direct-Connect Transceiver Board User's Manual

G.4 LONBUILDER 3.0 SOFTWARE REVISION

Summary

This alert applies only to LonBuilder systems running release 3.0 and 3.00.01. This alert does not affect NodeBuilder systems.

This software patch should be loaded, if one of the following applies:

- Creating PROM or flash programmer files with a system image for devices with more than 32K of off-chip ROM or flash. This does not apply to EEPROM images or flash images that do not include the system image.
- Importing external interface files for host applications into a LonBuilder project, where the host application has more than 85 network variables.

Description of the Patch

If exporting ROM files (.NRI extension) or flash files that include the system image (.NEI extension), and have specified more than 32 Kbytes (128 pages) of off-chip ROM or flash, then the device programmer file will contain incorrect information. If using a LONWORKS control module, a Neuron 3120 Chip, or a custom node with a 32 Kbyte PROM (such as a 27,256) for the system image, this problem does not apply.

If importing an external interface file (.XIF extension) for a host-based node that has more than 85 network variables, and execute a build operation on that node, the PC memory may become corrupted, causing unpredictable behavior. If importing external interface files for Neuron Chip-hosted nodes, this problem does not apply.

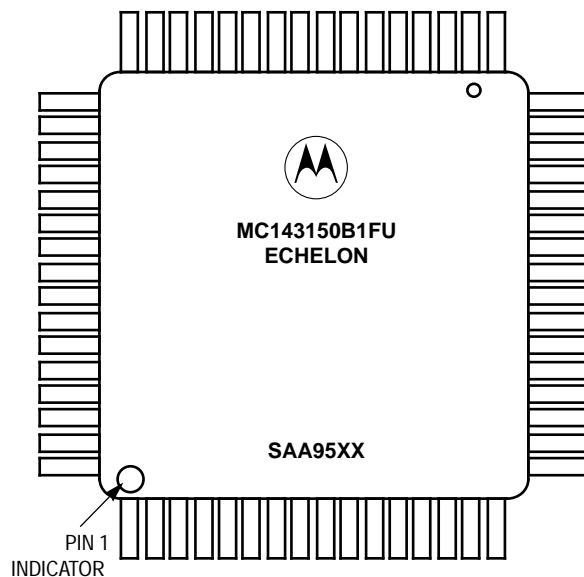
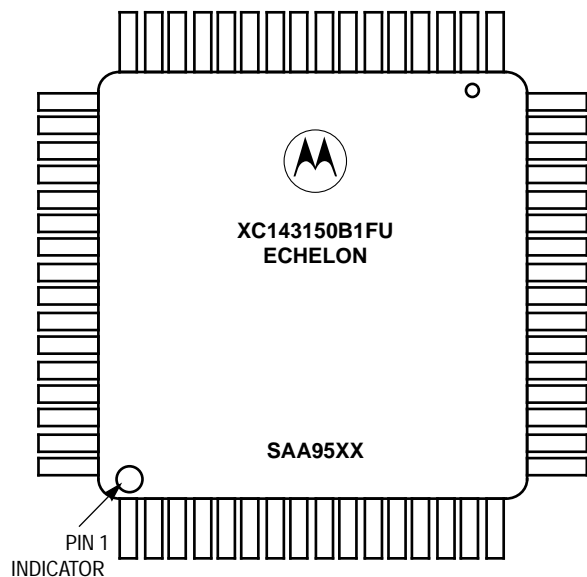
Description of Solution

Download the LBPATCH.ZIP archive file from Echelon's web site, <http://www.lonworks.echelon.com>, in the Developer's Toolbox area. Unpack this file using the PKUNZIP program. The procedure for updating your LonBuilder software is described in the README.TXT file included in that archive.

For questions about this alert, call 1-800-258-4LON (US and Canada) or +1-415-855-7400.

MC143150B1FU Neuron IC Product Alert

The pin 1 indicator has changed for the 0.8 micron XC and MC (143150B1FU) devices. The larger dimple at the bottom left of the marking indicates pin 1. Please notify the manufacturing area, so as to avoid any potential confusion regarding package orientation.



APPENDIX H

Neurowire (SPI INTERFACE) COMPATIBLE DEVICES

This is a small sampling of the products available from various manufacturers, and as such, is not an exhaustive list. For additional information and latest product updates, refer to the respective manufacturers' data books. This list will be updated on an ongoing basis.

MICROCONTROLLERS

4-Bit	COP400 family
8-Bit	Many varieties of MC68HC05s, MC68HC08 family, MC68HC11 family, MC68HC16 family, PIC16C64, PIC16C74, COP800 family
16-Bit	MC68302, MC68331/2/3/4, National HPC family
DSP	DSP56001, DSP56116, DSP56156

INPUT PORTS

74HC165	8-bit serial or parallel-input, serial-output shift register
74HC589	8-bit serial or latchable parallel-input, serial-output shift register with three-state output
74HC597	8-bit serial or latchable parallel-input, serial-output shift register with reset

OUTPUT PORTS

74HC595	8-bit serial-input, serial or parallel-output shift register with three-state outputs
MC33298	8-output, high-current, low side switch
CD4094	8-bit serial-input, three-state output
UCN58xx	Family of 8-, 10-, 12-, and 32-bit drivers, source or sink output, latched or non-latched, high current and voltage outputs available

MEMORIES

NMC93C06	16 x 16 CMOS EEPROM
NMC93C566	256 x 16 CMOS EEPROM, with write protect
MCM2814	256-byte serial EEPROM memory
X2444	16 x 16 serial NOVRAM, CMOS versions available
X24C45	16 x 16 serial NOVRAM, with AutoStore, CMOS
93AA46/56/66	1/2/4K bits (Microchip)

DISPLAY DRIVERS

MC145000/5001	12 FP + 4 BP multiplexed master/cascadable 12 FP slave LCD driver
MC145453	33-segment non-multiplexed LCD driver
MC14489	Multi-character LED display/lamp driver
MM5450	35-output LED display driver
MM5483	31-segment LCD display driver
MM58201	8-backplane and 24-segment multiplexed LCD driver
MM58241	32-output high-voltage display driver

A/D CONVERTERS*

ADC0831/2/4/8	8-bit, 1-/2-/4-/8-channel
LTC1091/92/93/94	10-bit, 1-/2-/6-/8-channel
LTC1291/92/93/94	12-bit, 1-/2-/6-/8-channel
LTC1095	10-bit, 6-channel, on-board reference
LTC1295	12-bit, 6-channel, on-board reference
MAX111	14-bit, 1-channel, low power with shut down
MAX121	14-bit, 1-channel, high speed
MAX170	12-bit, 1-channel
MAX190	12-bit, 1-channel, low power, on-board reference
MC145040/41	8-bit, 11-channel, external or internal conversion clock
MC145050/51	10-bit, 11-channel, external or internal conversion clock
MC145053	10-bit, 5-channel, internal conversion clock

* Refer to EB155, *Analog to Digital Conversion with the Neuron Chip* (Echelon Document No. 005-0019-01), in this data book for more information.

D/A CONVERTERS

MAX529	8-bit, 8-channel, single supply
MAX539	12-bit, 1-channel, low power
MAX543	12-bit, 1-channel

REAL-TIME CLOCK/CALENDAR

MC68HC68T1	Time-of-day clock/calendar + 32 bytes of scratchpad RAM
------------	---

AUDIO/RADIO

DS8906/7/8	AM/FM digital PLL synthesizers
DS8911	AM/FM/TV sound up-conversion frequency synthesizer
LMC1992/93	Stereo volume/tone/fade
LMC835	7-band graphic equalizer
MC145161, '67, '69	Cordless phone PLL frequency synthesizers
MC145149, '55-2, '56-2, '57-2, '58-2, '59-2, '70, '91, '92	General-purpose PLL frequency synthesizers, various frequency ranges for radio, TV, CATV, scanners, PCs, etc.
MC33218	Serial control of volume, mute, range selection, and operating mode
X9241	Quad EEPROM-based potentiometer

TELECOMMUNICATION

TP3400	Digital adapter for subscriber loops (DASL)
TP3410	Echo canceller
TP3420	S interface device

Manufacturer Prefix Key:

ADC = National	MC = Motorola
CD = National	MM = National
COP = National	PIC = Microchip
DS = National	TP = National
LTC = Linear Technology	UCN = Allegro Microsystems
LMC = National	X = Xicor
MAX = Maxim	

APPENDIX I USAGE GUIDELINES FOR ECHELON TRADEMARKS

NOTE: These guidelines do not cover usage of the LONWORKS® Independent Developer logo, the trademark LONMARK®, or the LONMARK logo (Section I.8 below).

I.1 ALWAYS USE ECHELON TRADEMARKS IN AN APPROVED FORM

Trademarks should be presented in the recommended styles shown below (all caps, upper and lower case, or a combination of large and small caps):

Registered Trademarks

Echelon®
LON®
LonTalk®
LonBuilder®
LonUsers®
NodeBuilder®

Neuron®
LonManager®
3150®
LONWORKS®
3120®

Other Trademarks

LonLink™
LonSupport™
LonResponse™

LONews™
LonMaker™
LonPoint™

Changes or alterations in the names above are not generally allowed. Any variation must be cleared through Echelon's Advertising and Creative Services Director.

RIGHT
LONWORKS®

RIGHT
LonTalk®

WRONG
Lonworks®

WRONG
LONTALK®

I.2 ALWAYS USE ECHELON TRADEMARKS IN A MANNER CLEARLY INDICATING THAT THEY ARE TRADEMARKS OWNED BY ECHELON

Trademarks should be properly marked to give notice that they are, in fact, trademarks of Echelon Corporation. Registered trademarks should always be used with the federal registration symbol “®,” while marks that have not been registered (including marks that are the subject of “pending” applications) should be used with the symbol “™” where the mark initially appears. Use of the registration symbol or “™” must appear with the first usage of the trademark in a document; subsequent occurrences do not require the symbols. When used in a printed document, the symbols should be half the point size of the word and then superscripted half the point size of the word.

A footnote reference to ownership of the trademarks must be used on all products, documentation, and advertisements in the following format: “**Echelon, LON, LONWORKS, LonBuilder, NodeBuilder, LonManager, LonTalk, LonUsers, Neuron, 3120, 3150, the Echelon logo, and the LonUsers logo are trademarks of Echelon Corporation registered in the United States and other countries. LonLink, LonResponse, LonSupport, LonMaker, and LonPoint are trademarks of Echelon Corporation.**” Where appropriate, a subset of this attribution may be used.

The list of registered and unregistered marks will be updated periodically as pending applications mature to registration, and as such, it is important to determine status of the marks before using either symbol.

Trademarks **should not be joined** with other terms (by a hyphen, for instance) nor used with **unapproved** logos, graphics, photos, slogans, numbers, design features, or symbols. Trademarks should never be “made plural,” never be mixed with other trademarks, and a trademark's spelling should never be altered.

I.3 ALWAYS USE ECHELON TRADEMARKS AS ADJECTIVES, NEVER NOUNS, AND THE MARKS SHOULD BE FOLLOWED BY THE APPROPRIATE GENERIC TERMINOLOGY

Trademarks are meant to signify the brand or source of the product and should not be used in a manner which suggests that the trademark is the name of the product. Companies that allow their trademarks to be used as nouns risk losing their rights in those trademarks. Classic examples of words that began as trademarks but were lost due to misuse include “aspirin,” “escalator,” and “cellophane.” Because the public came to see these terms as the product names instead of the brand names, trademark rights were forfeited.

The most common mistake is to use the trademark as a noun instead of as an adjective followed by the generic term. Examples:

RIGHT

“LONWORKS® networks can control existing home systems.”

WRONG

“LONWORKS® can control existing home systems.”

RIGHT

“The LonTalk® protocol is built into every Neuron® chip.”

WRONG

“LonTalk® is built into every Neuron®.”

I.4 DO NOT IMPLY OR SUGGEST THAT A PRODUCT BASED ON ECHELON'S TECHNOLOGY IS AN ECHELON PRODUCT OR THAT ECHELON SPONSORS THE PRODUCT

While Echelon understands and appreciates the customer's need to accurately describe the technology incorporated into the product as Echelon technology, the product materials and advertising can not wrongly imply that the customer's product is an Echelon product, explicitly or implicitly. Care must be taken to clearly distinguish Echelon's products (and trademarks) from the customer's products and marks, as discussed more specifically in these guidelines. Some basic examples are:

RIGHT

“The ACME™ light switch is for use in LONWORKS® networks.”

WRONG

“The LONWORKS® light switch is made by ACME™.”

RIGHT

“The ACME™ security system incorporates Echelon Corporation's LonTalk® protocol.”

WRONG

“The ACME™ LonTalk® security system incorporates the Echelon protocol.”

I.5 NEVER USE THE TRADEMARK “LON” AS A SEPARATE WORD, EVEN IF USED AS AN ADJECTIVE

Even though LON is a registered trademark of Echelon Corporation, it should not be used by a customer except as part of Echelon's family of marks, such as LonTalk, LonBuilder, and the like.

I.6 NEVER USE AN ECHELON TRADEMARK AS PART OF YOUR NAME OR TRADEMARK

Echelon trademarks may not be used as part of another company's mark or company name. While Echelon's marks can be used in explanatory fashion (such as “for use in LONWORKS® networks”), the mark can not be used as part of the company or product name. The only exception to this rule relates specifically to LON, where precise guidelines have been established. The mark LON may be used ONLY if it is incorporated into a composite mark in such a way that the LON portion is not highlighted in any way, including by CAPITAL letters, graphics, or stylizations, and that it is not used as the beginning letters of the composite word mark.

ACCEPTABLE

Avalon
DRESLON
brinlon
Elongate

UNACCEPTABLE

AvaLON
DresLon
Lonbrin
ELongate

Also, customers may not personify trademarks or create characters that represent the trademarks. However, the trademarks may be used in taglines in accordance with the guidelines listed above.

I.7 ALWAYS USE ECHELON LOGOS (WITH WORDS OR WITHOUT) PROPERLY, IN ACCORDANCE WITH THESE GUIDELINES

Echelon's logos are company assets just like Echelon's word marks. These logos must be used in the approved format and should never be modified without prior written approval. The Echelon logo and Echelon symbol are approved for use on advertising of products incorporating Echelon technology, provided that the logo or symbol is not used in a way that would give the end-user or purchaser a mistaken impression that the product is an Echelon product or is sponsored or approved by Echelon. Echelon will provide camera-ready artwork of the Echelon logo and symbol.



In general, the registered trademark symbol (®) must be used with every occurrence of the logos. Also, the trademark symbol should be used in the position indicated in these examples.

When Echelon symbols and logos are used, they should be included in the footnote reference to ownership of the trademarks, in addition to the trademark symbols used with the first occurrence of the symbol or logo. Depending on which words, symbols, and logos are used, all or a subset of the following may be used: “Echelon, LON, LONWORKS, LonBuilder, NodeBuilder, LonManager, LonTalk, LonUsers, Neuron, 3120, 3150, the Echelon logo, and the LonUsers logo are trademarks of Echelon Corporation registered in the United States and other countries. LonLink, LonResponse, LonSupport, LonPoint, and LonMaker, are trademarks of Echelon Corporation.”

I.7.1 Echelon Logo and Accompanying Words

The Echelon logo is composed of two main components: *The Echelon Symbol*. This is the graphic element. The registered trademark symbol (®) should appear at the lower right hand corner of the symbol when used by itself such as on products. *The Accompanying Word*. This is the word Echelon that may at times appear beside the symbol. This word should accompany every use of the Echelon logo on packaging, advertising, collateral, and documentation.

I.7.2 Placement Requirements

A minimal amount of empty space must be left between the Echelon logo and any other object such as type, photography, borders, edges, etc. The required area must be $1/2x$, where x = the height of the symbol when it is used by itself.

You may not combine the logo with any other feature including, but not limited to, other logos, words, graphics, photos, slogans, numbers, design features, or symbols.

I.7.2.1 Color Treatment. The preferred color treatment for the Echelon logo is the four-color application. This treatment uses the four Echelon colors (PMS 320, PMS 123, PMS 185, PMS 402). See the Echelon letterhead to see how these colors are used.

I.7.2.2 One-Color Applications. The Echelon logo may also appear in one-color applications. Black, white, or any of the four Echelon colors is preferred. However, any color that provides sufficient contrast with the background is acceptable. The logo may be positive or reversed.

I.8 CERTAIN ECHELON MARKS ARE NOT AVAILABLE FOR USE EXCEPT BY WRITTEN LICENSE AGREEMENT AND COMPLIANCE WITH STANDARDS

Echelon owns marks which may only be used by customers that have executed written agreements and have complied with certain requirements. These marks include the LONWORKS Independent Developer logo, the trademark LONMARK, and the LONMARK logo. This document **does not** give the customer the right to use such trademarks. These marks may only be used by approved developers and members of the LONMARK Interoperability Association. For more information about these marks and the association, please contact the LONWORKS Independent Developer Program Manager or LONMARK Program Manager at Echelon Corporation, 4015 Miranda Avenue, Palo Alto, CA 94303, USA.

Glossary **GL**

GLOSSARY

A

ABS	average busy stream (a built-in function code)
ALU	arithmetic logic unit
API	Application Programming Interface

B

BIST	built-in self test
BCD	binary coded decimal
BP	Base Page

C

CDet	collision detect
CPU	central processing unit
CRC	cyclic redundancy check
CSMA	carrier sense multiple access

E

EMI	electromagnetic interference
EOM	end of message
ESD	electrostatic discharge

H

HS	handshake
----	-----------

I

I2C	Inter-Integrated Circuit (Philips trademark)
IR	Infrared

L

LRC	Longitudinal Redundancy Check
LSB	least significant bit/byte
LVI	low-voltage inhibit

M

MAC	media access control
MCU	microcontroller unit
MIP	Microprocessor Interface Program
MPU	microprocessor unit
MSB	most significant bit

N

ND	network diagnostic
----	--------------------

NEC	National Electric Code
NM	network management
NPO	non-polarized

P

PAL	programmable array logic
PCB	printed circuit board
PCLTA	PC LonTalk Adapter
PTC	positive temperature coefficient

R

RF	radio frequency
RFI	radio frequency interference

S

SCL	serial clock
SCR	silicon-controlled rectifier
SDA	serial data
SIDAC	trademark of Teccor Corp.
SLTA/2	Serial LonTalk Adapter
SMT	surface mount technology
SNVT	Standard Network Variable Type

T

TOS	top of stack
-----	--------------

V

VLSI	very large-scale integration
------	------------------------------

X

XIF	external interface data
-----	-------------------------

A

A/D *see Analog-to-Digital*

acknowledge 9–15 (I), 9–31 (I)
address table 9–5 (I), 9–12 (I), 9–39 (I),
9–90 (I), EB–37 (II), AL–177 (III)
alerts 9–132 (I)
Analog-to-Digital 9–137 (I), EB–88 (II),
AL–34 (III), AL–35 (III), AL–38 (III),
AL–163 (III)
authentication 7–16 (I), 8–5 (I), 9–12 (I),
9–31 (I), 9–33 (I), EB–16 (II),
AL–20 (III)
automatic installation EB–17 (II)

B

board layout *see Appendix D*
broadcast address 9–13 (I)
buffer 9–4 (I), 9–6 (I), EB–34 (II), EB–169 (II),
EB–240 (II), EB–267 (II), EB–270 (II),
EB–281 (II), AL–147 (III)

C

capacitors EB–165 (II)
checksum 9–31 (I), 9–46 (I), 9–132 (I)
clock 1–6 (I), 4–3 (I), 4–17 (I), 4–18 (I),
9–4 (I), EB–43 (II), EB–46 (II),
EB–85 (II), EB–88 (II), EB–146 (II),
EB–148 (II), EB–150 (II), EB–168 (II),
EB–184 (II)
collision avoidance 8–6 (I), EB–27 (II),
EB–28 (II), EB–32 (II), EB–34 (II)
collision detection 1–6 (I), 4–6 (I), 8–6 (I),
9–44 (I), EB–34 (II), EB–129 (II)
communications 1–6 (I), 4–3 (I), 4–4 (I),
EB–11 (II), EB–173 (II), EB–175 (II),
EB–264 (II), AL–3 (III), AL–44 (III),
AL–175 (III)
differential mode 1–6 (I), 4–4 (I), 4–5 (I),
4–8 (I), 6–15 (I)
single-ended mode 4–4 (I), 4–5 (I), 4–6 (I)
special-purpose mode 4–4 (I), 4–9 (I)
see transceivers
configuration structure 9–4 (I), 9–24 (I),
9–31 (I), 9–35 (I), 9–93 (I), EB–11 (II),
EB–46 (II)

D

D/A *see Digital-to-Analog*

Digital-to-Analog 9–137 (I), AL–35 (III)

E

Echelon 1–4 (I), EB–19 (II), EB–34 (II),
EB–38 (II), EB–43 (II), EB–85 (II),
EB–173 (II), EB–179 (II), EB–195 (II),
EB–224 (II), EB–226 (II), EB–253 (II),
EB–261 (II), AL–10 (III), AL–22 (III),
AL–30 (III), AL–112 (III), AL–145 (III)
licensing 4–9 (I)
trademark usage 9–139 (I)
EEPROM 9–132 (I), 9–136 (I), EB–16 (II),
EB–184 (II), EB–185 (II), EB–193 (II),
AL–164 (III), AL–171 (III), AL–184 (III)
protection 1–6 (I), 9–132 (I)
EIA-232 9–96 (I), EB–163 (II), AL–3 (III),
AL–10 (III), AL–101 (III), AL–103 (III),
AL–112 (III), AL–146 (III), AL–193 (III)
EIA-485 4–12 (I), 4–13 (I), 9–100 (I),
EB–223 (II), AL–3 (III), AL–47 (III),
AL–195 (III)
electrical specifications 6–3 (I), 6–4 (I),
EB–85 (II), AL–3 (III)
communications port
glitch filter 6–14 (I)
hysteresis 6–14 (I)
differential transceiver 6–15 (I)
EPROM 9–3 (I), EB–183 (II), AL–86 (III),
AL–93 (III), AL–165 (III), AL–188 (III)
explicit messages 7–4 (I), 9–31 (I), 9–32 (I),
EB–24 (II), EB–244 (II), AL–147 (III),
AL–171 (III), AL–177 (III), AL–180 (III),
AL–181 (III), AL–183 (III), AL–186 (III),
AL–187 (III), AL–189 (III)
external memory 4–17 (I), 4–18 (I), 6–11 (I),
6–12 (I), 9–57 (I), EB–183 (II),
AL–16 (III), AL–20 (III), AL–85 (III)
EPROM memory interface 9–57 (I),
EB–186 (II)
with 32 Kbyte EPROM AL–161 (III),
AL–165 (III)
with 32 Kbyte EPROM and 24 Kbyte RAM
9–58 (I)

F

firmware 1–5 (I), 1–6 (I), EB–6 (II), EB–7 (II),
EB–16 (II), EB–34 (II), EB–39 (II),
EB–82 (II), EB–149 (II), EB–163 (II),
EB–164 (II), EB–183 (II), EB–184 (II),

- EB-185 (II), EB-187 (II), EB-189 (II),
EB-273 (II), AL-10 (III), AL-34 (III),
AL-85 (III), AL-87 (III), AL-146 (III),
AL-161 (III)
- additional library functions 7-10 (I)
- built-in variables 7-15 (I)
- extensions 7-16 (I)
- I/O timing 5-10 (I)
- scheduler I/O timing 4-26 (I), 5-8 (I)
- supporting Neuron 3120 ICs 7-16 (I),
AL-113 (III)
- version 4-31 (I), 9-6 (I), 9-52 (I),
AL-114 (III)
- flash 9-3 (I), 9-60 (I), 9-61 (I), 9-62 (I)
- functions *see I/O models*

G

- Glossary GL-3
- group address 9-13 (I), 9-39 (I)

I

- I/O 5-3 (I), AL-9 (III)
- 16-bit timer/counters 1-6 (I), 5-3 (I),
5-37 (I), EB-6 (II), EB-7 (II)
- bidirectional pins 1-6 (I), EB-6 (II),
EB-7 (II), EB-38 (II), EB-82 (II),
EB-85 (II), EB-88 (II), AL-9 (III),
AL-22 (III), AL-74 (III), AL-76 (III),
AL-151 (III), AL-161 (III)
- serial I/O objects EB-164 (II), EB-165 (II),
EB-167 (II)
- timing issues 5-8 (I), 5-10 (I), 5-16 (I),
5-17 (I), 5-21 (I), 5-31 (I), 5-32 (I),
6-11 (I), 6-12 (I), EB-4 (II),
EB-180 (II), EB-184 (II),
AL-16 (III), AL-22 (III), AL-32 (III),
AL-38 (III)
- I/O models (objects) 5-3 (I), EB-55 (II),
EB-82 (II)
- direct I/O modes 5-4 (I)
- bit I/O 5-8 (I), 5-10 (I), EB-82 (II)
- byte I/O 5-10 (I), 5-12 (I)
- leveldetector input 5-6 (I), 5-10 (I),
5-13 (I), 5-36 (I)
- nibble I/O 5-10 (I), 5-13 (I), 5-14 (I),
EB-39 (II), AL-38 (III)
- parallel I/O modes 5-4 (I), AL-9 (III),
AL-74 (III), AL-145 (III),
AL-151 (III), AL-153 (III)

- muxbus 5-55 (I)
- serial I/O modes 5-5 (I), 5-24 (I), 5-33 (I),
5-34 (I), EB-43 (II), EB-164 (II),
EB-167 (II), EB-168 (II),
EB-170 (II), EB-171 (II),
AL-10 (III)
- bitshift I/O 5-24 (I), 5-25 (I), 5-26 (I)
- I²C 5-26 (I), 5-27 (I)
- magcard input 5-28 (I)
- magtrack1 input 5-29 (I)
- Neurowire I/O 5-30 (I), 5-31 (I),
5-32 (I), EB-43 (II), EB-69 (II),
EB-70 (II), EB-72 (II),
EB-88 (II), AL-10 (III),
AL-35 (III), AL-46 (III)
- touch I/O 5-34 (I), 5-35 (I)
- wiegand input 5-36 (I), 5-37 (I)
- timer/counter input modes 5-5 (I), 5-37 (I),
5-38 (I), EB-61 (II), EB-64 (II),
EB-82 (II)
- dualslope input 5-39 (I), 5-56 (I),
EB-55 (II), EB-64 (II),
EB-66 (II)
- edgelog input 5-40 (I), 5-56 (I)
- infrared input 5-41 (I), 5-56 (I),
EB-255 (II)
- on-time EB-61 (II), EB-78 (II),
EB-88 (II), AL-38 (III)
- ontime 5-38 (I), 5-42 (I), 5-56 (I)
- period input 5-43 (I), 5-56 (I)
- pulsecount input 5-44 (I), EB-78 (II)
- quadrature input 5-45 (I), EB-3 (II)
- totalcount input 5-46 (I)
- timer/counter output modes 5-6 (I),
5-37 (I), 5-47 (I), EB-65 (II)
- edgedivide output 5-47 (I), 5-56 (I)
- frequency output 5-48 (I), 5-56 (I),
EB-88 (II)
- oneshot output 5-49 (I), 5-56 (I)
- pulsecount output 5-50 (I), 5-56 (I)
- pulsewidth output 5-51 (I), 5-56 (I)
- triac output 5-52 (I), 5-53 (I), 5-56 (I)
- triggered count output 5-54 (I), 5-56 (I)
- ID *see Neuron ID*
- installation 9-97 (I), 9-133 (I), EB-10 (II),
EB-83 (II), EB-179 (II), EB-186 (II),
EB-193 (II), EB-195 (II), EB-256 (II),
EB-257 (II), AL-44 (III), AL-163 (III),
AL-175 (III), AL-196 (III)

Internet *see* *World Wide Web*

L

licensing

Echelon 9–139 (I)

LiteNode Kit Connectors 9-131 (I)

LonBuilder 1–3 (I), 1–4 (I), 1–5 (I), 2–3 (I),
9–3 (I), 9–134 (I), EB–12 (II),
EB–14 (II), EB–17 (II), EB–82 (II),
EB–147 (II), EB–179 (II), EB–180 (II),
EB–181 (II), EB–182 (II), EB–183 (II),
EB–184 (II), EB–185 (II), EB–186 (II),
EB–187 (II), EB–189 (II), EB–193 (II),
AL–16 (III), AL–20 (III), AL–22 (III),
AL–75 (III), AL–85 (III), AL–87 (III),
AL–97 (III), AL–112 (III), AL–145 (III),
AL–146 (III), AL–183 (III), AL–189 (III)

LonManager 9–5 (I), 9–97 (I), EB–21 (II),
EB–23 (II), EB–25 (II), EB–26 (II)

LONMARK EB–195 (II), AL–177 (III),
AL–189 (III)

LonTalk 1–3 (I), 1–6 (I), 2–3 (I), 8–3 (I),
9–3 (I), 9–30 (I), 9–97 (I), EB–10 (II),
EB–12 (II), EB–13 (II), EB–14 (II),
EB–16 (II), EB–18 (II), EB–23 (II),
EB–24 (II), EB–27 (II),
EB–117 (II)–EB–143 (II), EB–145 (II),
EB–146 (II), EB–148 (II), EB–163 (II),
EB–164 (II), EB–226 (II), EB–240 (II),
EB–265 (II), AL–44 (III), AL–55 (III),
AL–146 (III), AL–161 (III), AL–164 (III),
AL–171 (III), AL–175 (III)

acknowledge 7–5 (I), 8–5 (I), EB–31 (II),
EB–129 (II), EB–145 (II),
AL–31 (III), AL–148 (III),
AL–164 (III)

addressing limits 8–4 (I)

request response 7–5 (I), 8–5 (I),
EB–33 (II), EB–129 (II),
EB–240 (II), AL–164 (III),
AL–184 (III)

unackd_rpt 7–5 (I), 8–5 (I), EB–129 (II),
AL–164 (III)

unacknowledge 7–5 (I), 8–5 (I),
EB–129 (II), EB–145 (II),
EB–240 (II), AL–148 (III),
AL–164 (III)

LONWORKS 2–3 (I), EB–27 (II), EB–28 (II),
EB–33 (II), EB–37 (II), EB–88 (II),

EB–117 (II), EB–135 (II), EB–138 (II),
EB–163 (II), EB–179 (II), EB–223 (II),
EB–261 (II), EB–263 (II), EB–264 (II),
EB–265 (II), EB–267 (II), AL–14 (III),
AL–48 (III), AL–61 (III), AL–62 (III),
AL–74 (III), AL–77 (III), AL–175 (III),
AL–176 (III), AL–177 (III), AL–178 (III),
AL–184 (III), AL–189 (III), AL–190 (III),
AL–191 (III), AL–197 (III)

overview and architecture 2–3 (I),
EB–10 (II)

programming model 7–3 (I), AL–112 (III),
AL–145 (III), AL–191 (III),
AL–192 (III), AL–197 (III)

M

M143120DWEVK 9–100 (I)

M143120FBEVK 9–103 (I)

M143150EVK 9–106 (I)

M143204EVK 9–110 (I)

M143206EVK 9–114 (I)

M143208EVK 9–117 (I)

M143232EVK 9–121 (I)

M143235EVK 9–124 (I)

MC143120B1 2–5 (I)

MC143120E2 2–6 (I)

programming AL–112 (III)

MC143120FE2 2–8 (I)

MC143120LE2 2–10 (I)

MC143150B1 2–12 (I)

MC143150B2 2–13 (I)

MC143238EVK 9-126 (I), 9-128 (I)

MC143239EVK 9-126 (I), 9-129 (I)

MC143240EVK 9-130 (I)

MC143245EVK 9-127 (I)

memory 9–57 (I), 9–132 (I), EB–12 (II),
EB–183 (II), EB–184 (II), EB–185 (II),
AL–85 (III)

allocation AL–16 (III), AL–20 (III),
AL–171 (III)

base page layout 3–7 (I)

checksums 9–132 (I)

see EEPROM

see EPROM

external 4–17 (I), 4–18 (I), AL–85 (III)

map 9–88 (I), AL–20 (III), AL–87 (III),
AL–97 (III)

preprogrammed ROM 1–6 (I), AL–85 (III)

static RAM 1–6 (I), 9–62 (I), AL–85 (III),

- AL-87 (III), AL-93 (III)
- memory structures and tables
 - domain tables 9-4 (I), 9-5 (I), 9-11 (I), 9-12 (I), 9-34 (I), 9-89 (I), AL-176 (III), AL-178 (III), AL-179 (III)
- Neuron fixed structure 9-4 (I), 9-88 (I), 9-91 (I)
- Neuron memory maps 9-5 (I), 9-87 (I), AL-87 (III)
- read-only data structure 9-6 (I)
- message services 8-5 (I), 9-30 (I), EB-14 (II), EB-30 (II), EB-31 (II), EB-33 (II), EB-36 (II), EB-37 (II), AL-184 (III)
- MIP (Microprocessor Interface Program) 9-41 (I), 9-91 (I), EB-24 (II), EB-163 (II), AL-10 (III), AL-189 (III)
- model number 9-8 (I)
- monitoring and control EB-24 (II)
- Motorola
 - evaluation and I/O interface boards AL-22 (III), AL-61 (III), AL-113 (III)
 - phone numbers *see back of book*

N

- network address EB-13 (II)
- network driver
 - required functions EB-267 (II)
- network management 7-3 (I), 9-93 (I), EB-10 (II), EB-12 (II), EB-23 (II), EB-24 (II), EB-28 (II), EB-30 (II), EB-93 (II), EB-96 (II), EB-179 (II), EB-186 (II), AL-10 (III), AL-14 (III), AL-112 (III), AL-115 (III), AL-146 (III), AL-148 (III), AL-150 (III), AL-152 (III), AL-163 (III), AL-183 (III), AL-189 (III), AL-191 (III), AL-196 (III)
- diagnostic services 8-6 (I), 9-30 (I)
- network variables 9-4 (I), 9-17 (I), 9-31 (I), EB-16 (II), EB-24 (II), EB-40 (II), EB-91 (II), EB-92 (II), EB-93 (II), EB-94 (II), EB-95 (II), EB-96 (II), EB-97 (II), EB-100 (II), EB-101 (II), EB-102 (II), EB-103 (II), EB-104 (II), EB-105 (II), EB-106 (II), EB-107 (II), EB-108 (II), EB-149 (II), EB-169 (II), EB-195 (II), EB-240 (II), EB-241 (II), EB-242 (II), AL-10 (III), AL-35 (III), AL-47 (III), AL-76 (III), AL-77 (III),

- AL-146 (III), AL-171 (III), AL-172 (III), AL-173 (III), AL-174 (III), AL-177 (III), AL-193 (III)
- aliases 7-6 (I), 9-11 (I), 9-17 (I), 9-40 (I)
- configuration table field descriptions 9-17 (I), AL-186 (III)
- Neuron EB-10 (II), EB-14 (II), EB-16 (II), EB-38 (II), EB-43 (II), EB-147 (II), EB-183 (II), EB-253 (II), AL-34 (III), AL-44 (III), AL-55 (III)
- block diagram 1-3 (I), 1-4 (I)
- dry pack 9-68 (I), 9-133 (I)
- family 1-4 (I), 1-5 (I)
- see firmware*
- handling precautions 9-68 (I), 9-133 (I)
- hardware considerations 5-4 (I), 5-10 (I), EB-5 (II), EB-6 (II), EB-7 (II), EB-10 (II), EB-28 (II), EB-34 (II), EB-43 (II), EB-240 (II), AL-34 (III), AL-37 (III), AL-44 (III), AL-85 (III)
- hardware design 9-73 (I)
- hardware resources 1-6 (I), EB-80 (II)
- instruction timings EB-147 (II)
- see LONWORKS*
- see model number*
- processing units AL-161 (III)
- register set 3-6 (I)
- specifications 1-5 (I)
- timer/counter external connections 5-3 (I)
- Neuron ID 9-32 (I), EB-14 (II)
- 48-bit ID 1-6 (I), 9-13 (I), 9-35 (I), AL-184 (III)
- address field descriptions 9-16 (I)
- address format 9-13 (I), 9-16 (I)
- Neurowire *see I/O models*
- NodeBuilder 1-4 (I), 2-3 (I), 9-3 (I), 9-52 (I), 9-98 (I)

O

- oscillator *see clock*

P

- package
 - MC143120 dimensions 6-21 (I)
 - MC143120 pad layout 6-23 (I)
 - MC143120 pin assignments 6-20 (I)
 - MC143150 dimensions 6-18 (I)
 - MC143150 pad layout 6-19 (I)
 - MC143150 pin assignments 6-17 (I)

- mechanical specifications 6–3 (I)
- pin descriptions 6–16 (I)
- sockets for Neuron ICs 6–23 (I)
- parallel I/O AL–9 (III), AL–15 (III)
 - handshaking 4–9 (I), 5–20 (I), 5–22 (I), AL–9 (III), AL–13 (III), AL–74 (III), AL–145 (III)
 - interface 5–15 (I), AL–9 (III), AL–145 (III)
 - MC683xx AL–74 (III), AL–146 (III)
 - MC68HC11 AL–14 (III), AL–15 (III), AL–145 (III), AL–151 (III)
 - slave A 5–15 (I), 5–17 (I), AL–9 (III), AL–74 (III)
 - slave B 5–19 (I), 5–20 (I), 5–21 (I), AL–9 (III), AL–74 (III), AL–76 (III)
 - token passing 5–20 (I), AL–9 (III), AL–10 (III), AL–11 (III), AL–75 (III), AL–76 (III), AL–145 (III)
- phantom router
 - creating AL–142 (III)
- pin assignment(s) *see package*
- preemption 7–9 (I), AL–16 (III)
- priority 4–7 (I), 8–6 (I), EB–16 (II), EB–34 (II), EB–243 (II), AL–11 (III), AL–20 (III), AL–152 (III)

Q

quadrature *see I/O models*

R

- Raytheon AL–197 (III)
- replacing a damaged node EB–13 (II)
- request response 9–15 (I), 9–31 (I)
- reset 4–3 (I), 9–34 (I), EB–85 (II), AL–14 (III), AL–15 (III)
 - 0.8 μ Neuron IC 4–22 (I)
 - LVI/LVD 4–23 (I), AL–14 (III), AL–150 (III)
 - MC143120 reset sequence 4–27 (I)
 - MC143150 reset sequence 4–28 (I)
 - Motorola low-voltage detector ICs
 - AL–14 (III), AL–150 (III)
 - output pin state transitions 4–30 (I)
 - power on 4–21 (I), 4–22 (I), 9–132 (I), EB–184 (II)
 - processes and timing 4–23 (I), 4–28 (I)
 - timeline
 - MC143120DW and MC143150FU/FU1 4–24 (I)
 - timing diagram for the 1.2 μ and 0.8 μ Neu-

- ron IC 4–22 (I)
- typical start-up times 4–19 (I)
- response time EB–28 (II)

S

- scheduler 1–6 (I), 7–5 (I), 7–9 (I), 9–3 (I), 9–34 (I), EB–37 (II), EB–169 (II)
- serial I/O modes
 - Neurowire I/O 9–136 (I)
- service pin 1–6 (I), 4–30 (I), 9–30 (I), 9–36 (I), EB–14 (II), EB–85 (II), EB–186 (II), EB–189 (II), EB–189 (II)–EB–192 (II), AL–184 (III), AL–191 (III)
 - buffer written into 4–31 (I), 4–32 (I)
 - circuit 4–31 (I)
- services EB–23 (II)
- sleep mode 1–6 (I)
 - sleep/wakeup circuitry 4–10 (I), 4–19 (I)
- SNVT EB–40 (II), EB–88 (II), EB–195 (II), EB–242 (II), EB–243 (II), AL–177 (III)
 - alias field descriptions 9–24 (I)
 - structures 9–20 (I), 9–35 (I), 9–92 (I), EB–242 (II), EB–243 (II), EB–244 (II)
- software
 - see firmware*
 - see LonBuilder*
 - see NodeBuilder*
- soldering 9–68 (I), 9–133 (I)
- special-purpose 4–4 (I), 4–7 (I), 4–9 (I), EB–263 (II)
- SPI *see I/O models (Neurowire)*
- subnet/node address 9–13 (I)
- support tools 2–3 (I), 4–3 (I), 9–96 (I), AL–75 (III), AL–112 (III), AL–145 (III), AL–148 (III), AL–149 (III), AL–160 (III), AL–175 (III), AL–191 (III), AL–197 (III)
 - Echelon 9–97 (I), EB–13 (II), EB–82 (II), EB–119 (II), EB–173 (II), EB–179 (II), EB–180 (II), EB–184 (II), EB–187 (II), EB–263 (II), EB–280 (II), AL–10 (III), AL–112 (III), AL–189 (III), AL–191 (III), AL–192 (III), AL–193 (III), AL–195 (III)
 - Motorola 9–96 (I), AL–55 (III), AL–112 (III)

T

- timer/counter circuits EB-88 (II)
- timer/counter input modes
 - infrared input 4-3 (I)
- timer/counters
 - see I/O objects*
 - pulse train output 5-57 (I)
 - resolution and range 5-56 (I)
 - square wave output 5-57 (I)
- timers 7-3 (I), 9-33 (I), EB-36 (II), EB-37 (II)
- tools *see support tools*
- transceivers 4-3 (I), 9-97 (I), 9-133 (I),
EB-184 (II), EB-280 (II), EB-281 (II),
EB-282 (II)
 - communications port 4-4 (I)
 - differential 4-8 (I), 9-28 (I), EB-263 (II),
AL-47 (III)
 - direct connect network interface 4-13 (I)
 - direct-drive 4-12 (I)
 - see EIA-232*
 - see EIA-485*
 - internal block diagram 4-4 (I)
 - packet timing 4-6 (I)
 - power-line 4-17 (I), 9-97 (I), EB-11 (II),
EB-163 (II), EB-184 (II),
EB-223 (II), EB-253 (II),
EB-280 (II), AL-48 (III)
 - radio frequency (RF) 4-3 (I), 4-17 (I),

- EB-28 (II), EB-256 (II), AL-48 (III)
- receiver jitter tolerance 4-8 (I)
- single-ended 4-4 (I)
- special-purpose mode 4-7 (I), 9-25 (I),
9-47 (I), EB-263 (II)
 - transmit and receive status bits 4-11 (I)
- transformer 4-12 (I), 4-13 (I), EB-173 (II)
- twisted pair 9-97 (I)
- twisted-pair 4-12 (I), 4-14 (I), EB-11 (II),
EB-34 (II), EB-163 (II),
EB-173 (II), EB-185 (II),
EB-223 (II), AL-160 (III)
- turnaround address 9-13 (I)

U

- unackd_rpt 9-31 (I)
- unacknowledge 9-31 (I)

W

- watchdog timer 4-20 (I), 9-44 (I), EB-273 (II),
EB-279 (II), AL-12 (III), AL-22 (III)
- when* clause 5-8 (I), 5-9 (I), 5-38 (I), 7-9 (I),
EB-37 (II), EB-153 (II), AL-16 (III),
AL-77 (III), AL-146 (III)
- World Wide Web AL-145 (III)
- WSI AL-85 (III), AL-86 (III), AL-87 (III),
AL-93 (III), AL-95 (III), AL-97 (III)

MOTOROLA AUTHORIZED DISTRIBUTOR & WORLDWIDE SALES OFFICES

NORTH AMERICAN DISTRIBUTORS

UNITED STATES

ALABAMA

Huntsville

Allied Electronics, Inc. (205)721-3500
 Arrow Electronics (205)837-6955
 FAI (205)837-9209
 Future Electronics (205)830-2322
 Hamilton/Hallmark (205)837-8700
 Newark (205)837-9091
 Wyle Electronics (205)830-1119

Mobile

Allied Electronics, Inc. (334)476-1875

ARIZONA

Phoenix

Allied Electronics, Inc. (602)831-2002
 FAI (602)731-4661
 Future Electronics (602)968-7140
 Hamilton/Hallmark (602)736-7000
 Wyle Electronics (602)804-7000

Tempe

Arrow Electronics (602)966-6600
 Newark (602)966-6340
 PENSTOCK (602)967-1620

ARKANSAS

Little Rock

Newark (501)225-8130

CALIFORNIA

Agoura Hills

Future Electronics (818)865-0040

Calabassas

Arrow Electronics (818)880-9686
 Wyle Electronics (818)880-9000

Culver City

Hamilton/Hallmark (310)558-2000

Irvine

Arrow Electronics (714)587-0404
 Arrow Zeus (714)581-4622
 FAI (714)753-4778
 Future Electronics (714)453-1515
 Hamilton/Hallmark (714)789-4100
 Wyle Laboratories Corporate .. (714)753-9953
 Wyle Electronics (714)789-9953

Los Angeles

FAI (818)879-1234

Manhattan Beach

PENSTOCK (310)546-8953

Newberry Park

PENSTOCK (805)375-6680

Orange County

Allied Electronics, Inc. (714)727-3010

Palo Alto

Newark (650)812-6300

Rancho Cordova

Wyle Electronics (916)638-5282

Riverside

Allied Electronics, Inc. (909)980-6522
 Newark (909)980-2105

Rocklin

Hamilton/Hallmark (916)632-4500

Roseville

Wyle Electronics (916)783-9953

Sacramento

Allied Electronics, Inc. (916)632-3104
 FAI (916)782-7882
 Newark (916)565-1760

San Diego

Allied Electronics, Inc. (619)279-2550
 Arrow Electronics (619)565-4800
 FAI (619)623-2888
 Future Electronics (619)625-2800
 Hamilton/Hallmark (619)571-7540
 Newark (619)453-8211
 PENSTOCK (619)623-9100
 Wyle Electronics (619)558-6600

San Fernando Valley

Allied Electronics, Inc. (818)598-0130

CALIFORNIA – continued

San Jose

Allied Electronics, Inc. (408)383-0366
 Arrow Electronics (408)441-9700
 Arrow Electronics (408)428-6400
 Arrow Zeus (408)629-4789
 FAI (408)434-0369
 Future Electronics (408)434-1122

Santa Clara

Wyle Electronics (408)727-2500

Santa Fe Springs

Newark (562)929-9722

Sierra Madre

PENSTOCK (818)355-6775

Sunnyvale

Hamilton/Hallmark (408)435-3600
 PENSTOCK (408)730-0300

Thousand Oaks

Newark (805)449-1480

Woodland Hills

Hamilton/Hallmark (818)594-0404

COLORADO

Lakewood

FAI (303)237-1400
 Future Electronics (303)232-2008

Denver

Allied Electronics, Inc. (303)790-1664
 Newark (303)373-4540

Englewood

Arrow Electronics (303)799-0258
 Hamilton/Hallmark (303)790-1662
 PENSTOCK (303)799-7845

Thornton

Wyle Electronics (303)457-9953

CONNECTICUT

Bloomfield

Newark (860)243-1731

Cheshire

Allied Electronics, Inc. (203)272-7730
 FAI (203)250-1319
 Future Electronics (203)250-0083
 Hamilton/Hallmark (203)271-5700

Wallingford

Arrow Electronics (203)265-7741
 Wyle Electronics (203)269-8077

FLORIDA

Altamonte Springs

Future Electronics (407)865-7900

Clearwater

FAI (813)530-1665
 Future Electronics (813)530-1222

Deerfield Beach

Arrow Electronics (305)429-8200
 Wyle Electronics (954)420-0500

Ft. Lauderdale

FAI (954)428-9494
 Future Electronics (954)426-4043
 Hamilton/Hallmark (954)677-3500
 Newark (954)486-1151

Jacksonville

Allied Electronics, Inc. (904)739-5920
 Newark (904)399-5041

Lake Mary

Arrow Electronics (407)333-9300
 Arrow Zeus (407)333-3055

Largo/Tampa/St. Petersburg

Hamilton/Hallmark (813)507-5000
 Newark (813)287-1578
 Wyle Electronics (813)576-3004

Miami

Allied Electronics, Inc. (305)558-2511

Maitland

Wyle Electronics (407)740-7450

Orlando

Allied Electronics, Inc. (407)539-0055
 FAI (407)865-9555
 Newark (407)896-8350

FLORIDA – continued

Tallahassee

FAI (904)668-7772

Tampa

Allied Electronics, Inc. (813)579-4660
 Newark (813)287-1578
 PENSTOCK (813)247-7556

Winter Park

Hamilton/Hallmark (407)657-3300
 PENSTOCK (407)672-1114

GEORGIA

Atlanta

Allied Electronics, Inc. (770)497-9544
 FAI (404)447-4767

Duluth

Arrow Electronics (404)497-1300
 Hamilton/Hallmark (770)623-4400

Norcross

Future Electronics (770)441-7676
 Newark (770)448-1300
 PENSTOCK (770)734-9990
 Wyle Electronics (770)441-9045

IDAHO

Boise

Allied Electronics, Inc. (208)331-1414
 FAI (208)376-8080

ILLINOIS

Addison

Wyle Laboratories (708)620-0969

Arlington Heights

Hamilton/Hallmark (847)797-7300

Chicago

Allied Electronics, Inc. (North) .. (847)548-9330
 Allied Electronics, Inc. (South) .. (708)535-0038
 FAI (708)843-0034
 Newark Electronics Corp. (773)784-5100

Hoffman Estates

Future Electronics (708)882-1255

Itasca

Arrow Electronics (708)250-0500
 Arrow Zeus (630)595-9730

Lombard

Newark (630)317-1000

Palatine

PENSTOCK (708)934-3700

Rockford

Allied Electronics, Inc. (815)636-1010
 Newark (815)229-0225

Springfield

Newark (217)787-9972

Wood Dale

Allied Electronics, Inc. (630)860-0007

INDIANA

Indianapolis

Allied Electronics, Inc. (317)571-1880
 Arrow Electronics (317)299-2071
 Hamilton/Hallmark (317)575-3500
 FAI (317)469-0441
 Future Electronics (317)469-0447
 Newark (317)844-0047
 Wyle Electronics (317)581-6152

Ft. Wayne

Newark (219)484-0766
 PENSTOCK (219)432-1277

IOWA

Bettendorf

Newark (319)359-3711

Cedar Rapids

Allied Electronics, Inc. (319)390-5730
 Newark (319)393-3800

KANSAS

Kansas City

Allied Electronics, Inc. (913)338-4372
 FAI (913)381-6800

Lenexa

Arrow Electronics (913)541-9542

AUTHORIZED DISTRIBUTORS – continued

UNITED STATES – continued

KANSAS – continued

Olathe
PENSTOCK (913)829-9330

Overland Park
Future Electronics (913)649-1531
Hamilton/Hallmark (913)663-7900
Newark (913)677-0727

KENTUCKY

Louisville
Allied Electronics, Inc. (502)452-2293
Newark (502)423-0280

LOUISIANA

New Orleans
Allied Electronics, Inc. (504)466-7575
Newark (504)838-9771

MARYLAND

Baltimore
Allied Electronics, Inc. (410)312-0810
FAI (410)312-0833

Columbia
Arrow Electronics (301)596-7800
Arrow Zeus (410)309-1541
Future Electronics (410)290-0600
Hamilton/Hallmark (410)720-3400
PENSTOCK (410)290-3746
Wyle Electronics (410)312-4844

Hanover
Newark (410)712-6922

MASSACHUSETTS

Bedford
Wyle Electronics (781)271-9953

Boston
Allied Electronics, Inc. (617)255-0361
Arrow Electronics (508)658-0900
FAI (508)779-3111
Newark 1-800-4NEWARK

Bolton
Future Corporate (978)779-3000

Burlington
PENSTOCK (617)229-9100

Lowell
Newark (978)551-4300

Peabody
Allied Electronics, Inc. (508)538-2401
Hamilton/Hallmark (508)532-3701

Wilmington
Arrow Zeus (978)658-4776

Worcester
Newark (508)229-2200

MICHIGAN

Detroit
Allied Electronics, Inc. (313)416-9300
FAI (313)513-0015
Future Electronics (616)698-6800

Grand Rapids
Allied Electronics, Inc. (616)365-9960
Newark (616)954-6700

Livonia
Arrow Electronics (810)455-0850
Future Electronics (313)261-5270
Hamilton/Hallmark (313)416-5800

Novi
Wyle Electronics (248)374-9953

Saginaw
Newark (517)799-0480

Troy
Newark (248)583-2899

MINNESOTA

Bloomington
Wyle Electronics (612)853-2280

Burnsville
PENSTOCK (612)882-7630

Eden Prairie
Arrow Electronics (612)941-5280
FAI (612)947-0909
Future Electronics (612)944-2200
Hamilton/Hallmark (612)881-2600

MINNESOTA – continued

Minneapolis
Allied Electronics, Inc. (612)938-5633
Newark (612)331-6350

MISSISSIPPI

Jackson
Newark (601)956-3834

MISSOURI

Earth City
Hamilton/Hallmark (314)770-6300

St. Louis
Allied Electronics, Inc. (314)240-9405
Arrow Electronics (314)567-6888
Future Electronics (314)469-6805
FAI (314)542-9922
Newark (314)991-0400

NEBRASKA

Omaha
Allied Electronics, Inc. (402)697-0038
Newark (402)592-2423

NEVADA

Las Vegas
Allied Electronics, Inc. (702)258-1087
Wyle Electronics (702)765-7117

NEW JERSEY

Bridgewater
PENSTOCK (908)575-9490

East Brunswick
Allied Electronics, Inc. (908)613-0828
Newark (732)937-6600

Fairfield
FAI (201)331-1133

Marlton
Arrow Electronics (609)596-8000
FAI (609)988-1500
Future Electronics (609)596-4080

Mt. Laurel
Hamilton/Hallmark (609)222-6400
Wyle Electronics (609)439-9110

Oradell
Wyle Electronics (201)261-3200

Pinebrook
Arrow Electronics (201)227-7880
Wyle Electronics (973)882-8358

Parsippany
Future Electronics (201)299-0400
Hamilton/Hallmark (201)515-1641

NEW MEXICO

Albuquerque
Allied Electronics, Inc. (505)266-7565
Hamilton/Hallmark (505)293-5119
Newark (505)828-1878

NEW YORK

Albany
Newark (518)489-1963

Buffalo
Newark (716)631-2311

Great Neck
Allied Electronics, Inc. (516)487-5211

Hauppauge
Allied Electronics, Inc. (516)234-0485
Arrow Electronics (516)231-1000
FAI (516)348-3700
Future Electronics (516)234-4000
Hamilton/Hallmark (516)434-7400
Newark (516)567-4200
PENSTOCK (516)724-9580
Wyle Electronics (516)231-7850

Henrietta
Wyle Electronics (716)334-5970

Konkoma
Hamilton/Hallmark (516)737-0600

Pittsford
Newark (716)381-4244

Poughkeepsie
Allied Electronics, Inc. (914)452-1470
Newark (914)298-2810

Purchase
Arrow Zeus (914)701-7400

NEW YORK – continued

Rochester
Allied Electronics, Inc. (716)292-1670
Arrow Electronics (716)427-0300
Future Electronics (716)387-9550
FAI (716)387-9600
Hamilton/Hallmark (716)272-2740

Syracuse

Allied Electronics, Inc. (315)446-7411
FAI (315)451-4405
Future Electronics (315)451-2371
Newark (315)457-4873

NORTH CAROLINA

Charlotte
Allied Electronics, Inc. (704)525-0300
FAI (704)548-9503
Future Electronics (704)547-1107
Newark (704)535-5650

Greensboro

Newark (910)294-2142

Morrisville

Wyle Electronics (919)469-1502

Raleigh

Allied Electronics, Inc. (919)876-5845
Arrow Electronics (919)876-3132
FAI (919)876-0088
Future Electronics (919)790-7111
Hamilton/Hallmark (919)872-0712

OHIO

Centerville
Arrow Electronics (513)435-5563

Cincinnati
Allied Electronics, Inc. (513)771-6990
Newark (513)942-8700

Cleveland

Allied Electronics, Inc. (216)831-4900
FAI (216)446-0061
Newark (216)391-9330

Columbus

Allied Electronics, Inc. (614)785-1270
Newark (614)326-0352

Dayton

FAI (513)427-6090
Future Electronics (513)426-0090
Hamilton/Hallmark (513)439-6735
Newark (937)294-8980

Mayfield Heights

Future Electronics (216)449-6996

Miamisburg

Wyle Electronics (937)436-9953

Solon

Arrow Electronics (216)248-3990
Hamilton/Hallmark (216)498-1100
Wyle Electronics (440)248-9996

Toledo

Newark (419)866-0404

Worthington

Hamilton/Hallmark (614)888-3313

OKLAHOMA

Oklahoma City
Newark (405)943-3700

Tulsa

Allied Electronics, Inc. (918)250-4505
FAI (918)492-1500
Hamilton/Hallmark (918)459-6000

OREGON

Beaverton

Arrow/Almac Electronics Corp. . (503)629-8090
Future Electronics (503)645-9454
Hamilton/Hallmark (503)526-6200

Portland

Allied Electronics, Inc. (503)626-9921
FAI (503)297-5020
Newark (503)297-1984
PENSTOCK (503)646-1670
Wyle Electronics (503)598-9953

AUTHORIZED DISTRIBUTORS – continued

UNITED STATES – continued

PENNSYLVANIA

Allentown

Newark (610)434-7171

Chadds Ford

Allied Electronics, Inc. (610)388-8455

Coatesville

PENSTOCK (610)383-9536

Ft. Washington

Newark (215)654-1434

Harrisburg

Allied Electronics, Inc. (717)540-7101

Philadelphia

Allied Electronics, Inc. (609)234-7769

Pittsburgh

Allied Electronics, Inc. (412)931-2774

Arrow Electronics (412)963-6807

Newark (412)788-4790

SOUTH CAROLINA

Greenville

Allied Electronics, Inc. (864)288-8835

Newark (864)288-9610

TENNESSEE

Knoxville

Newark (423)588-6493

Memphis

Newark (901)396-7970

TEXAS

Austin

Allied Electronics, Inc. (512)219-7171

Arrow Electronics (512)835-4180

Future Electronics (512)502-0991

FAI (512)346-6426

Hamilton/Hallmark (512)219-3700

Newark (512)338-0287

PENSTOCK (512)346-9762

Wyle Electronics (512)833-9953

Benbrook

PENSTOCK (817)249-0442

Brownsville

Allied Electronics, Inc. (210)548-1129

Carrollton

Arrow Electronics (972)380-6464

Arrow Zeus (972)380-4330

Dallas

Allied Electronics, Inc. (214)341-8444

FAI (972)231-7195

Future Electronics (972)437-2437

Hamilton/Hallmark (214)553-4300

Newark (972)458-2528

El Paso

Allied Electronics, Inc. (915)779-6294

FAI (915)577-9531

Newark (915)772-6367

Ft. Worth

Allied Electronics, Inc. (817)595-3500

Houston

Allied Electronics, Inc. (281)446-8005

Arrow Electronics (281)647-6868

FAI (713)952-7088

Future Electronics (713)785-1155

Hamilton/Hallmark (713)781-6100

Newark (281)894-9334

Wyle Electronics (713)784-9953

Richardson

PENSTOCK (972)479-9215

Wyle Electronics (972)235-9953

San Antonio

FAI (210)738-3330

UTAH

Draper

Wyle Electronics (801)523-2335

Salt Lake City

Allied Electronics, Inc. (801)261-5244

Arrow Electronics (801)973-6913

FAI (801)467-9696

Future Electronics (801)467-4448

Hamilton/Hallmark (801)266-2022

Newark (801)261-5660

West Valley City

Wyle Electronics (801)974-9953

VIRGINIA

Herndon

Newark (703)707-9010

Richmond

Newark (804)282-5671

Springfield

Allied Electronics, Inc. (703)644-9515

Virginia Beach

Allied Electronics, Inc. (757)363-8662

WASHINGTON

Bellevue

Almac Electronics Corp. (206)643-9992

PENSTOCK (206)454-2371

Bothell

Future Electronics (206)489-3400

Kirkland

Newark (425)814-6230

Redmond

Hamilton/Hallmark (206)882-7000

Wyle Electronics (425)881-1150

Seattle

Allied Electronics, Inc. (206)251-0240

FAI (206)485-6616

Spokane

Newark (509)327-1935

WISCONSIN

Brookfield

Arrow Electronics (414)792-0150

Future Electronics (414)879-0244

Wyle Electronics (414)879-0434

Madison

Newark (608)278-0177

Milwaukee

Allied Electronics, Inc. (414)796-1280

FAI (414)792-9778

New Berlin

Hamilton/Hallmark (414)780-7200

Wauwatosa

Newark (414)453-9100

CANADA

ALBERTA

Calgary

FAI (403)291-5333

Future Electronics (403)250-5550

Hamilton/Hallmark (800)663-5500

Newark (800)463-9275

Edmonton

FAI (403)438-5888

Future Electronics (403)438-2858

Hamilton/Hallmark (800)663-5500

Newark (800)463-9275

Saskatchewan

Hamilton/Hallmark (800)663-5500

BRITISH COLUMBIA

Vancouver

Allied Electronics, Inc. (604)420-9691

Arrow Electronics (604)421-2333

FAI (604)654-1050

Future Electronics (604)294-1166

Hamilton/Hallmark (604)420-4101

Newark (800)463-9275

MANITOBA

Winnipeg

FAI (204)786-3075

Future Electronics (204)944-1446

Hamilton/Hallmark (800)663-5500

Newark (800)463-9275

ONTARIO

Kanata

PENSTOCK (613)592-6088

London

Newark (519)685-4280

Mississauga

PENSTOCK (905)403-0724

Newark (905)670-2888

Ottawa

Allied Electronics, Inc. (613)228-1964

Arrow Electronics (613)226-6903

FAI (613)820-8244

Future Electronics (613)727-1800

Hamilton/Hallmark (613)226-1700

Toronto

Arrow Electronics (905)670-7769

FAI (905)612-9888

Future Electronics (905)612-9200

Hamilton/Hallmark (905)564-6060

Newark (905)670-2888

QUEBEC

Montreal

Arrow Electronics (514)421-7411

FAI (514)694-8157

Future Electronics (514)694-7710

Hamilton/Hallmark (514)335-1000

Mt. Royal

Newark (514)738-4488

Quebec City

Arrow Electronics (418)687-4231

FAI (418)682-5775

Future Electronics (418)877-6666

INTERNATIONAL DISTRIBUTORS

ARGENTINA

Electrocomponentes (5-41) 375-3366
Elko (5-41) 372-1101

AUSTRALIA

Avnet VSI Electronics (Aust.) (61)2 9878-1299
Farnell (61)2 9645-8888
Veltek Australia Pty. Ltd. (61)3 9574-9300

AUSTRIA

EBV Elektronik (43) 189152-0
Farnell (49) 8961 393939
SEI/Elbatex GmbH (43) 1 866420
Spoerle Electronic (43) 1 360460

BELGIUM

EBV Elektronik (32) 2 716 0010
Farnell (32) 3 227 3647
SEI/Belgium (32) 2 460 0747
Spoerle Electronic (32) 2 725 4660

BRAZIL

Farnell (5511) 445-7400
Future (019) 235-1511
Intertek (011) 266-2922
Karimex (011) 524-2366
Masktrade (011) 3361-2766
Panamericana (011) 223-0222
Siletek (011) 536-4401
Tec (011) 5505-2046
Teleradio (011) 574-0788

BULGARIA

Macro Group (359) 2708140

CHINA

Arrow Asia/Pac Ltd (852)2 484-2113
Avnet WKK Components Ltd. (852)2 357-8888
China El. App. Corp. Beijing (86)10 6828-9951
Future Advanced Electronics Ltd. . (852)2 305-3633
Nanco Electronics Supply Ltd. . (852)2 765-3025
Qing Cheng Enterprises Ltd. . (852)2 493-4202

CZECH REPUBLIC

EBV Elektronik (420) 2 90022101
Spoerle Electronic (420) 2 71737173
SEI/Elbatex (420) 2 4763707
Macro Group (420) 2 3412182

DENMARK

Arrow Denmark A/S (45) 44 508200
A/S Avnet EMG (45) 44 880800
EBV Elektronik - Soeborg (45) 39690511
EBV Elektronik - Abyhoj (45) 86250466
Future Electronics (45) 961 00 961

ESTONIA

Arrow Field Eesti (372) 6503288
Avnet Baltronic (372) 6397000

FINLAND

Arrow Finland (358) 9 476660
Avnet Nortek (358) 9 613181
EBV Elektronik (358) 9 8557730
Future Electronics (358) 9 345 5400

FRANCE

Arrow Electronique (33) 1 49 78 49 78
Avnet (33) 1 49 65 27 00
EBV Elektronik (33) 1 40963000
Farnell (33) 474 659466
Future Electronics (33) 1 69821111
Newark (33) 1 30954060
Sonepar Electronique (33) 1 69 19 89 00

GERMANY

Avnet EMG (49) 89 4511001
EBV Elektronik GmbH (49) 89 99114-0
Farnell (49) 89 61 393939
Future Electronics GmbH (49) 89-957 270
SEI/Jermyn GmbH (49) 6431-5080
Newark (49)2154-70011
Sasco Semiconductor (49) 89-46110
Spoerle Electronic (49) 6103-304-0

GREECE

EBV Elektronik (30) 13414300

HONG KONG

Avnet WKK Components Ltd. (852)2 357-8888
Farnell (65) 788-0200
Future Advanced Electronics Ltd. . (852)2 305-3633
Nanco Electronics Supply Ltd. . (852)2 333-5121
Qing Cheng Enterprises Ltd. . (852)2 493-4202

HUNGARY

EBV Elektronik KFT (36) 1 4313 495
Future Electronics (36) 1 2240 510
Macro Group (36) 1 2030 277
SEI/Elbatex (36) 1 1409 194
Spoerle Electronic (36) 1 1294 202

INDIA

Max India Ltd 0091 11 625-0250

INDONESIA

P.T. Ometraco (62) 21 619-6166

IRELAND

Arrow Electronics (353) 14595540
EBV Elektronik (353) 14564034
Farnell (353) 18309277
Future Electronics (353) 6541330
Macro Group (353) 16766904

ISRAEL

Future Israel Ltd. (972) 9 9586555

ITALY

Avnet EMG (39) 02 381901
EBV Elektronik (39) 02 66096290
Future Electronics (39) 02 660941
Silverstar LTD (39) 02 661251

JAPAN

AMSC Co., Ltd. 81-422-54-6800
Fuji Electronics Co., Ltd. 81-3-3814-1411
Marubun Corporation 81-3-3639-8951
OMRON Corporation 81-3-3779-9053
Tokyo Electron Device Ltd. . 81-45-474-7030

KOREA

Jung Kwang Semiconductors Ltd. . 82-2-278-5333
Liteon Korea Ltd 82-2-650-9700
Nasco Co. Ltd 82-2-3772-6810

LATVIA

Avnet Baltronic Ltd. (371) 8821118
Macro Group (371) 7313195

LITHUANIA

Macro Group (370) 7764937

MALAYSIA

Farnell (60) 3 773-8000
Strong Electronics (60) 4 656-3768
Ultron Technologies Pte. Ltd. (65) 545-7811

MEXICO

Avnet (3) 632-0182
Dicopel (5) 705-7422
Future (3) 122-0043
Semiconductores Profesionales (5) 658-6011
Stereon (5) 325-0925

NETHERLANDS

HOLLAND

EBV Elektronik (31) 3465 83010
Farnell (31) 30 241 2323
Future Electronics (31) 76 544 4888
SEI/Benelux B.V. (31) 7657 22500
Spoerle Electronics -
Nieuwegein (31) 3060 91234
Spoerle Electronics -
Veldhoven (31) 4025 45430

NEW ZEALAND

Arrow Components NZ Ltd (64)4 570-2260
Avnet Pacific Ltd (64)9 636-7801
Farnell (64)9 357-0646

NORWAY

Arrow Tahonic A/S (47) 2237 8440
A/S Avnet EMG (47) 6677 3600
EBV Elektronik (47) 2267 1780
Future Electronics (47) 2290 5800

PHILIPPINES

Alexan Commercial (63) 2241-9493
Ultron Technologies Pte. Ltd. (65) 545-7811

POLAND

EBV Elektronik (48) 713 422944
Future Electronics (48) 22 61 89202
Macro Group (48) 22 224337
SEI/Elbatex (48) 22 6217122
Spoerle Electronic (48) 22 6465227

PORTUGAL

Amitron Arrow (35) 11471 4182
Farnell (44) 113289 0040
SEI/Selco (35) 12973 8203

ROMANIA

Macro Group (401) 6343129

RUSSIA

EBV Elektronik (7) 095 9761176
Macro Group - Moscow (7) 095 30600266
Macro Group - St. Petersburg (7) 81 25311476

SCOTLAND

EBV Elektronik (44) 141 4202070
Future (44) 141 9413999

SINGAPORE

Farnell (65) 788-0200
Future Electronics (65) 479-1300
Strong Pte. Ltd (65) 276-3996
Uraco Technologies Pte Ltd. (65) 545-7811

SLOVAKIA

Macro Group (42) 89634181
SEI/Elbatex (42) 17295007

SLOVENIA

EBV Elektronik (386) 611 330216
SEI/Elbatex (386) 611 597198

S. AFRICA

Avnet-ASD (27) 11 4442333
Reutech Components (27) 11 3972992

SPAIN

Amitron Arrow (34) 91 304 3040
EBV Elektronik (34) 91 804 3256
Farnell (44) 113 231 0447
SEI/Selco S.A. (34) 1 637 10 11

SWEDEN

Arrow-Th:s AB (46) 8 56265500
Avnet EMG AB (46) 8 629 14 00
EBV Elektronik (46) 405 92100
Farnell (46) 8 730 5000
Future Electronics (46) 8 441 5470

SWITZERLAND

EBV Elektronik (41) 1 7456161
Farnell (41) 1204 6464
SEI/Elbatex AG (41) 56 4375111
Spoerle Electronic (41) 1 8746262

TAIWAN

Avnet-Mercuries Co., Ltd. (886)2 516-7303
Solomon Technology Corp. . (886)2 788-8989
Strong Electronics Co. Ltd. . (886)2 917-9917

THAILAND

Sahapiphat Ltd. (662) 237-9474
Ultron Technologies Pte. Ltd. (65) 540-8328

TURKEY

EBV Elektronik (90) 216 4631352

UNITED KINGDOM

Arrow Electronics (UK) Ltd . (44) 1 234 270027
Avnet EMG (44) 1 438 788300
EBV Elektronik (44) 1 628 783688
Farnell (44) 1 132 636311
Future Electronics Ltd. (44) 1 753 763000
Macro Group (44) 1 628 606000
Newark (44) 1 420 543333

MOTOROLA WORLDWIDE SALES OFFICES

UNITED STATES

ALABAMA

Huntsville (205)464-6800

ALASKA (800)635-8291

ARIZONA

Phoenix (602)302-8056

CALIFORNIA

Calabasas (818)878-6800

Irvine (714)753-7360

Los Angeles (818)878-6800

San Diego (619)541-2163

Sunnyvale (408)749-0510

COLORADO

Denver (303)337-3434

CONNECTICUT

Wallingford (203)949-4100

FLORIDA

Clearwater (813)524-4177

Maitland (407)628-2636

Pompano Beach/Ft. Lauderdale (954)351-6040

GEORGIA

Atlanta (770)729-7100

IDAHO

Boise (208)323-9413

ILLINOIS

Chicago/Schaumburg (847)413-2500

INDIANA

Indianapolis (317)571-0400

Kokomo (765)455-5100

KANSAS

Kansas City/Mission (913)451-8555

MARYLAND

Columbia (410)381-1570

MASSACHUSETTS

Marlborough (508)357-8207

Woburn (781)932-9700

MICHIGAN

Detroit (248)347-6800

MINNESOTA

Minnetonka (612)932-1500

MISSOURI

St. Louis (314)275-7380

NEW JERSEY

Fairfield (973)808-2400

NEW YORK

Fairport (716)425-4000

Fishkill (914)896-0511

Hauppauge (516)361-7000

NORTH CAROLINA

Raleigh (919)870-4355

OHIO

Cleveland (440)349-3100

Columbus/Worthington (614)431-8492

Dayton (937)438-6800

OREGON

Portland (503)641-3681

PENNSYLVANIA

Colmar (215)997-1020

Philadelphia/Horsham (215)957-4100

TENNESSEE

Knoxville (423)584-4841

TEXAS

Austin (512)502-2100

Houston (281)251-0006

Plano (972)516-5100

WASHINGTON

Bellevue (425)454-4160

Seattle (toll free) (206)622-9960

WISCONSIN

Milwaukee/Brookfield (414)792-0122

Field Applications Engineering Available
Through All Sales Offices

CANADA

ALBERTA

Calgary (403)216-2190

BRITISH COLUMBIA

Vancouver (604)606-8502

ONTARIO

Ottawa (613)226-3491

Mississauga (905)501-3500

QUEBEC

Montreal (514)333-3300

INTERNATIONAL

AUSTRALIA

Melbourne (61-3)9887 0711

Sydney (61-2)9437 8944

BRAZIL

Sao Paulo 55(011)3030-5244

CHINA

Beijing 86-10-65642288

Guangzhou 86-20-87537888

Shanghai 86-21-63747668

Tianjin 86-22-25325050

CZECH REPUBLIC

..... (420) 2 21852222

FINLAND

Helsinki (358) 9 6866 880

Direct Sales Lines (358) 9 6866 8844

..... (358) 9 6866 8845

FRANCE

Paris 33134 635900

GERMANY

Langenhagen/Hanover 49(511)786880

Munich 49 89 92103-0

Nuremberg 49 911 96-3190

Sindelfingen 49 7031 79 710

Wiesbaden 49 611 973050

HONG KONG

Kwai Fong 852-2-610-6888

Tai Po 852-2-666-8333

HUNGARY

..... (36) 1 250 83 29

INDIA

Bangalore 91-80-5598615

ISRAEL

Herzlia 972-9-9522333

ITALY

Milan 39(2)82201

JAPAN

Kyusyu 81-92-725-7583

Gotanda 81-3-5487-8311

Nagoya 81-52-232-3500

Osaka 81-6-305-1801

Sendai 81-22-268-4333

Takamatsu 81-878-37-9972

Tokyo 81-3-3440-3311

KOREA

Pusan 82(51)4635-035

Seoul 82-2-3440-7200

MALAYSIA

Penang 60(4)228-2514

MEXICO

Chihuahua 52(14)39-3120

Mexico City 52(5)282-0230

Guadalajara 52(36)78-0750

Zapopan Jalisco 52(36)78-0750

Marketing 52(36)21-2023

Customer Service 52(36)669-9160

NETHERLANDS

Best (31)4993 612 11

PHILIPPINES

Manila (63)2 807-8455

Paranaque (63)2 824-4551

Salcedo Village (63)2 810-0762

POLAND

..... (48) 34 27 55 75

PUERTO RICO

Rio Piedras (787)282-2300

RUSSIA

..... (7) 095 929 90 25

SCOTLAND

East Kilbride (44)1355 565447

SINGAPORE (65)4818188

SPAIN

Madrid 34(1)457-8204

or 34(1)457-8254

SWEDEN

Solna 46(8)734-8800

SWITZERLAND

Geneva 41(22)799 11 11

Zurich 41(1)730-4074

TAIWAN

Taipei 886(2)717-7089

THAILAND

Bangkok 66(2)254-4910

TURKEY

..... (90) 212 274 66 48

UNITED KINGDOM

Aylesbury 44 1 (296)395252

NORTH AMERICA

FULL LINE REPRESENTATIVES

ARIZONA, Tempe

S&S Technologies, Inc. (602)414-1100

CALIFORNIA, Loomis

Galena Technology Group (916)652-0268

INDIANA, Indianapolis

Bailey's Electronics (317)848-9958

NEVADA, Clark County

S&S Technologies, Inc. (602)414-1100

NEVADA, Reno

Galena Tech. Group (702)746-0642

NEW MEXICO, Albuquerque

S&S Technologies, Inc. (602)414-1100

TEXAS, El Paso

S&S Technologies, Inc. (915)833-5461

UTAH, Salt Lake City

Utah Comp. Sales, Inc. (801)572-4010

WASHINGTON, Spokane

Doug Kenley (509)924-2322

NORTH AMERICA

HYBRID/MCM COMPONENT SUPPLIERS

Chip Supply (407)298-7100

Elmo Semiconductor (818)768-7400

Minco Technology Labs Inc. (512)834-2022

Semi Dice Inc. (310)594-4631

