MC143150
MC143120

Advance Information

NEURON® CHIP Distributed Communications and Control Processors
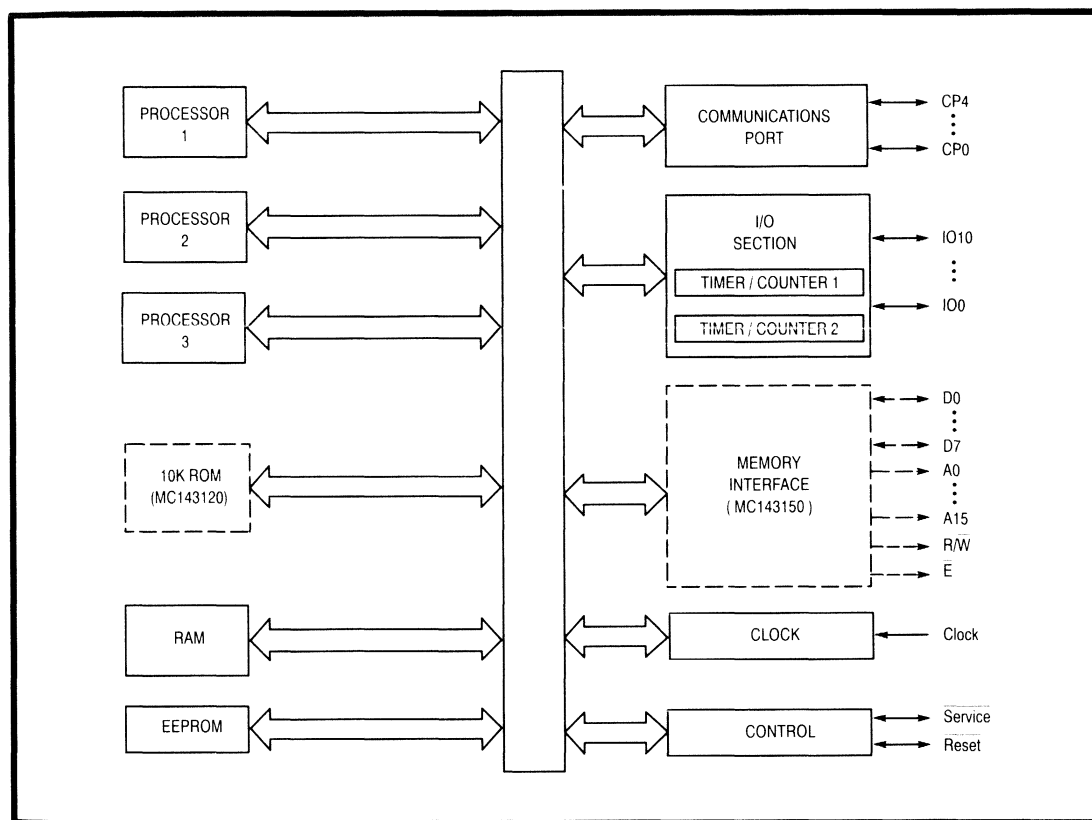
Ⓜ MOTOROLA ▪

REV 1
3/92

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, and NEURON are registered trademarks of Echelon Corporation. LONWORKS, LONBUILDER, LONMANAGER, LONTALK, and NEUROWIRE are trademarks of Echelon Corporation.

CONTENTS

CONTENTS (Continued)

CONTENTS (Continued)

CONTENTS (Concluded)

LIST OF FIGURES

LIST OF FIGURES
(Concluded)

LIST OF TABLES

SECTION 1
INTRODUCTION

The Motorola MC143150 and MC143120 NEURON CHIPS are sophisticated 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 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
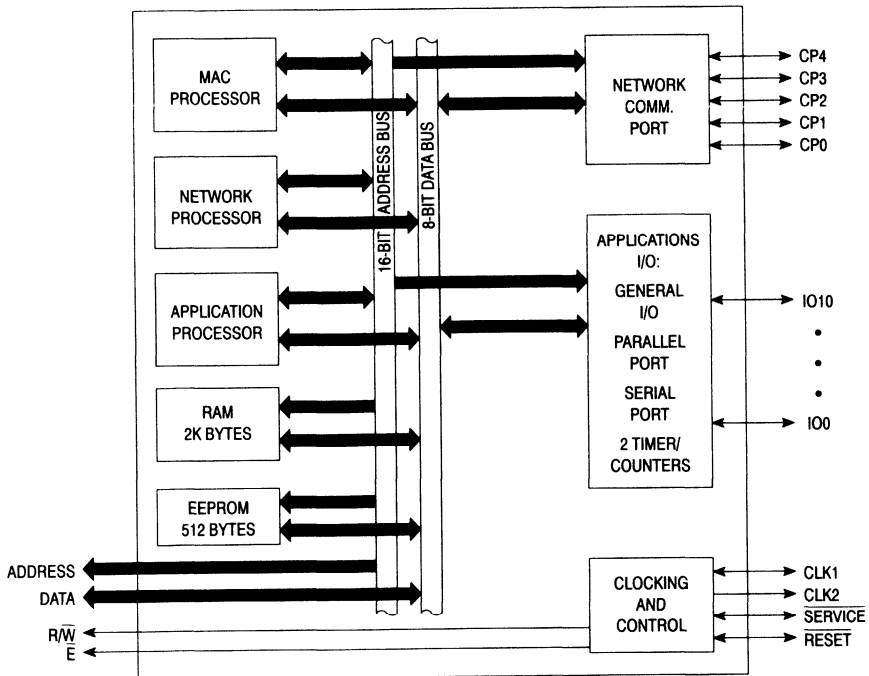
Figure 1.1. MC143150 NEURON CHIP Simplified Block Diagram
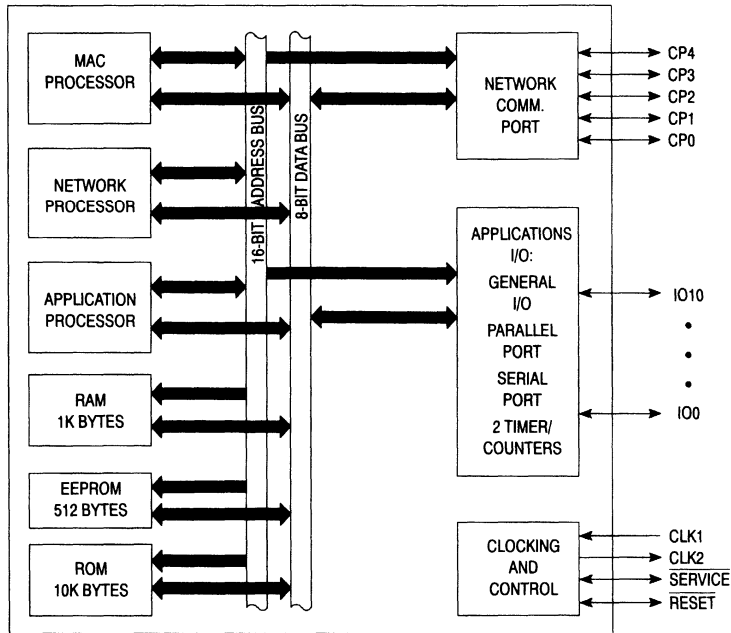
Figure 1.2. MC143120 NEURON CHIP Simplified Block Diagram

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 chip. The communications protocol, operating system and 24 I/O function object code is provided with the LONBUILDER starter kit (MC143160EVK). The protocol and application code can be located in external ROM, EEPROM, NVRAM, or battery-backup static RAM.

The MC143120 has no external memory interface and is designed for applications that require smaller application programs. It contains 10K of masked ROM that implements the communications protocol, operating system and the 24 I/O functions that can be accessed by the application program. The application program resides in the internal 512 bytes of EEPROM and utilizes the firmware in the masked ROM for the specific application.

Both parts have an eleven pin I/O interface with integrated hardware and software 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.

SECTION 2
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 Developer's Workbench

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.
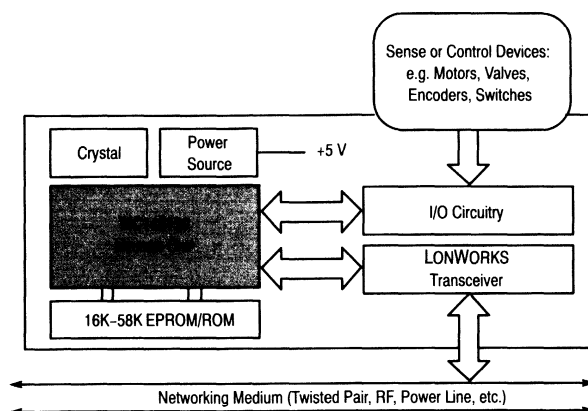
Figure 2.1. MC143150 in a Typical Node Block Diagram
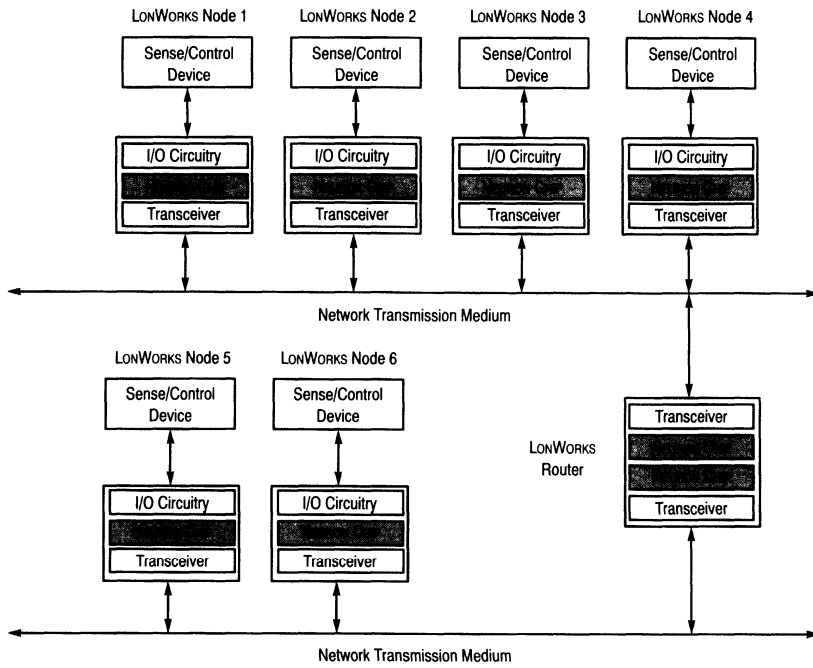
Figure 2.2. The MC143150 or MC143120 in a LONWORKS Network

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 (Figure 3.1) supports external memory for more complex applications, while the MC143120 (Figure 3.2) is a complete system on a chip that supports many applications. The major hardware blocks of both processors are the same, except where noted; see Table 3.1.

Table 3.1. Comparison of MC143150 and MC143120 Processors

Characteristic	MC143150	MC143120
CPUs	3	3
RAM Bytes	2,048	1,024
ROM Bytes	—	10,240
EEPROM Bytes	512	512
16-Bit Timer/Counters	2	2
External Memory Interface	Yes	No
Package	PQFP and PQCC	SOG
Pins	64/68	32

3.1 PROCESSING UNITS

The three identical 8-bit state machines form a symmetric multiprocessor, each one executing a different microcycle within a three-phase minor cycle at any one time. Each has its own register set (Table 3.2), but all three share 8-bit data and address ALUs and memory access circuitry. This reduces die size without affecting performance.

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
SAR	8	Signature Analysis Register

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 3.15, 3.16, and 3.17 outline the instruction set.

A0 — 64
A1 — 63
A2 — 62
A3 — 61
A4 — 60
A5 — 59
A6 — 58
A7 — 57
A8 — 56
A9 — 55
A10 — 54
A11 — 53
A12 — 52
A13 — 51
A14 — 50
A15 — 47
D0 — 43
D1 — 42
D2 — 38
D3 — 37
D4 — 36
D5 — 35
D6 — 34
D7 — 33
R/W̄ — 45
Ē — 46

Memory Expansion Bus

Processor 1

Processor 2

Processor 3

A0–A15

D0–D7

2 / Control

RAM 2K X 8

EEPROM 512 X 8

16 / 8 /

Internal Address Bus

Internal Data Bus

Internal Control Bus

Communi-cations Port

28 — CP0
29 — CP1
30 — CP2
31 — CP3
32 — CP4

Input/Output Section

Two Timer/Counters

2 — IO0
3 — IO1
4 — IO2
5 — IO3
10 — IO4
11 — IO5
12 — IO6
13 — IO7
14 — IO8
15 — IO9
16 — IO10

Clock and Control Section

24 — Clock 1
23 — Clock 2
17 — Servicē
6 — Reset̄

NOTE: Pin numbers apply to 64-lead package.

Figure 3.1. MC143150
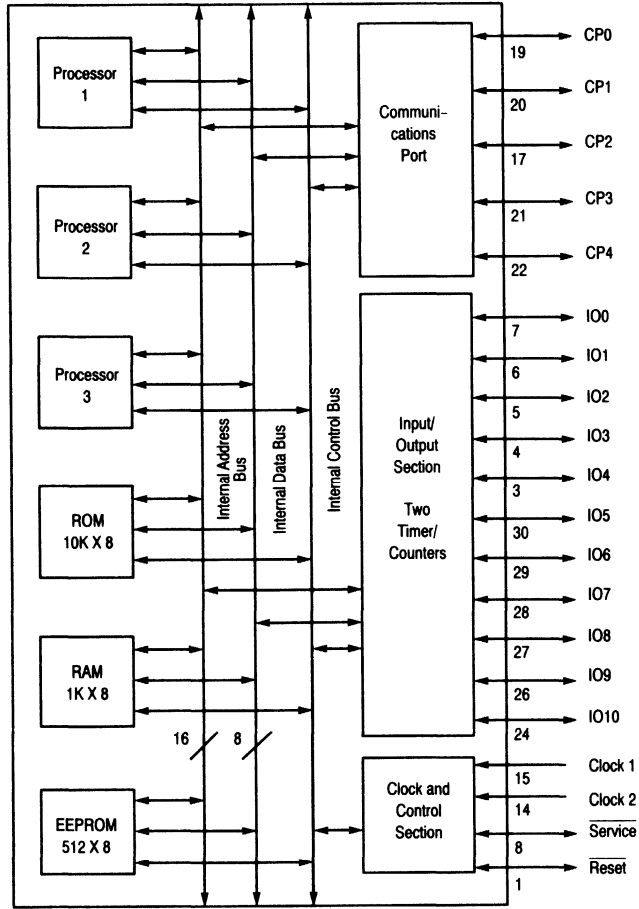
Figure 3.2. MC143120

Figure 3.3 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.

	Return Stack	
BP+RSP		
BP+DSP	NEXT	- - - TOS
BP+0x18	Data Stack	
BP+0x17	16 Directly-Addressable Bytes	
BP+0x8		
BP+0x7	Four Two-Byte Pointer Registers	
BP		
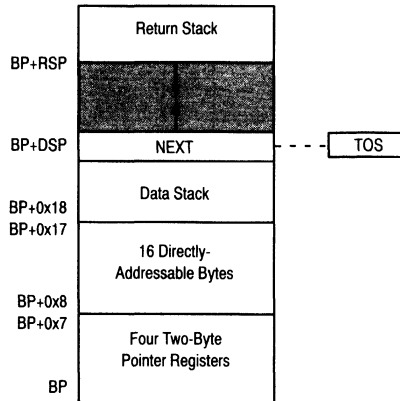
Figure 3.3. Base Page Memory Layout

3.2 MEMORY

3.2.1 EEPROM

Both the MC143150 and the MC143120 have 512 bytes in-circuit programmable EEPROM. All but eight bytes of the EEPROM can be written under program control using an on-chip charge pump to generate the required programming voltage. The remaining eight 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 time and write time are each 10 ms per byte. The EEPROM may be written up to 10,000 times with no data loss. 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. The application code for the MC143150 may either be stored on-chip, or off-chip in external memory. 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, and while there is built-in protection to prevent EEPROM corruption during power-down, it is important to hold the $\overline{\text{RESET}}$ pin low whenever V_{DD} is below its minimum operating level to avoid this possibility (see Section 3.6.3).

3.2.2 Static RAM

The MC143150 has 2048 bytes of static RAM and the MC143120 has 1024 bytes of static RAM. The RAM state is retained as long as power is applied to the device, even in "sleep" mode.

3.2.3 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 development system.

3.2.4 External Memory Interface (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 maps are shown in Figure 4.2 and Figure 4.3. The bus has eight bidirectional data lines, and sixteen address lines driven by the processor. Two interface lines (R/W̄ and Ē) are used for external memory access (see Figure 3.2). At the maximum clock rate (10 MHz input clock), 70 ns or better access time to external memory is required. If the input clock is scaled down, slower memory can be used. The allowable input clock rates are 10 MHz, 5 MHz, 2.5 MHz, 1.25 MHz, and 625 kHz. The Enable Clock (Ē) 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 shows diagrams of interfacing the MC143150 to different types of memory. A minimum hardware configuration would use one external EPROM containing both the protocol code and user application code (see Figure 3.4). 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, NVRAM or battery backed up SRAM are required if the user code will not fit in the 512 bytes of internal EEPROM. Also 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 3.3. Timing information is listed in Table 3.4 and is measured relative to the rising edge of Ē clock. The Ē clock signal is not typically used for *reading* external memory but can be gated with decoded address lines and the R/W̄ signal to generate WRITE signals to external memory. A15 (address line 15) or a PAL decoded signal gated with R/W̄ can be used to generate read signals to external memory. The MC143150 needs to see data (setup time) 65 ns ahead of the rising edge of Ē clock. The data hold time required when reading data is 0 ns.
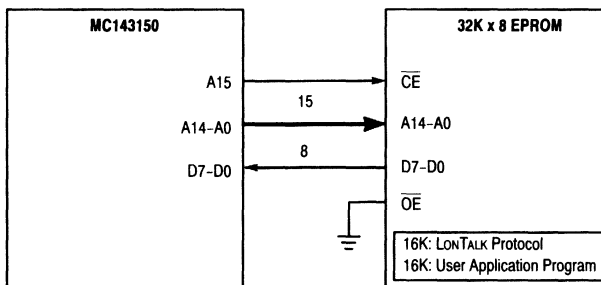
Figure 3.4. Minimum MC143150 Memory Interface

Table 3.3. External Memory Interface Pins

Pin Designation	Direction	Function
A0–A15	Output	Address pins
D0–D7	Input/Output	Data pins
Ē	Output	Enable clock
R/W̄	Output	Read/not Write select

Because one of the internal processors is always using the address bus, sharing the address and data bus (and memory) with another MPU's 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.

Table 3.4. External Memory Bus Timing* (V_{DD} = 4.5 to 5.5 V, T_A = − 40° to 85°C)

Symbol	Parameter	Min	Max	Unit
t_{cyc}	Memory Cycle Time (System Clock Period) (Note 1)	200	3200	ns
PW_{EH}	Pulse Width, $\overline{E}$ High	$t_{cyc}/2 - 5$	$t_{cyc}/2 + 5$	ns
PW_{EL}	Pulse Width, $\overline{E}$ Low	$t_{cyc}/2 - 5$	$t_{cyc}/2 + 5$	ns
t_{AD}	Delay, $\overline{E}$ High to Address Valid	—	55	ns
t_{AH}	Address Hold Time	5	—	ns
t_{RD}	Delay, $\overline{E}$ High to R/$\overline{W}$ Valid Read	—	25	ns
t_{RH}	R/$\overline{W}$ Hold Time Read	5	—	ns
t_{DSR}	Read Data Setup Time	65	—	ns
t_{DZR}	Delay Data Bus High-Z to R/$\overline{W}$ High	5	—	ns
t_{DHR}	Data Hold Time Read	0	—	ns
t_{WR}	Delay, $\overline{E}$ High to R/$\overline{W}$ Valid Write	—	25	ns
t_{WDD}	Delay, R/$\overline{W}$ Low to Data Drivers On (Note 2)	10	—	ns
t_{DDW}	Delay, $\overline{E}$ Low to Data Valid	—	67	ns
t_{WH}	R/$\overline{W}$ Hold Time Write	5	—	ns

NOTES:
*   * All values are preliminary and subject to change.
    1. t_{cyc} = 2•1/f, where f is the input clock frequency.
    2. See Figure 3.5. The NEURON CHIP drives the previously read data until after the falling edge of $\overline{E}$. Therefore, an external memory and the NEURON CHIP may both be driving the data lines to the same levels during this time without contention.

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-ups. They are enabled or disabled with a compiler directive (see NEURON C Programmers 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 to IO10) have TTL-level inputs with hysteresis. Pins IO0 to IO7 also have low level detect latches.

Figure 3.5. External Memory Interface Timing Diagram
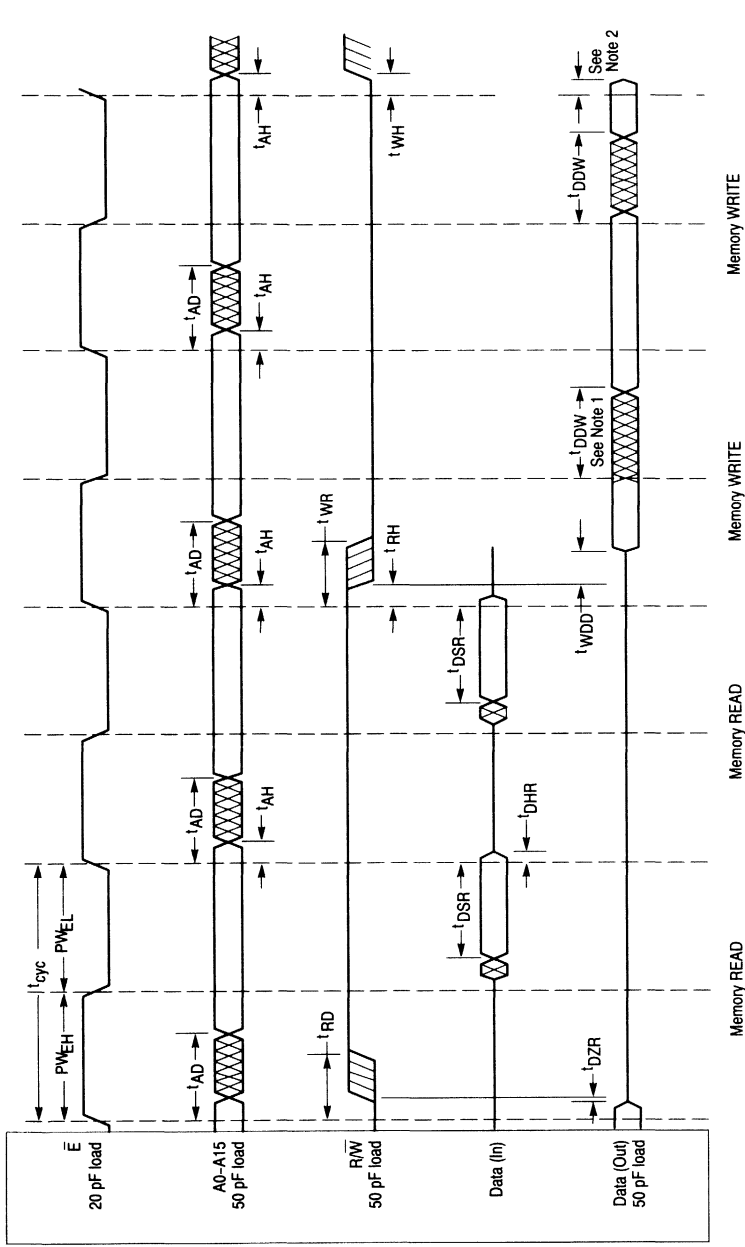
NOTES:
1. The Neuron Chip drives the previously read data until after the falling edge of E̅. Therefore, an external memory and the Neuron Chip may both be driving the data lines to the same levels during this time without contention.
2. Since the data bus goes high-Z at the beginning of a read cycle, the effective data hold time is dependent on the current load on the bus. Typically this will be >>50 ns.

3.3.2 Two 16-Bit Timer/Counters

The timer/counters are implemented as a load register writeable by the processor, a 16-bit counter, and a latch readable by the processor. The 16-bit registers are accessed a byte at a time. Both the MC143150 and MC143120 have one timer/counter whose input is selectable among pins IO4 through IO7, and whose output is pin IO0, and a second timer/counter with input from pin IO4 and output to pin IO1 (Figure 3.6). 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.
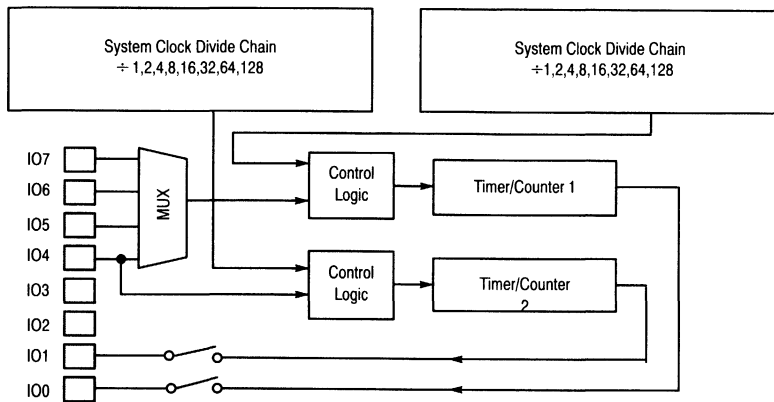
Figure 3.6. Timer/Counter Circuits

3.4 NETWORK COMMUNICATIONS

The five-pin communications port is the network connection that interfaces the processor to a wide variety of media interfaces (network transceivers). The communications port may be configured to operate in one of two modes: Direct Mode (single ended or differential) or Special Purpose Mode. See Table 3.5 and Figure 3.7.

Table 3.5. Communications Port Pin Characteristics

Pin	Drive Current (mA)	Direct Mode Differential	Direct Mode Single-Ended	Special Purpose Mode
CP0	1.4	RX+(in)	RX(in)	RX(in)
CP1	1.4	RX–(in)	TX(out)	TX(out)
CP2	35	TX+(out)	TX Enable (out)	Bit Clock (out)
CP3	35	TX–(out)	Sleep (out)	Sleep (out) or Wake Up (out)
CP4	1.4	CDet(in)	CDet (in)	Frame Clock (out)

RX—Receiver   TX—Transmitter   CDet—Collision Detect
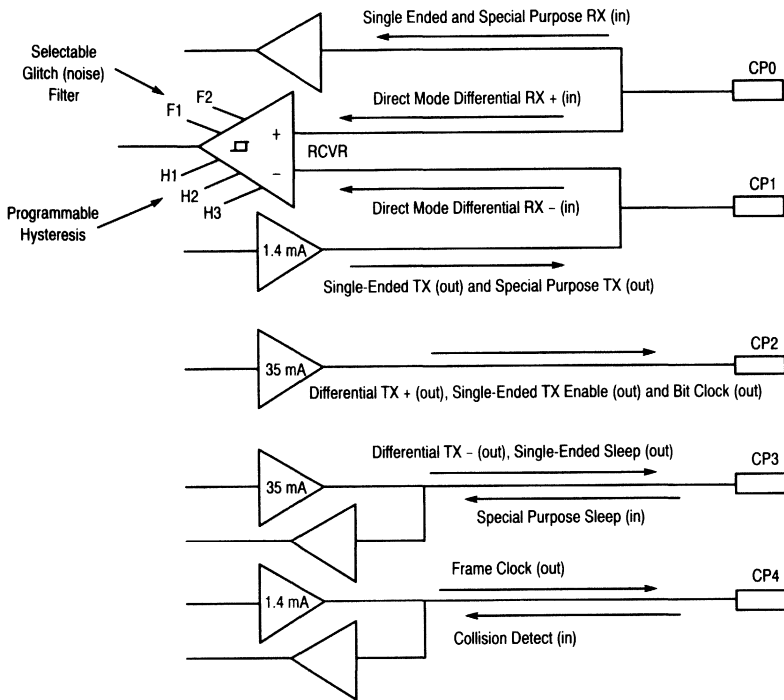
Figure 3.7. Internal Transceiver Block Diagram

3.4.1 Direct Mode

In Direct mode, the processor communicates with the transceiver using a stream of Differential Manchester encoded data (see Figure 3.8). This guarantees a transition in every bit period for purposes of synchronizing the receiver clock. Zero/one data is indicated by the presence/absence (respectively) of a second transition halfway between clock transitions. This mode is polarity insensitive.
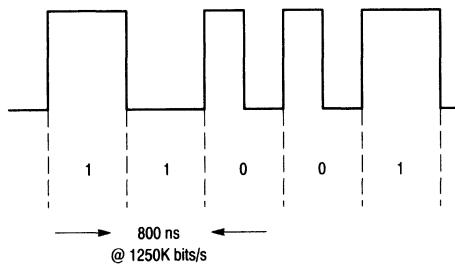
Figure 3.8. Direct Mode Differential Manchester Data Encoding

Table 3.6. Single-Ended and Differential Network Data Rates

Network Bit Rate (kbps)	Minimum Input Clock (MHz)	Maximum Input Clock (MHz)
1,250	10.0	10.0
625	5.0	10.0
312.5	2.5	10.0
156.3	1.25	10.0
78.1	0.625	10.0
39.1	0.625	10.0
19.5	0.625	10.0
9.8	0.625	10.0
4.9	0.625	5.0

3.4.1.1 DIRECT MODE SINGLE ENDED. 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 3.9 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 3.6, as a function of the NEURON CHIP input clock rate.
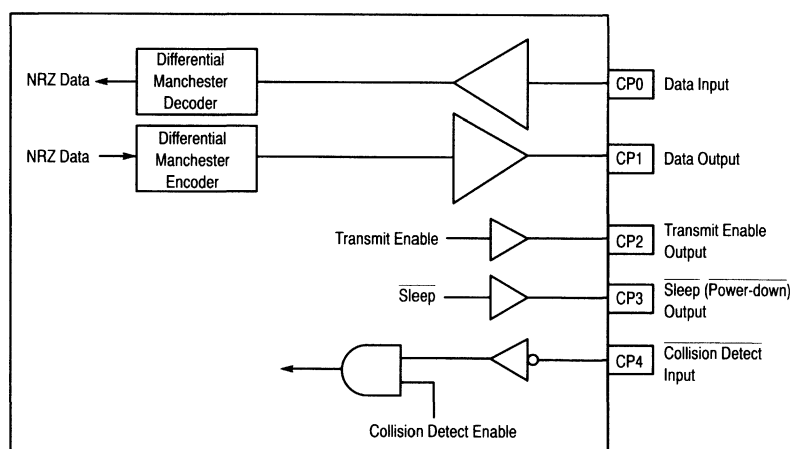
Figure 3.9. Single-Ended Mode Configuration

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 guarantees a transition at the beginning of every bit period for the purpose of synchronizing the receiver clock. Zero/one 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 3.10 shows a typical packet where T is the bit period, equal to 1/(bit rate).

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 1's. Its duration is at least six bits long and is selectable by the user.
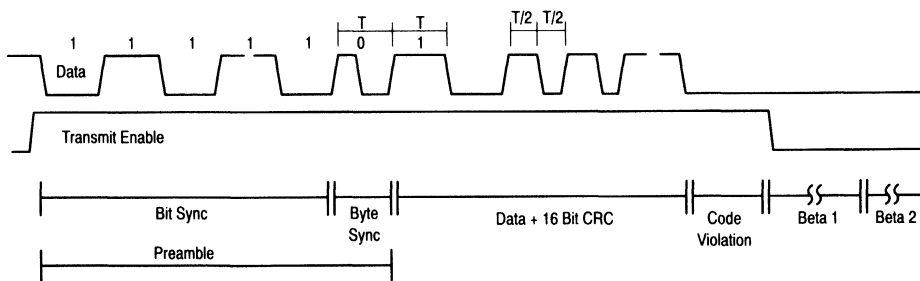
Figure 3.10. Single-Ended Mode Data Format

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 2-1/2 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 to 44.8 ms depending on the input clock rate (10 MHz to 625 kHz).

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 resent. The node then attempts to reaccess the channel.

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 acknowledgement. When acknowledged service is used, retry timer is set to allow sufficient time for a message to be sent and acknowledged (typically 48 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.

Beta 1 time is the idle period after a packet has been sent.

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•16 where "n" has a range of 1 to 63). See Figure 3.11.
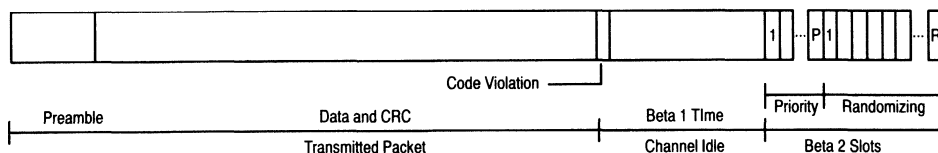
Figure 3.11 Packet Timing

When put in direct single-ended mode the NEURON CHIP can interface to an external EIA-485 transceiver IC. See Figure 3.12. While not having the performance of a transformer coupled circuit (Section 3.4.1.2) this can offer a more economical solution. EIA-485 chips can be found in bipolar and CMOS versions. CMOS uses less power but is usually more expensive. The communications port must be put in single-ended mode for implementing an EIA-485 network. EIA-485 specifies 32 drivers and 32 receivers on a single twisted pair of wires. Many wire types can be used with this approach but collision detection may be difficult to implement.
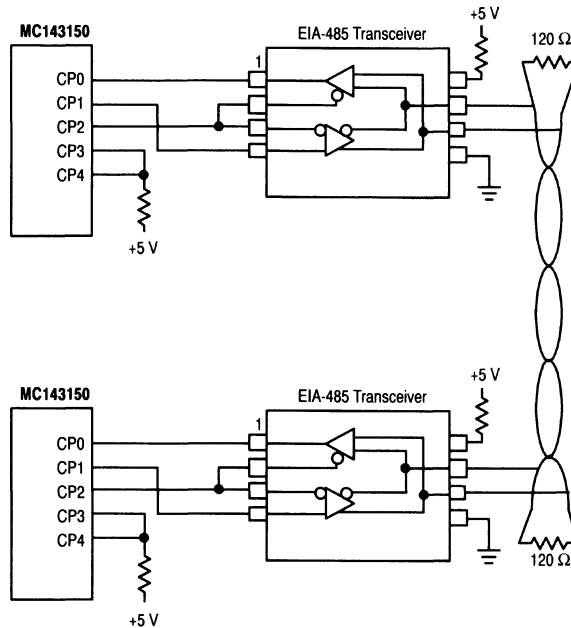
Figure 3.12. EIA-485 Twisted Pair Interface
(Used with single-ended mode)

3.4.1.2 DIRECT MODE DIFFERENTIAL (DIRECT CONNECT). 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 3.13. The selectable hysteresis and filter permit optimizing receiver performance to line conditions. See Tables 3.7 and 3.8 for specific values.

Figure 3.14 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 35 mA of current at 4.2 V and can sink 35 mA of current at 0.8 V (V_{DD} = 5 V). The programmable hysteresis values shown in Table 3.7 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. See Tables 3.8 and 3.9.
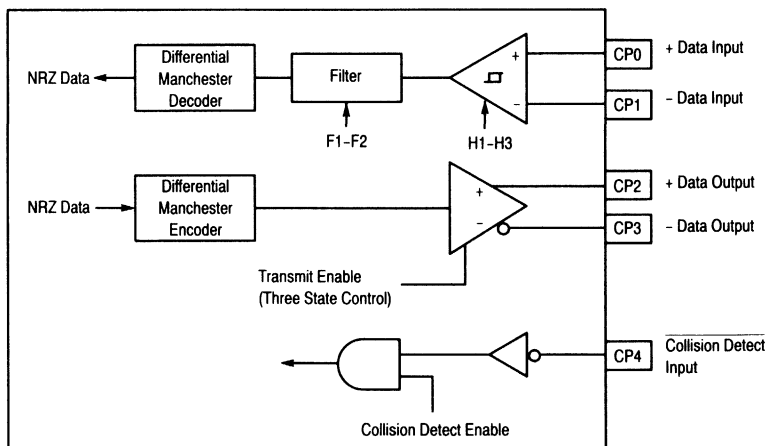
Figure 3.13. Differential Mode
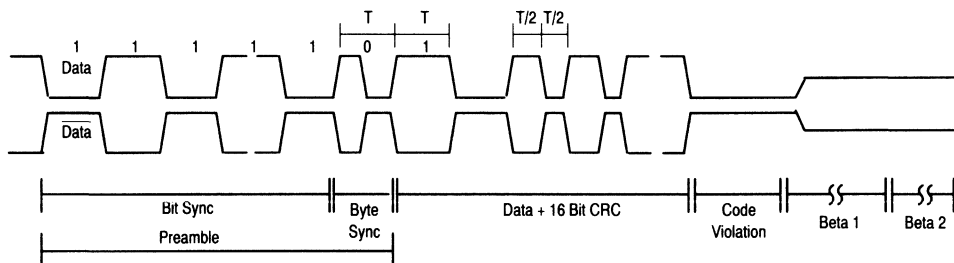
Figure 3.14. Differential Mode Data Format

Table 3.7. 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}

Table 3.8. Communications Port Programmable Glitch Filter Values

Filter (F)	Min	Typ	Max	Unit
0	2	6	9	ns
1	90	270	580	ns
2	200	535	960	ns
3	410	1070	1920	ns

Table 3.9. Differential Transceiver Electrical Characteristics

Characteristic	Min	Max
Receiver Common Mode Voltage Range to maintain hysteresis as specified below	1.2 V	V_{DD} – 2.2 V
Receiver Common Mode Range to operate with unspecified hysteresis	0.9 V	V_{DD} – 1.75 V
Input Offset Voltage	-0.05 V_{hys} – 35 mV	0.05 V_{hys} + 35 mV
Propagation Delay (F = 0, V_{ID} = V_{hys}/2 + 200 mV)	—	230 ns
Input Resistance	5 MΩ	—
Wake-Up Time	—	10 μs
Filter Asymmetry (t_{TLH}/t_{PHL})	0.7	1.42

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. Table 3.10 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 3.10.

Table 3.10. Receiver Jitter Tolerance Windows

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

The allowable network bit rates for Direct Mode are given in Table 3.6. All the active components necessary to implement twisted pair networks using Direct Mode are present on the NEURON CHIP. For a small local network of nodes a simplified direct-connect network interface, such as that shown in Figure 3.15, may be employed. Use this circuit only when all the NEURON CHIPS on the network share a common power supply. Up to 64 nodes can be networked in direct connect mode at 1.25 Mbps. The 51 Ω resistors provide current limiting when two NEURON CHIPS might be attempting to drive the line to alternate states (a collision) and also bias the transmit signal to be within the common mode voltage range of the receiver. The diodes provide some transient protection.
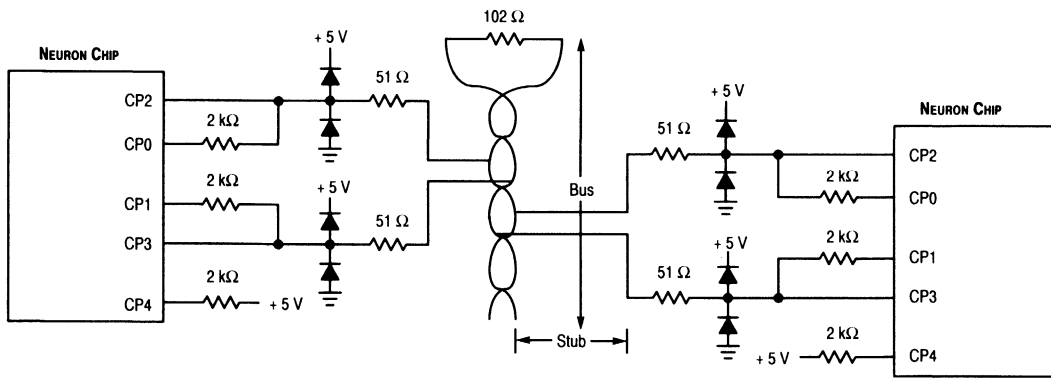
Figure 3.15. Simple Direct Connect Network Interface
(Used with direct mode differential)

3.4.1.3 DIRECT MODE DIFFERENTIAL (TRANSFORMER COUPLED). Transformer coupled twisted pair transceivers are designed to communicate between multiple remote nodes, and incorporate circuitry for common mode rejection, collision detection, line isolation, power-down protection, etc. An example is shown in Figure 3.16.

When using the transformers described in Table 3.11, collision detection can be supported over specified wire types (see twisted pair application note) and high common mode isolation and noise immunity is achieved. Other data rates and wire types would require different transformers. Designers may develop their own transformer coupled circuits. Properly designed transformer-coupled circuits can support up to 64 nodes on a single twisted pair of wires.

Table 3.11. Suggested Twisted-Pair Transformer Manufacturers for 78 kbps and 1.25 Mbps

	78 kbps	1.25 Mbps
Part Number	0505-0542	PE-65948
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	708-543-6448	619-268-2400
Contact	Bill Gray	John G. Schuler
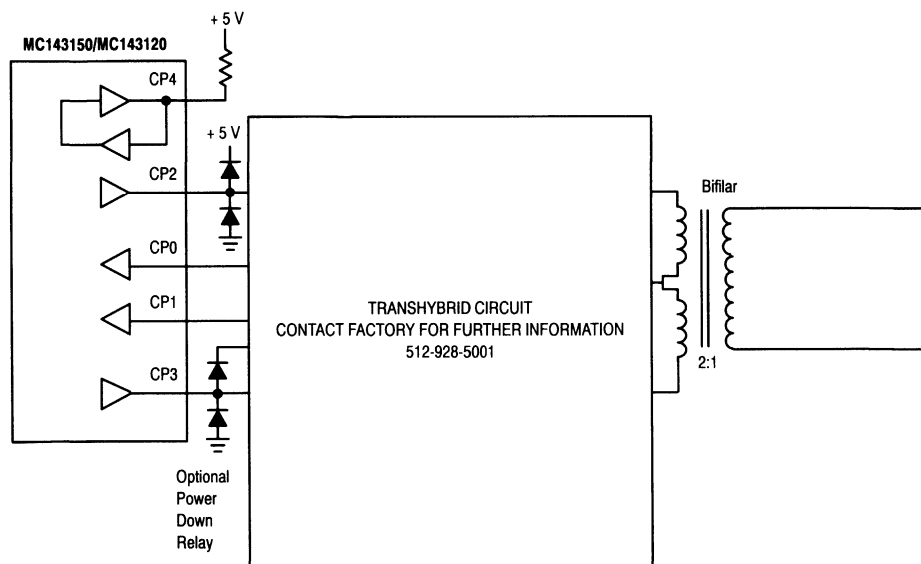
a. Used with direct mode differential @ 78 kbps
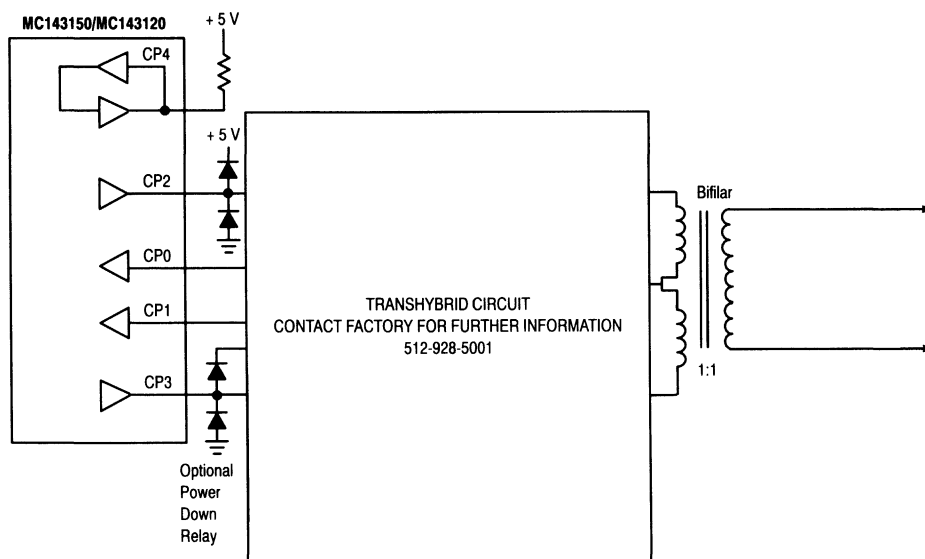
b. Used with direct mode differential @ 1.25 Mbps

Figure 3.16. Transformer Coupled Twisted Pair Interface

3.4.2 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. 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

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 3.17) 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.
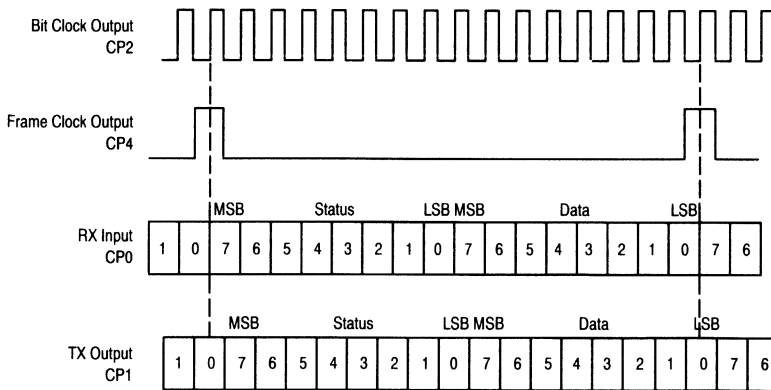
Figure 3.17. Special-Purpose Mode Data Format

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 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: Information 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 status flags are listed in Table 3.12. If it is determined that the special-purpose mode is necessary and practical for the user's transceiver design then the LONBUILDER Developer's Workbench can be used to test this mode of operation with the user's own transceiver. See Table 3.13 for Part Timing Characteristics and Figure 3.18 Timing Diagram.

Table 3.12 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 through 7 are defined by the transceiver implementation.

While the special purpose mode offers custom features, it is expected that most transceivers will use the single-ended mode for most types of medium, from coaxial cable to RF to fiber optic, since it offers Differential Manchester encoding, which takes care of clock recovery. A more detailed document entitled: *Special-purpose mode, Transceiver interface specification* will be available.

Table 3.13. Special-Purpose Mode Communication Port Timing Characteristics

Num	Parameter*	Min**	Max
1	Clock High Time	TBD	TBD
2	Clock Low Time	TBD	TBD
3	Frame Clock Low to Bit Clock Low	TBD	TBD
4	Frame Clock High	TBD	TBD
5	Frame Clock Low to First Status Bit Out	TBD	TBD

* Accuracy of values is dependent on crystal accuracy (recommended ±1.5%).
** Characteristics are @ 1.25 Mbps.
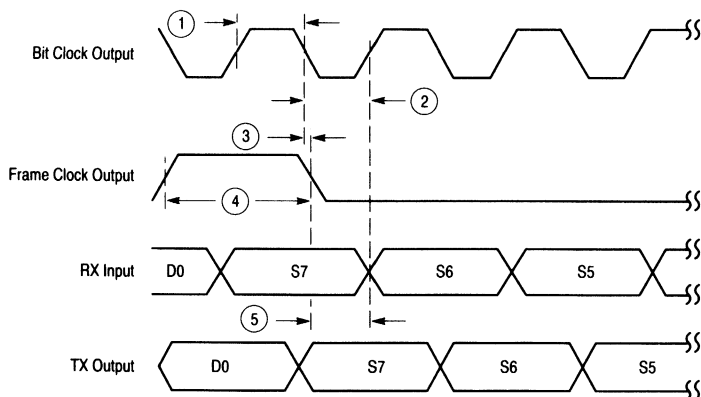
Figure 3.18. Special Purpose Mode Timing Diagram

3.5 CLOCKING SYSTEM

3.5.1 Clock Generation

The NEURON CHIP includes an oscillator that may be used to generate an input clock using an external crystal . The transconductance of this oscillator is 2.1 millisiemens minimum. The NEURON CHIP may operate over a range of input clock rates from 10 MHz to 625 kHz for low power applications. The valid input clock frequencies are: 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. The accuracy of the clock frequency must be ±1.5 % or better to ensure that nodes may correctly synchronize their bit clocks when using Direct Mode transceivers. Crystal values are listed down to 625 kHz. See Figure 3.19 and Table 3.14. Because the input clock is divided by two, a wide range input duty cycle is allowed.
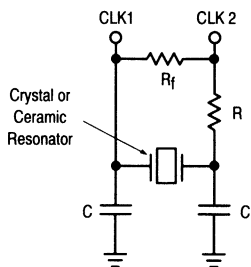
Figure 3.19. NEURON CHIP Clock Generator Circuit

Table 3.14. Clock Generator Component Values

Input Clock Frequency	Crystal		Ceramic Resonator	
	R	C	R	C
10.0 MHz	270 Ω	30 pF	270 Ω	30 pF
5.0 MHz	470 Ω	30 pF	270 Ω	30 pF
2.5 MHz	1 kΩ	36 pF	1 kΩ	36 pF
1.25 MHz	1.2 kΩ	47 pF	1.2 kΩ	47 pF
625 kHz	2.7 kΩ	47 pF	1.2 kΩ	100 pF

$R_f = 100$ kΩ for all conditions. Ceramic or crystal resonator frequency = input clock frequency. Crystal may be parallel or series resonant. NPO-type ceramic capacitors are recommended. Resistor and capacitor tolerance is ±5%. A suggested supplier for the ceramic resonators is :

Murata Erie	10 MHz	CSA 10.0 MT
2200 Lake Park Drive	5 MHz	CSA 5.0 MT
Smyrna, Georgia 30080	2.5 MHz	CSA 2.5 MG
Tel: 404-436-1300		

3.5.2 Clock-Divide Circuitry

The NEURON CHIP divides the input clock by a factor of two to provide a symmetrical on-chip system clock. As mentioned in the previous section, the input clock may be generated either by an external free-running TTL source, or by an on-chip oscillator using an external crystal or ceramic resonator.

A processor instruction cycle is three system clock cycles, or six input clock 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 μs and 4.2 μs. The formula for instruction time is:

$$\text{<Instruction Time> (μs)} = \text{<\# Cycles> x 6 / <Input Clock> (MHz)}$$

Tables 3.15, 3.16, and 3.17 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 3.15. 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

MC143150•MC143120

Table 3.16. Program Control Instruction Timings

Mnemonic	Cycles	Addressing Mode
PUSH TOS	3	TOS register
PUSH/POP cpureg	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 displ.
PUSH/POP absolute	7	Push memory data to data stack

Table 3.17. ALU Instruction Timings

Mnemonic	Cycles	Operation
INC/DEC/NOT	2	Increment/decrement/negate
TOSROLC/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
IN/OUT	7+4* #bytes	Block input/output

*These instructions optionally return from the current subroutine — add "_R" to the mnemonic and 3 cycles to the timing.

3.6 ADDITIONAL FUNCTIONS

3.6.1 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 through IO7, selected by the timer/counter multiplexer
- 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 service pin and I/O pins IO4–IO7 be disabled to further reduce power consumption.

3.6.2 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) and scales inversely with the input clock rate. When the NEURON CHIP is in sleep mode, all the watchdog timers are disabled.

3.6.3 Reset Circuitry

3.6.4 Power On Reset

The RESET pin is an open-drain active low input to the NEURON CHIP, with an internal pull-up and also an output under software control. When power is applied to the following circuit, the reset capacitor C_r charges via the internal pull-up and the diode, providing a clean reset to the NEURON CHIP and other attached circuitry. When V_{DD} is turned off, the circuit resets the NEURON CHIP by discharging the capacitor through the external diode and the source impedance of the power supply in conjunction with the internal P channel transistor acting as a diode to ground when power down occurs.

The value of the capacitance is given by:

$$C_r = (300\ \mu A + I) \times T/V$$

where:
- C_r = minimum value of reset capacitance
- 300 μA = maximum built-in pull-up current
- I = maximum pull-up current due to external devices (e.g. TTL input current)
- T = time for V_{DD} to reach 90% of final value
- V = final V_{DD} voltage

e.g., if a 5.25 V power supply ramps up in one full 60 Hz cycle (16.67 ms), and external components could contribute up to 15 μA of pull-up current, a total minimum value of 1 μF capacitance is required. The RESET pin may also be activated as an output under software control. It may therefore be used to reset external circuitry such as transceivers. In this case, the optional external reset capacitor C_e is needed, as shown in Figure 3.20. Figure 3.21 shows a typical analog waveform.

$$C_e = (300\ \mu A + I) \times T/(V - 0.5\ V)$$

where:
- C_e = minimum value of external reset capacitance
- 300 μA = maximum built-in pull-up current
- I = maximum pull-up current due to external circuits
- T = reset pin low hold time for external devices to reset properly
- V = minimum reset V_{IL} level of external device
- 0.5 V = maximum expected offset of RESET pin

For example, if two external LSTTL devices (with $V_{IL} = 0.8$ V) contribute 400 μA each of pull-up current, and require a 30 ns reset low hold time, a total minimum value of 110 pF capacitance is required. The total capacitance directly connected to the RESET pin, including stray and external device input capacitance, may not exceed 250 pF. A NEURON CHIP needs no capacitance on the RESET pin to reset itself or be reset by an external device. These components will provide proper reset for a normal power down and power up supply condition. For systems in which the power supply voltage may fluctuate or does not completely power down in all cases, a low voltage detect circuit, as shown in Figure 3.22, is recommended.
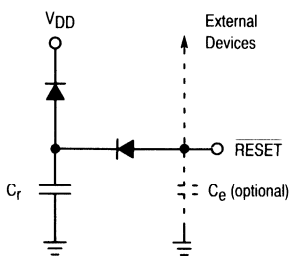
Figure 3.20. Power on Reset Circuit
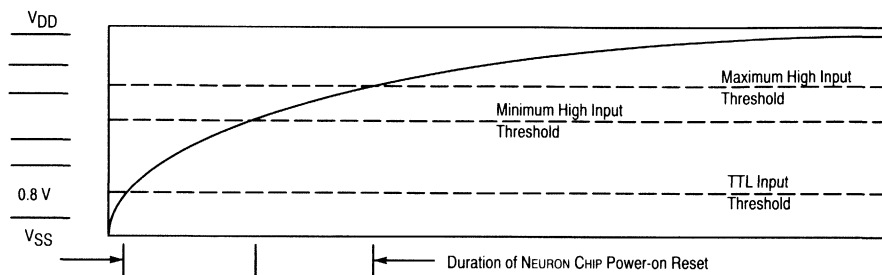
Figure 3.21. Typical Analog Waveform on Reset Pin at Power On
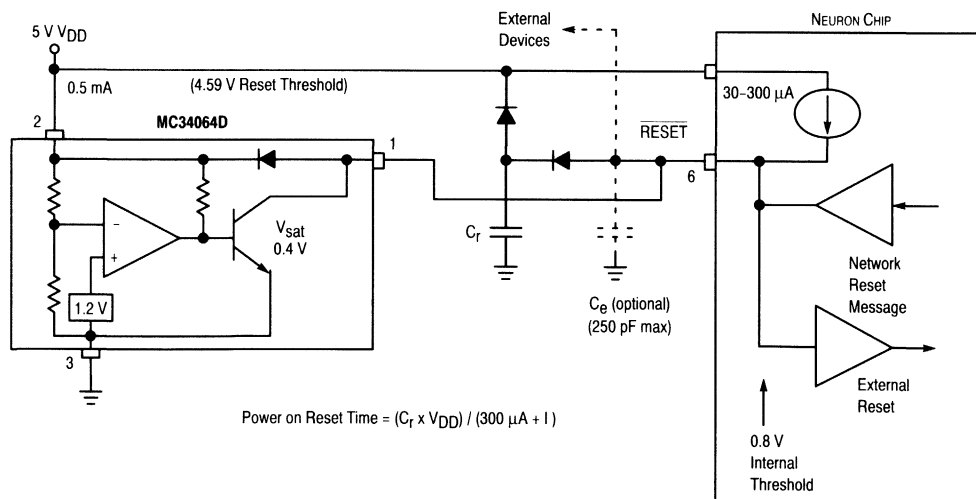
Figure 3.22. Example Low Voltage Detect/Reset Circuit

To reset the NEURON CHIP from an outside source, the $\overline{\text{RESET}}$ pin must be pulled below 0.8 V by an open drain output for a duration of 20 ns. After release of the pin by the external source the pin must be allowed to rise using the internal pull-up source and any external pull-up current source of not greater than 1.6 mA.

Using the minimum internal capacitance of 10 pF and the maximum pull-up current of 1.9 mA (1.6 mA external +300 µA internal) the minimum risetime on the pin can be obtained. During reset the communications port pins and I/O pins are three-state. The address bus is an output (high or low) and all others are indeterminate (high, low, or three-state).

When a processor watchdog timer expires, or a software command to reset occurs, output N-channel transistor turns on and pulls down the RESET pin (the power on capacitor is not discharged). This transistor has the same drive capability as the other high sink pins (20 mA @ 0.8 V). When the TTL compatible input buffer senses the RESET pin is low, it sends a signal through the control logic to shut off the N-channel transistor. The RESET pin and optional capacitor (<250 pF) are allowed to begin their slow rise time, and provide the required duration of reset.

3.7 SERVICE PIN

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. Under control of NEURON CHIP firmware, this pin is used during configuration and installation of the node containing the NEURON CHIP. The firmware lights the LED 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 48-bit unique 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 3.23. During reset the Service pin state is indeterminate.
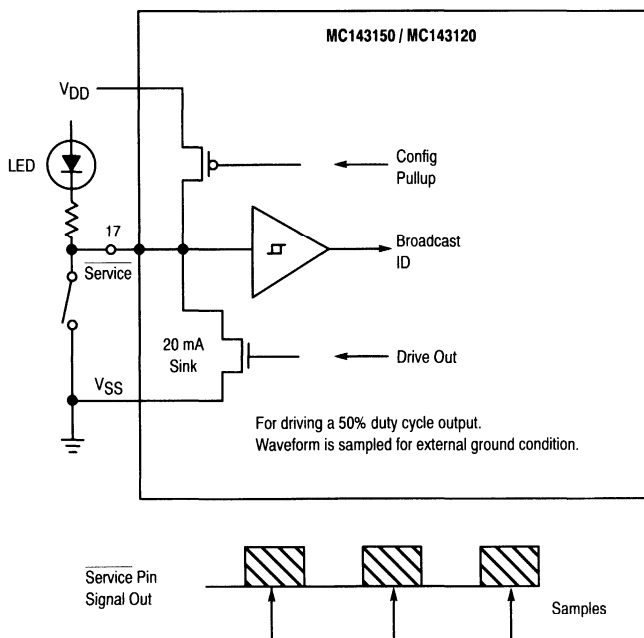
Figure 3.23. Service Pin Circuit

SECTION 4
PROCESSOR ORGANIZATION

4.1 PROCESSING UNITS

The three processors are dedicated to the following functions by the system software (Figure 4.1).

Processor 1 is the Media Access Control (MAC) layer processor that handles layer one of the seven-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 packet buffers located in shared memory.

Processor 2 is the Network Processor that implements layers two through six 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 packet buffers in shared memory to communicate with Processor 1, and Service Interface Control Blocks (SICBs) to communicate with Processor 3.

Processor 3 is the Application Processor. It runs 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 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 run-time library of function calls to perform event checking and input/output, to send and receive messages across the network, and to control miscellaneous functions of the NEURON.

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.
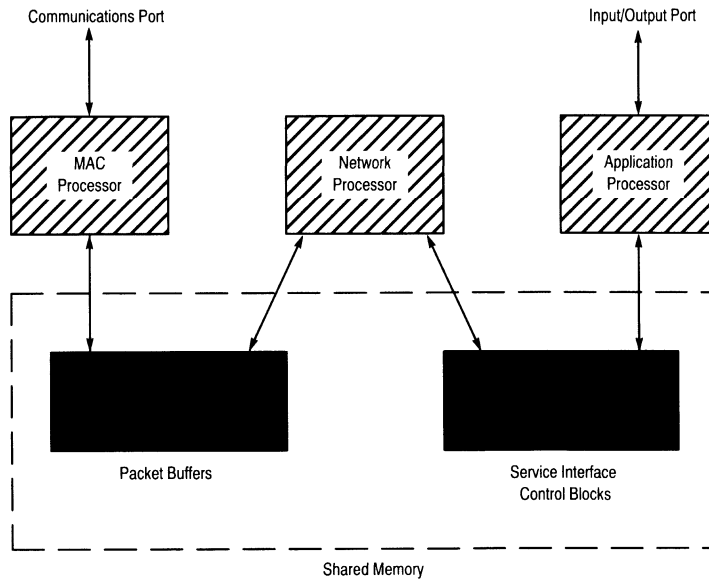
Figure 4.1. Processor Organization Memory Allocation

4.2 MEMORY ALLOCATION

4.2.1 MC143150 Memory Allocation (See Figure 4.2)

- 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
- 2,048 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; 6,144 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

4.2.2 MC143120 Memory Allocation (See Figure 4.3)

- 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
- 1,024 bytes of static RAM 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

0000	16K NEURON CHIP Firmware and Reserved Space
3FFF	
4000	42K of Memory Space Available to the User
E7FF	
E800	2K RAM
EFFF / F000	0.5K EEPROM
F1FF	2.5K Reserved
FC00	1K Memory-mapped I/O and Reserved Space
FFFF	
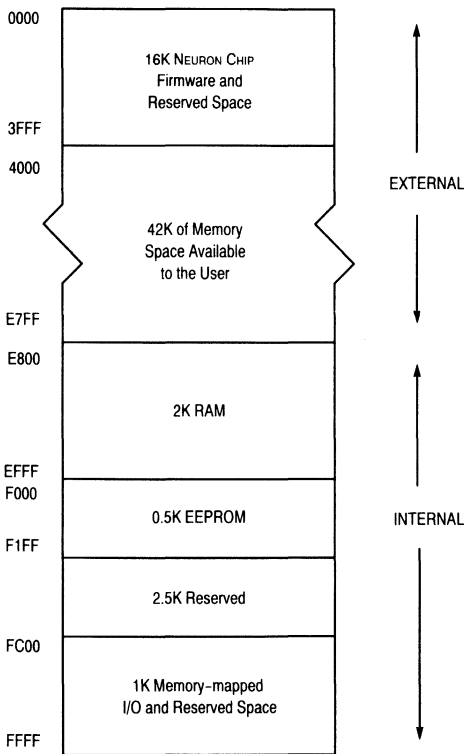
EXTERNAL

INTERNAL

Figure 4.2. MC143150 Processor Memory Map

0000	10K NEURON CHIP Firmware (ROM)
27FF	
	Unavailable
EC00	1K RAM
EFFF / F000	0.5K EEPROM
F1FF	2.5K Reserved
FC00	1K Memory-mapped I/O and Reserved Space
FFFF	
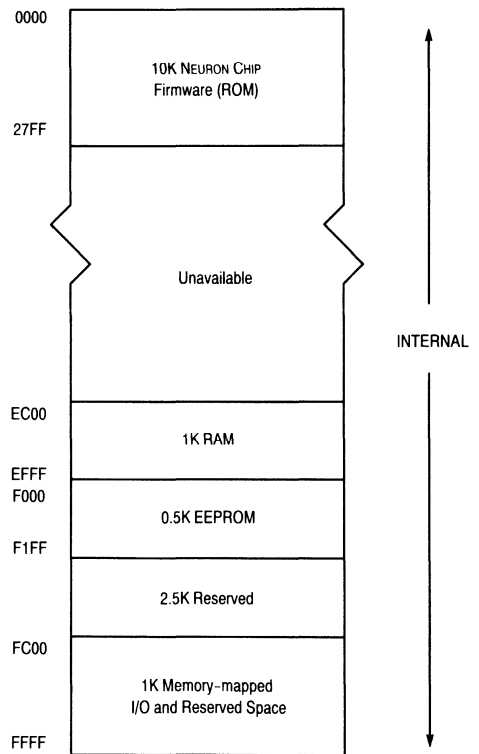
INTERNAL

Figure 4.3. MC143120 Processor Memory Map

MC143150•MC143120

SECTION 5
INPUT/OUTPUT INTERFACES

The NEURON CHIP connects to application-specific external hardware via eleven pins named IO0 through 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 (the 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 24 different I/O functions available — 11 for input, 11 for output, and two bidirectional functions.

5.1 HARDWARE CONSIDERATIONS

Various combinations of I/O pins may be configured as digital inputs or outputs as listed in Tables 5.1, 5.2, and 5.3. For the electrical characteristics of these pins, see Tables 6.1, 6.2, and 6.3 in Section 6. 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, IO5, IO6, and IO7 have optional pull-up current sources (30 μA to 300 μA). These are enabled with a NEURON C compiler directive (# pragma enable_io_pullups). Pins IO0, IO1, IO2, and IO3 have high sink capability (20 mA @ 0.8 V). The others have the standard sink capability (1.4 mA @ 0.4 V). The latency and timing values described in this section are typical. The accuracy is ±10%. Most latency values scale up at lower input clock rates.

Table 5.1. Summary of Input Functions

I/O Mode	Applicable I/O Pins	Input Value
Bit	IO0–IO10	0, 1 binary data
Bit Shift	Any adjacent pair (with even bit being the clock)	Up to 16 bits of clocked data
Byte	IO0–IO7	0 .. 255 binary data
Level Detect	IO0–IO7	Negative edge or no edge detected
Nibble	IO0..IO3 to IO4..IO7	0 .. 15 binary data
On-Time	IO4–IO7	Pulse width of 0.2 μs – 1.678 s
Period	IO4–IO7	Signal period of 0.2 μs – 1.678 s
Pulse Count	IO4–IO7	0 .. 65,535 input edges during 0.839 s
Quadrature	IO4 & IO5, IO6 & IO7	± 16,383 binary Gray code transitions
Serial	IO8	8-bit characters at 600, 1200, 2400, or 4800 baud
Total Count	IO4–IO7	0 .. 65,535 input edges

Table 5.2. Summary of Output Functions

I/O Mode	Applicable I/O Pins	Output Value
Bit	IO0–IO10	0, 1 binary data
Bit Shift	Any adjacent pair	Up to 16 bits of clocked data
Byte	IO0 .. IO7	0 .. 255 binary data
Frequency	IO0, IO1	Square wave of 0.3 Hz to 2.5 MHz
Nibble	IO0 .. IO3, IO4.. IO7	0 .. 15 binary data
One-Shot	IO0, IO1	Pulse of duration 0.2 µs to 1.678 s
Pulse Count	IO0, IO1	0 .. 65,535 pulses
Pulse Width	IO0, IO1	0 .. 100% duty cycle pulse train
Serial	IO10	8-bit characters at 600, 1200, 2400, or 4800 baud
Triac	IO0, IO1 & IO4–IO7	Delay of output pulse w.r.t. input edge
Triggered Count	IO0, IO1 & IO4–IO7	Output pulse controlled by counting input edges

Table 5.3. Summary of Bi-Directional Functions

I/O Mode	Applicable I/O Pins	Output Value
NEUROWIRE	IO8 (clk) & IO9, IO10 (data)	Up to 255 bits of clocked data
Parallel I/O	IO0–IO10	Up to 255 bytes of data

To maintain and provide consistent behavior for external events, all eleven 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 (5 MHz). Therefore, for any signal to be reliably synchronized it must be at least 220 ns in duration (see Figure 5.1).
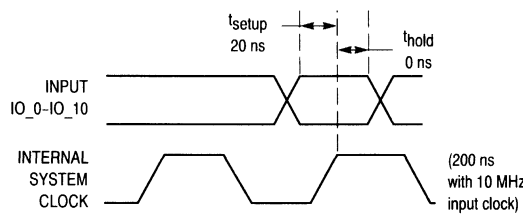
Figure 5.1. Synchronization of External Signals

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 programmers 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 delay detecting the event. If the state changes at a faster rate than software sampling can occur then the interim changes will go undetected. One exception to this is the level detect input which is latched by a flip-flop with a 200 ns clock. The level detect transition event will be latched but there will be a delay in software detection (see Level Detect description in this section). 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 programmers guide for detailed programming information.*

5.2 SOFTWARE CONSIDERATIONS

Certain events, which are associated with changes in input values, are predefined. The task scheduler can execute associated user code when these changes occur. The *when* clause, which is defined in the NEURON C programming language, is used to evaluate the state of the events. There is latency associated with evaluating events on the I/O pins due to *when*-clause to *when*-clause software processing delays. The delay varies with the number *when* statements and other factors. Figure 5.2 illustrates the best case latency associated with two *when* clauses doing bit outputs (high then low) that evaluate to be true. The NEURON C code which would perform this is:

```
/* NEURON C code to drive I/O pin high then low */
    IO_0 output bit testbit;
    when (1) {
        io_out (testbit, 1); }
    when (1) {
        io_out (testbit,0 ); }
```
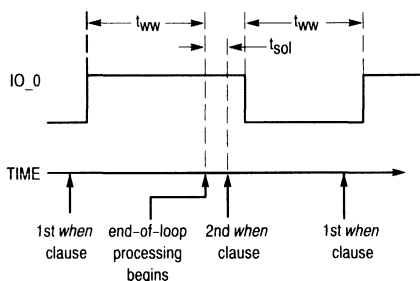

Symbol	Description	Typ @ 10 MHz
t_{WW}	*when*-clause to *when*-clause latency	940 µs
t_{sol}	Scheduler overhead latency (see text)	54 µs

Figure 5.2. *when* -Clause to *when* -Clause Latency, t_{WW} and Scheduler Overhead Latency, t_{sol}
(Not to scale)

Figure 5.2 shows the best case delays associated between two *when* clauses that evaluate to be true and illustrates the scheduler's end-of-loop latency, t_{sol}. 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}.

NOTE

Some input/output functions suspend application processing until the task is complete. This is because they are firmware driven. These are BIT SHIFT, NEUROWIRE, PARALLEL, and SERIAL I/O FUNCTIONS. They do not suspend network communication as this is handled by the NETWORK processor and the MEDIA ACCESS processor.

5.3 DIGITAL INTERFACES (BIT I/O, BYTE I/O, LEVEL DETECT, AND NIBBLE)

5.3.1 Bit I/O

Pins IO0 through 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 through IO3 allows these pins to drive many I/O devices directly. See Figures 5.3, 5.4, and 5.5.

Figure 5.3. Bit I/O

Figure 5.4. Bit Input Latency Values

Symbol	Description	Typ @ 10 MHz
t_{fin}	Function call to sample IO0 to IO10	41 µs
t_{ret}	Return from Function IO0 IO1 IO2 IO3 IO4 IO5 IO6 IO7 IO8 IO9 IO10	19 µs 23 µs 28 µs 32 µs 37 µs 41 µs 46 µs 50 µs 19 µs 23 µs 28 µs
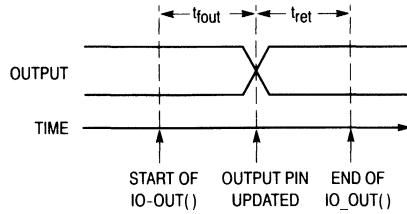
Symbol	Description	Typ @ 10 MHz
t_{fout}	Function call to update IO3 to IO5 All others	69 µs 60 µs
t_{ret}	Return from function IO0 to IO10	5 µs

Figure 5.5. Bit Output Latency Values

5.3.2 Byte I/O

Pins IO0 through 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 5.6, 5.7, and 5.8.
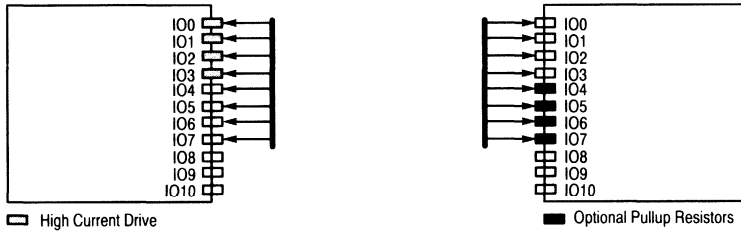
Figure 5.6. Byte I/O
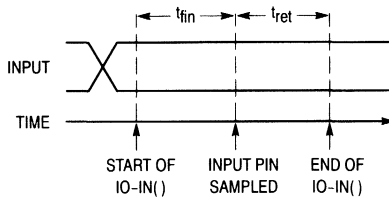
Symbol	Description	Typ @ 10 MHz
t_{fin}	Function call to input sample	24 µs
t_{ret}	Return from function	4 µs

Figure 5.7. Byte Input Latency Values
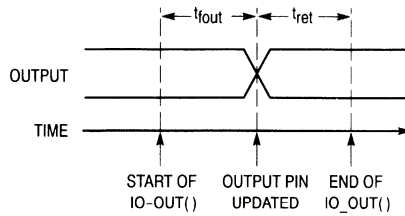
Symbol	Description	Typ @ 10 MHz
t_{fout}	Function call to update	57 μs
t_{ret}	Return from function	5 μs

Figure 5.8. Byte Output Latency Values

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

Pins IO0 through IO7 may be individually configured as level detect 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 function 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 5.9.
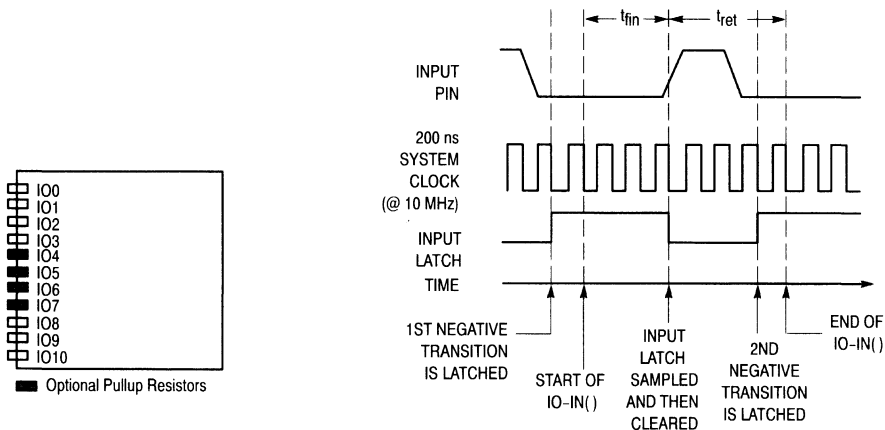
Symbol	Description	Typ @ 10 MHz
t_{fin}	Function call to sample	
	IO0	35 μs
	IO1	39 μs
	IO2	44 μs
	IO3	48 μs
	IO4	53 μs
	IO5	57 μs
	IO6	62 μs
	IO7	66 μs
t_{ret}	Return from function	32 μs

Figure 5.9. Level Detect Input Latency Values

5.3.4 Nibble I/O

Groups of four consecutive pins between IO0 and 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 5.10, 5.11, and 5.12.
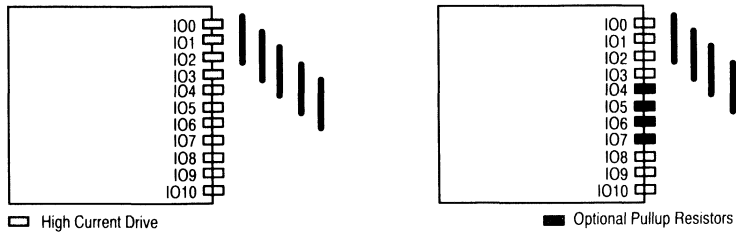
Figure 5.10. Nibble I/O
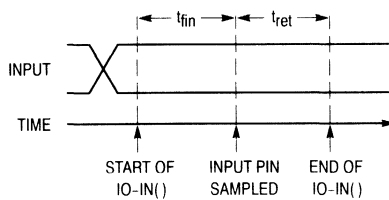
Symbol	Description	Typ @ 10 MHz
t_{fin}	Function call to sample IO0 to IO4	41 μs
t_{ret}	Return from function IO0 IO1 IO2 IO3 IO4	18 μs 23 μs 28 μs 32 μs 37 μs

Figure 5.11. Nibble Input Latency Values
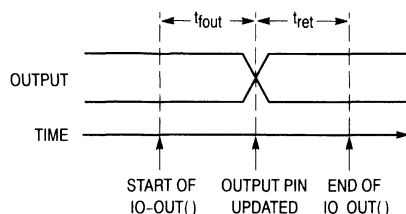
Symbol	Description	Typ @ 10 MHz
t_{fout}	Function to update IO0 IO1 IO2 IO3 IO4	 78 µs 90 µs 102 µs 113 µs 125 µs
t_{ret}	Return from function IO0 to IO4	 5 µs

Figure 5.12. Nibble Output Latency Values

5.4 PARALLEL I/O INTERFACE FUNCTION

5.4.1 Introduction

Pins IO0 through IO10 may be configured as a bidirectional 8-bit data and 3-bit control port for connection to another 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 either master/slave A, or master/slave B mode. Typically, two NEURON CHIPS interface in master/slave A mode and interface with a non-NEURON CHIP processor in the master/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 per transfer). *Consult the NEURON C programmers guide for detailed programming instructions.* Upon a reset condition the master processor monitors the low transition of the handshake line from the slave NEURON CHIP, then passes a CMD_RESYNC (0 X 5A) to the NEURON CHIP for synchronization purposes. This must be done within 0.84 seconds after reset goes high with a NEURON CHIP running at 10 MHz to avoid a watchdog reset error condition (see NEURON C programmers guide). The CMD_RESYNC is followed by the slave acknowledging with a CMD_ACKSYNC (0 X 07). 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.

5.4.2 Master/Slave A Mode

This mode is recommended when interfacing two NEURON CHIPS. In master/slave A mode, the master drives IO8 as a chip select and IO9 to specify a read or write cycle, and the slave drives IO10 as a handshake acknowledgement (see Figure 5.13). The maximum data transfer rate is one byte per four processor instruction cycles, or 2.4 µs per byte at the maximum input clock rate (10 MHz). The data transfer rate scales proportionally to the input clock rate. Typical timing is shown in Table 5.4 (note that a master write is a slave read). Timing for the case where the NEURON CHIP is the master (Figure 5.14) refers to measured output timing at 10 MHz. After every byte write or byte read the handshake line is monitored by the master

to verify the slave has completed processing (when HS = 0) and the slave is ready for next byte transfer. This is done automatically in NEURON CHIP to NEURON CHIP (master/slave A mode) data transfers.
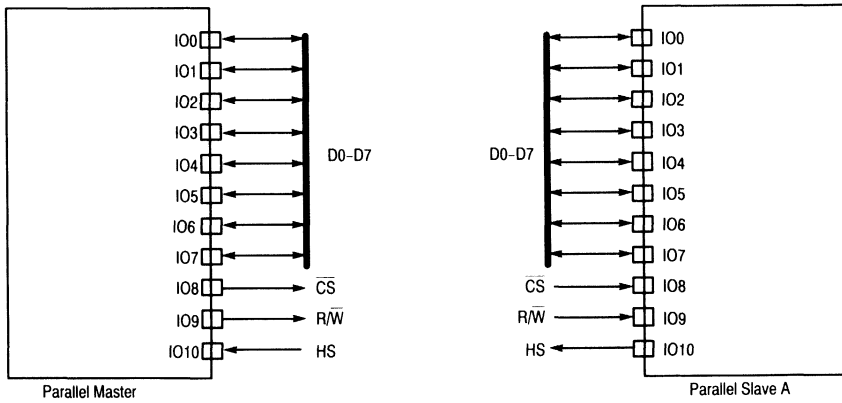
Figure 5.13. Parallel I/O—Master and Slave A
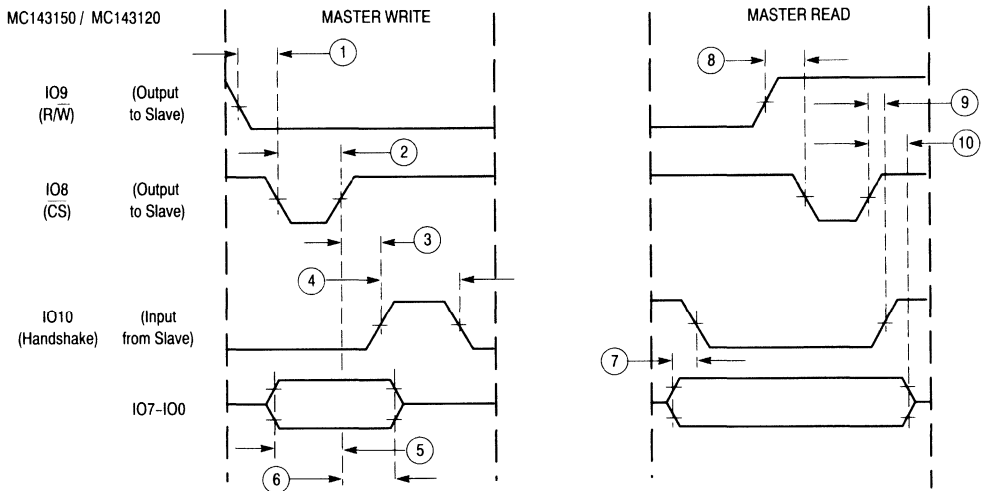
Figure 5.14. Parallel Interface with NEURON CHIP as Master and NEURON CHIP or Non-NEURON CHIP as Slave A Mode Timing Diagram
[Master Write (Slave Read) / Master Read (Slave Write)]

Table 5.4. Typical Parallel Interface NEURON as Master Mode Timing

Num	Description	Value*	Unit
1	Write low setup before CS low	≥25	ns
2	Period, chip select low	≥50	ns
3	Delay, CS rising to HS high	25	ns typ
4	Handshake output high	>2.4	µs
5	Data setup to CS rising	≥25	ns
6	Data hold from CS rising	≥5	ns
7	Delay, new data to HS low	200	ns typ
8	Read high setup before CS low	≥25	ns
9	CS high to handshake high	≥25	ns
10	Data hold time CS high	≥200	ns

*Values are preliminary and subject to change.

Following is an example program for transferring data in parallel I/O master/slave A mode. The code is for two NEURON emulators hardwired as shown in Figure 5.13. An I/O extender card or cable can be built to access 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 to verify the transfer was complete. The master transmits [5,1,1,1,1,1] to the slave and the slave transmits [7,1,2,3,4,5,6,7,0,0,0,0,0,0] 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 zeros. 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 ones 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 [7,1,2,3,4,5,6,7,0,0,0,0,0,0].
/*

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 arrray length

struct parallel_out          //output structure
{
unsigned int len;            //actual length of data to be output
unsigned int buffer[max_out]; //array set up 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 ones
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, &pt_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, &pt_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]; 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 arrray length

struct  parallel_out            //output structure
{
unsigned int len;               //actual length of data to be output
unsigned int buffer[max_out];   //array set up 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 ones
p_out.buffer[i]=i+1;
}

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

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

Debugging the Above Programs: If a watchdog time out occurs, simultaneously reset the two emulators using the reset pushbutton switches on the face of the emulators. Also, both JP1 and JP2 on the emulator boards should be disconnected for this application.

5.4.3 Master/Slave B Mode

The master/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 selector input. When $\overline{CS}$ is asserted and either IO10 is low or IO10 is high and R/$\overline{W}$ is low , pins IO0 through IO7 form the bi-directional data bus. When IO10 is high, R/$\overline{W}$ is high, and $\overline{CS}$ is asserted, IO0 is driven as the handshake acknowledgement 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 control register. Therefore, reads by the master to an odd address access the control register for hand-shaking acknowledgements and all other reads or writes access the data register for I/O transfers. The least significant bit of the control register, which is read through pin IO0, is the handshake (HS) bit. The master reads the handshake 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 handshake 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 least significant bit of the master's address bus is typically connected to the NEURON CHIP's IO10 pin. This is illustrated in Figures 5.15, 5.16, and Table 5.5.
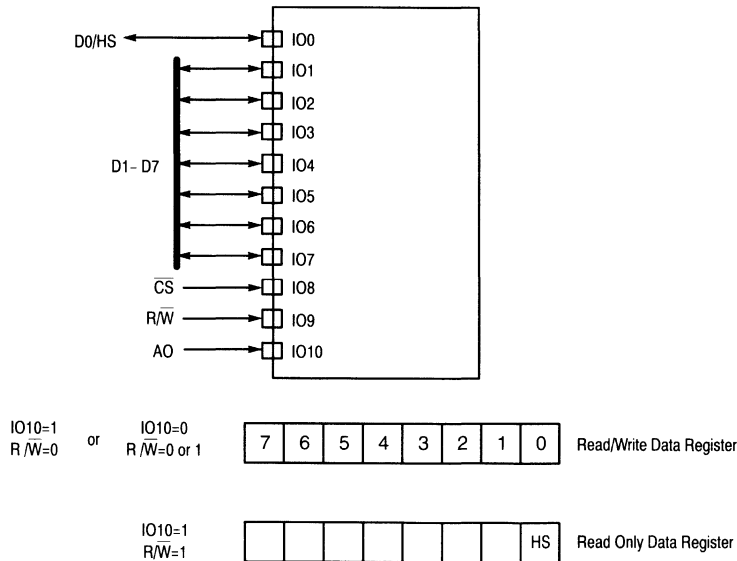
Figure 5.15. Parallel I/O Master/Slave B
(NEURON CHIP as Memory-Mapped I/O Device)
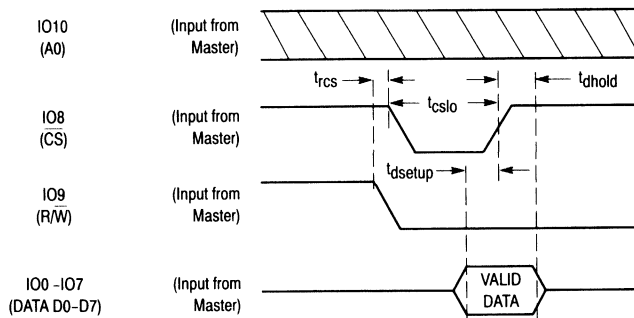
a. Slave B (NEURON CHIP) Reads From Master (Non-NEURON CHIP)
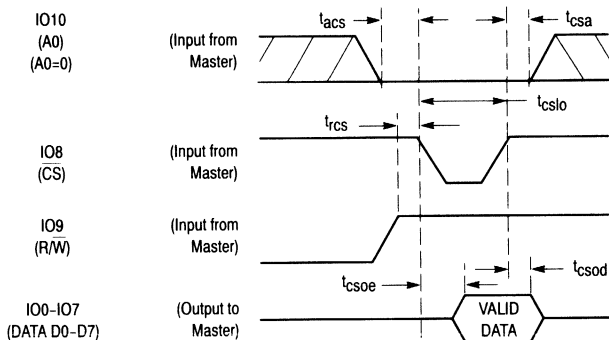
b. Slave B (NEURON CHIP) Writes Data to Master (Non-NEURON CHIP)
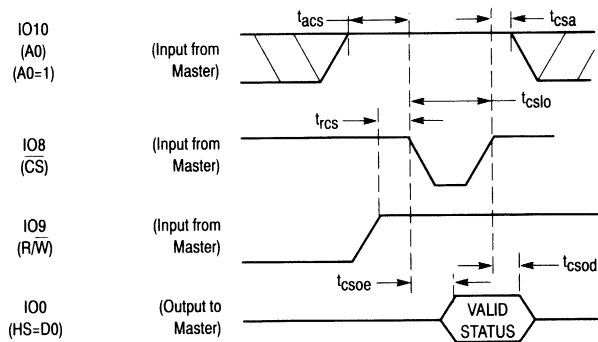
c. Slave B (NEURON CHIP) Writes Handshake (HS) Bit Through the D0 Pin to Master (Non-NEURON CHIP): Repeats until HS is low verifying previous transfer has been processed

Figure 5.16. Parallel Master/Slave B Mode Timing Diagrams

Table 5.5. Typical Parallel Mode Slave B Timing

Symbol	Description	Value
t_{rcs}	R/W setup before $\overline{CS}$ active (min)	25 ns
t_{dsetup}	Data setup before $\overline{CS}$ rising edge (min)	25 ns
t_{dhold}	Data hold after $\overline{CS}$ inactive (min)	5 ns
t_{cshs}	$\overline{CS}$ rising edge to HS transition (min)	25 ns
t_{cslo}	$\overline{CS}$ low pulse width (min)	25 ns
t_{csoe}	$\overline{CS}$ to slave output enable (data valid)	50 ns
t_{csod}	$\overline{CS}$ to slave output disabled (data invalid)	50 ns
t_{acs}	Address vaild to $\overline{CS}$ (max)	50 ns
t_{csa}	$\overline{CS}$ inactive to address invalid	25 ns

5.4.4 Token Passing

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 EOM (0 X 00) 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 = 0 X 00) 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 flow table in Section 5.4.6 for master/slave 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 handshake from the slave before writing data to the bus. If the slave owns the token, the master monitors the low transition of the handshake before reading the bus. In master/slave A mode, the NEURON CHIP handshake line is monitored on pin IO10. In master/slave B, the NEURON CHIP handshake bit is monitored on IO0 which corresponds to the least significant data bit of the control register.

5.4.6 Data Transferring

Resynchronization Procedure: The following procedure applies to master/slave A and master/slave B modes. The master initiates the resynchronization with a RESYNC (0X5A) command, and the slave acknowledges with a 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	SLAVE	
Owns Token		
Write XFER		//master has data to write (XFER=0X01)
	Read XFER	
Write (length)		//length=number of bytes of data
	Read (length)	
Write (data_0)		//master begins data transfer to slave
	Read (data_0)	
.	.	
.	.	
.	.	
Write (data_n)		//last byte of data to be transferred
	Read (data_n)	
Write EOM		//end of data transfer (EOM=00)
	Process EOM	//exchange token
	Owns 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	//end of data transfer
Process EOM		//exchange token
Owns Token		

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

MASTER	SLAVE	
Owns Token		
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   //exchange token
Owns Token

5.5 SERIAL INTERFACE

5.5.1 Bitshift I/O

Pairs of adjacent pins may be configured as serial input or output lines, the lower numbered pin being used for the clock (driven by the NEURON CHIP) and the higher numbered pin being used for up to 16 bits of serial data. The data rate may be configured as 1, 10, 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 function is useful for transferring data to external logic employing shift registers. This function suspends application processing until the operation is complete. See Figures 5.17, 5.18, and 5.19.

Bitshift Output

Bitshift Input

*Data can be either an input or an output

Figure 5.17. Bitshift I/O

Active clock edge assumed
to be positive in the diagram
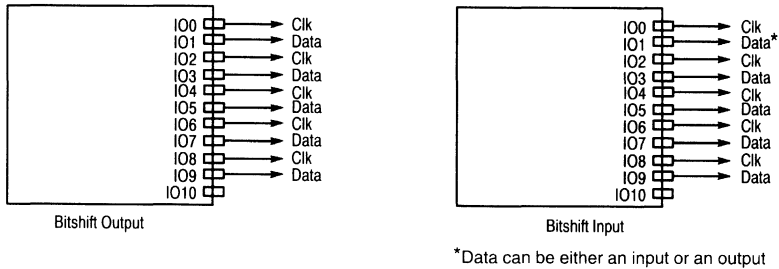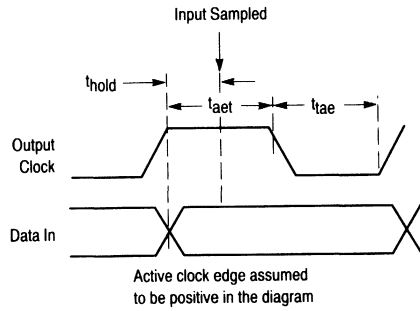
Symbol	Description	Typ @ 10 MHz
t_{hold}	Active clock edge to sampling of input data 15 kbaud data rate 10 kbaud data rate 1 kbaud data rate	 9 µs 40.8 µs 938.2 µs
t_{aet}	Active clock edge to next clock transition 15 kbaud data rate 10 kbaud data rate 1 kbaud data rate	 31.8 µs 63.6 µs 961 µs
t_{tae}	Clock transition to next active clock edge 15 kbaud data rate 10 kbaud data rate 1 kbaud data rate	 14.4 µs 14.4 µs 14.4 µs
f	Clock frequency 15 kbaud data rate 10 kbaud data rate 1 kbaud data rate	 21.6 kHz 12.8 kHz 1.03 kHz

Figure 5.18. Bitshift Input Latency Values

Active clock edge assumed
to be positive in the diagram
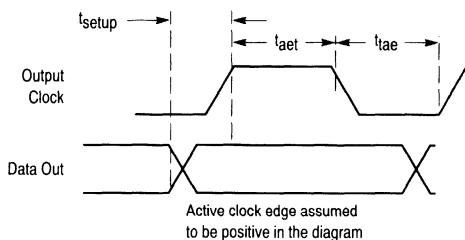
Symbol	Description	Typ @ 10 MHz
t_{setup}	Data out stable to active clock edge 15 kbaud data rate 10 kbaud data rate 1 kbaud data rate	 10.8 μs 10.8 μs 10.8 μs
t_{aet}	Active clock edge to next clock transition 15 kbaud data rate 10 kbaud data rate 1 kbaud data rate	 10.2 μs 42 μs 939.5 μs
t_{tae}	Clock transition to next active clock edge 15 kbaud data rate 10 kbaud data rate 1 kbaud data rate	 34.8 μs 34.8 μs 34.8 μs
f	Clock frequency 15 kbaud data rate 10 kbaud data rate 1 kbaud data rate	 17.9 kHz 11.4 kHz 1.02 kHz

Figure 5.19. Bitshift Output Latency Values

5.5.2 NEUROWIRE (SPI Interface) I/O Function

The NEUROWIRE function 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. NEUROWIRE function 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 5.20.
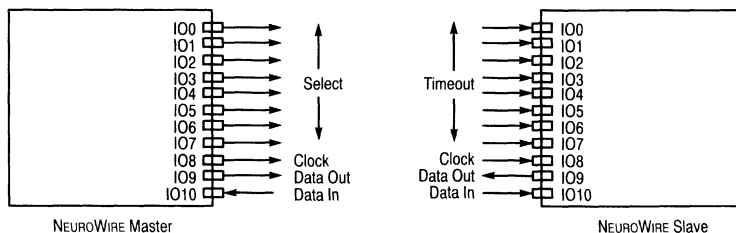
Figure 5.20. NEUROWIRE I/O

5.5.2.1 NEUROWIRE MASTER MODE. In NEUROWIRE master mode, pin IO8 is the clock (driven by the NEURON CHIP), IO9 is serial data output, and IO10 is 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. In addition, one or more of the pins IO0 through IO7 may be used as a chip select, allowing multiple NEUROWIRE devices to be connected on a 3-wire bus. The clock rate may be specified as 1, 10, or 20 kbps at an input clock rate of 10 MHz; these scale proportionally with input clock. See Figure 5.21 and Table 5.6.
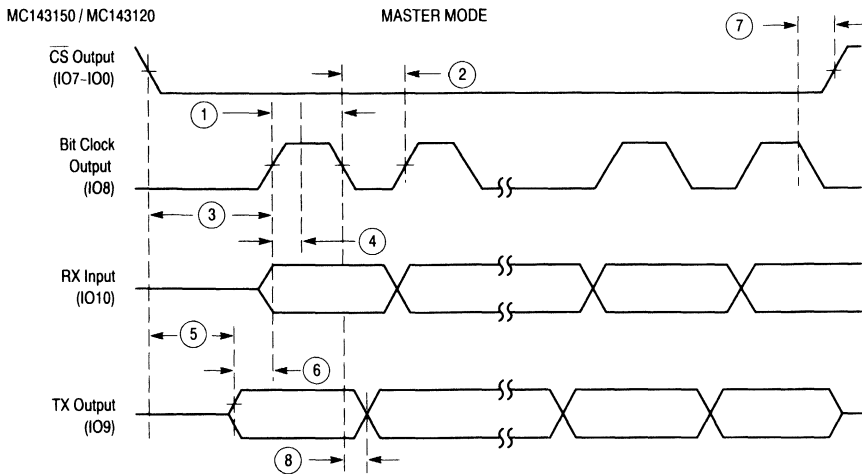
Figure 5.21. NEUROWIRE (SPI) Master Timing Diagram

Table 5.6. NEUROWIRE (SPI) Master Timing

Num	Parameter	1 kbps	10 kbps	20 kbps	Unit
1	Clock High Time	983.5	76.2	24	µs
2	Clock Low Time	22.2	22.2	22.2	µs
3	Chip select to clock high	91.2	91.2	91.2	µs
4	Clock high to RX sampling	970.9	63.6	11.4	µs
5	$\overline{CS}$ low to TX data valid	85.8	85.8	85.8	µs
6	TX data valid to clock high	5.4	5.4	5.4	µs
7	Last clock low to $\overline{CS}$ high	81.6	81.6	81.6	µs
8	TX data hold time after bit clock goes low	17	17	17	µs

5.5.2.2 NEUROWIRE SLAVE MODE. In NEUROWIRE slave mode, pin IO8 is the clock (driven by the external master), IO9 is serial data output, and IO10 is 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, which may be up to 18 kbps. One of the pins IO0 through IO7 may be designated as a time-out pin. A logic one level on the time-out 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 5.22 and Table 5.7.
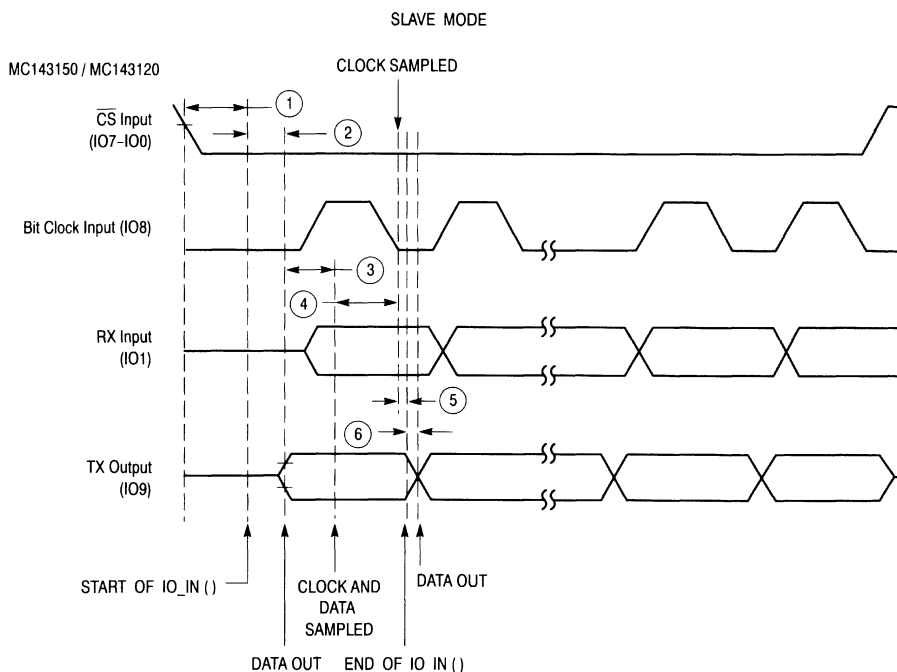
SLAVE MODE

Figure 5.22. NEUROWIRE (SPI) Slave Mode Timing Diagram

Table 5.7. NEUROWIRE Slave Mode Timing

Num	Parameter	Value
1	CS input low to start of I/O input function call (see note)	>1 ms
2	I/O_input function call to data bit out	41.2 μs
3	Data bit out to sample on RX input data bit	4.8 μs
4	Data sampled to clock low sampled	24.0 μs
5	Return from function call	19.2 μs
6	Clock low sampled to data output	25.8 μs
—	Maximum input clock rate	18.31 kHz

NOTE: Application programmer must ensure CS from master is low long enough
for slave NEURON to return to process the NEUROWIRE *when* statement

5.5.3 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 bps at maximum input clock rate (10 MHz). The data rate scales proportionally to the input clock rate. The frame format is fixed at one start bit, eight data bits, and one 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. This function suspends application processing until the operation is completed. On input, the *io_in* request will time out after 20 frame times if no start bit is received. If the stop bit has the wrong polarity (it should be a one), 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 5.23 and 5.24.
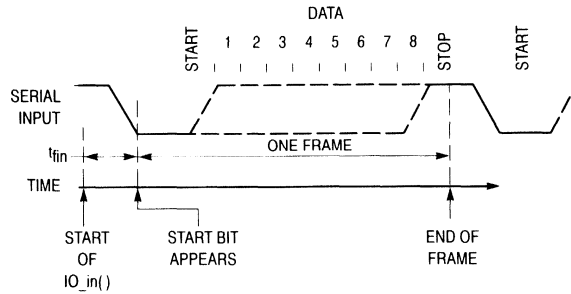
Symbol	Description	Typ @ 10 MHz
t_{fin}	Function call to input sample Min (first sample) Max (time out)	67 µs 20 byte frame
t_{ret}	Return from function	10 µs

Figure 5.23. Serial Input
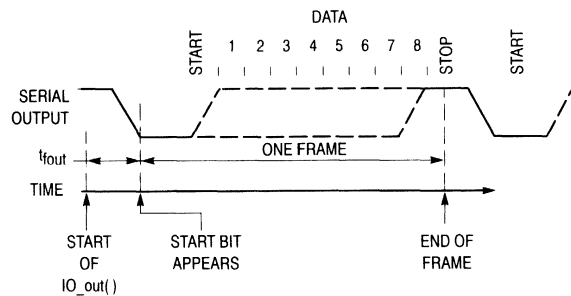
Symbol	Description	Typ @ 10 MHz
t_{fout}	Function call to start bit	79 µs
t_{ret}	Return from function	10 µs

Figure 5.24. Serial Output

5.6 TIMER/COUNTER INTERFACE FUNCTIONS

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 through IO7 as the input. The second timer/counter uses IO4 as the input, and IO1 as the output (see Figure 3.4). 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 configured as a quadrature input, then it cannot be re-assigned to another timer/counter function in the same application program.

5.6.1 Timer/Counter Input Functions (On-Time, Period, Pulsecount Input, Quadrature, Totalcount)

Input timer/counter functions have the advantage (over non-timer/counter functions) 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 cannot 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 cannot be measured with the timer/counters. Figure 5.25 shows an example of how the timer/counter functions are processed with NEURON C *when* statement.
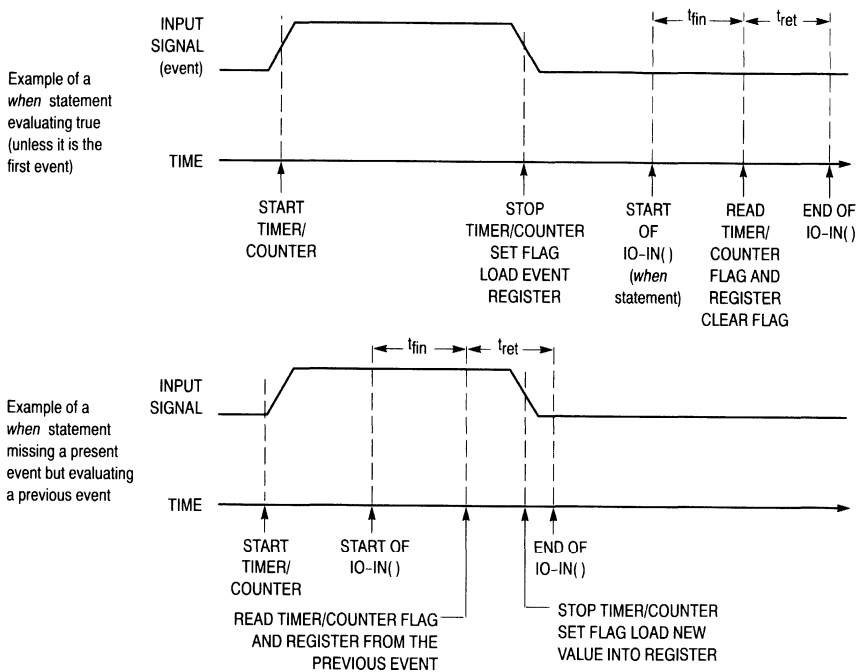
Figure 5.25. *when* **Statement Processing Using the On-Time Input Function**

5.6.1.1 ON-TIME INPUT. A timer/counter may be configured to measure the time for which its input is asserted. Table 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 function 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 5.25 and 5.26.
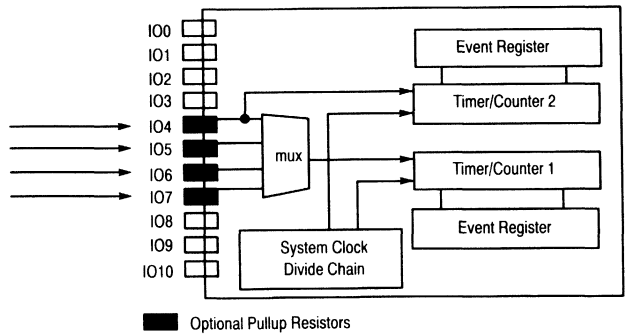
Symbol	Description	Typ @ 10 MHz
t_{fin}	Function call to input sample	86 μs
t_{ret}	Return from function	52 μs

Figure 5.26. On-Time Latency Values (see Figure 5.25)

5.6.1.2 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 5.8 shows the resolution and maximum time measured for various clock selections. This function is useful for instantaneous frequency or tachometer applications. Analog-to-digital conversion can be implemented using a voltage-to-frequency converter with this function. See Figure 5.27.
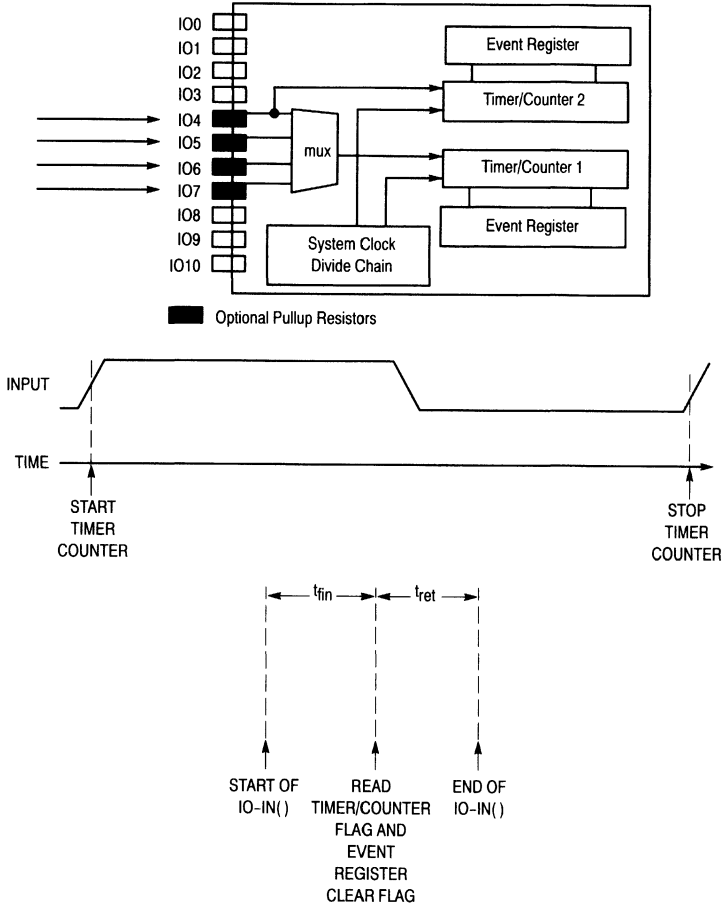
Reference Figure 5.25

Symbol	Description	Typ @ 10 MHz
t_{fin}	Function call to input sample	86 μs
t_{ret}	Return from function	52 μs

Figure 5.27. Period Input Latency Values

5.6.1.3 PULSE COUNT 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 function is useful for average frequency measurements, or tachometer applications. See Figure 5.28.
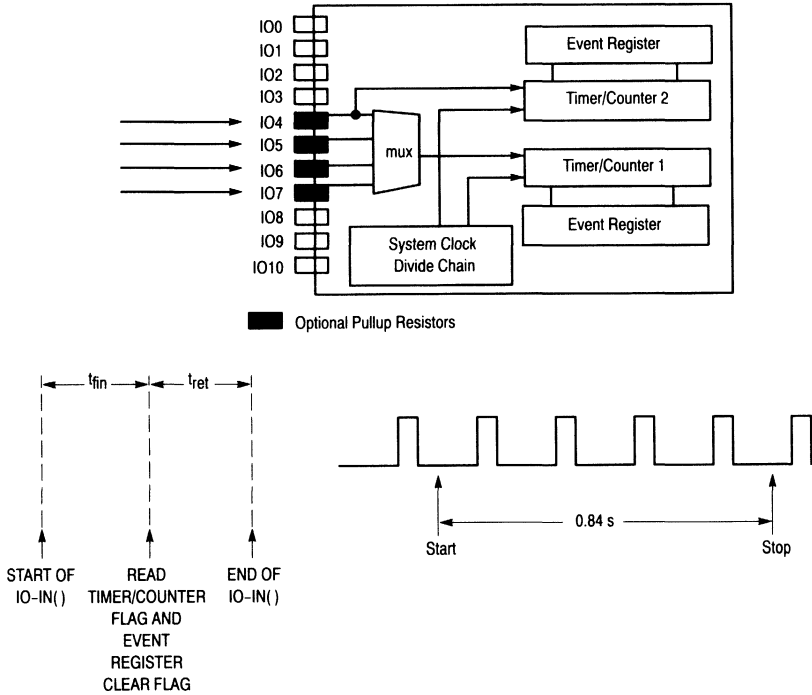
Reference Figure 5.25

Symbol	Description	Typ @ 10 MHz
t_{fin}	Function call to input sample	86 μs
t_{ret}	Return from function	52 μs

Figure 5.28. Pulse Count Input Latency Values

5.6.1.4 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 half 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 1.25 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 5.29.
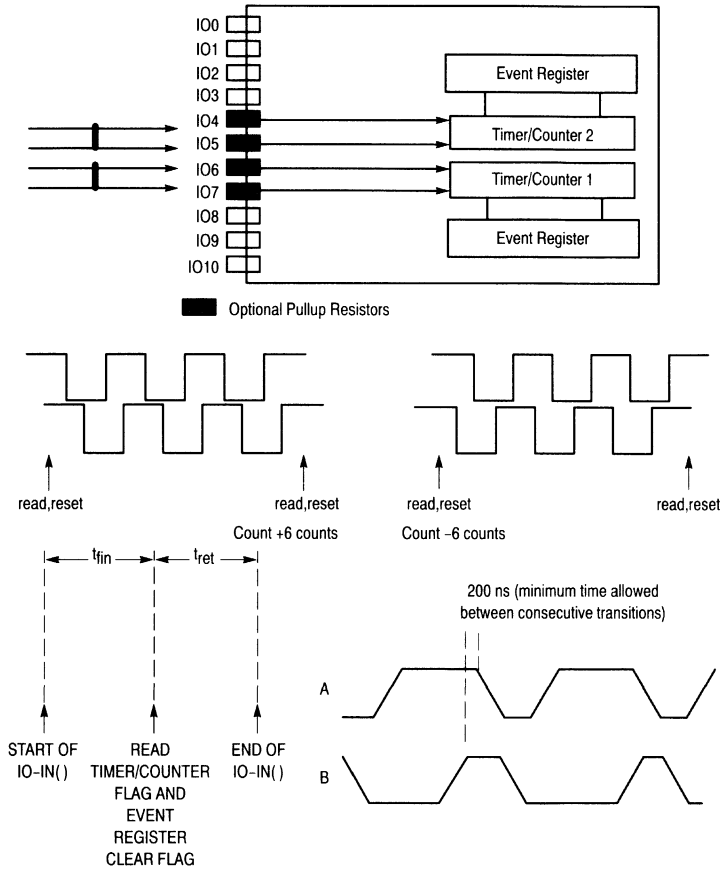
Reference Figure 5.25

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

Figure 5.29. Quadrature Input Latency Values

5.6.1.5 TOTAL COUNT INPUT. A timer/counter may be configured to count input edges, either rising or falling but not both. Reading the value of a total count 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 function is useful for counting external events such as contact closures, where it is important to keep an accurate running total. See Figure 5.30.

Optional Pullup Resistors
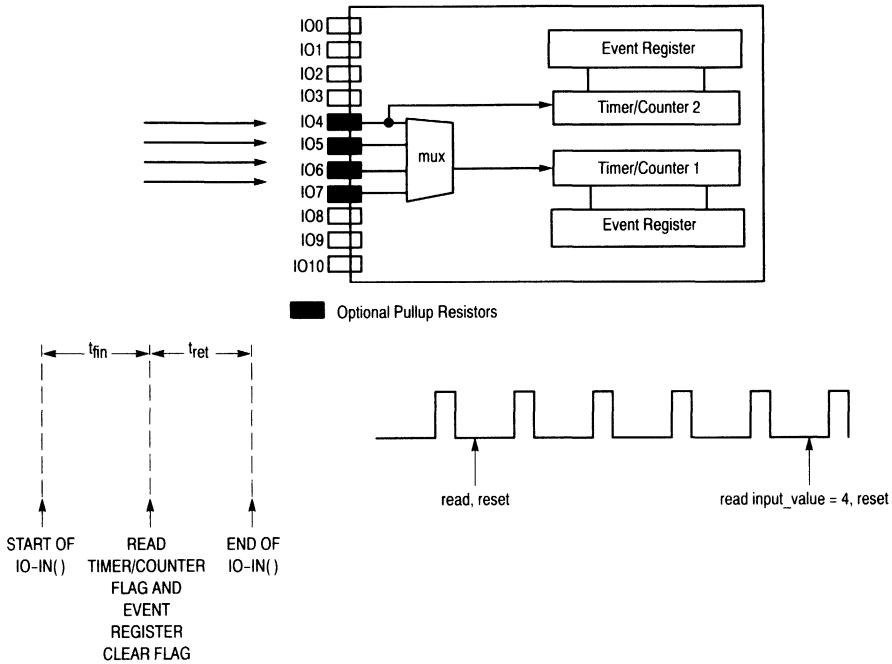
Reference Figure 5.25

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

Figure 5.30. Total Count Input Latency Values

5.6.2 Timer/Counter Output Functions (Frequency, One-Shot, Pulsecount Output, Pulsewidth, Triac, Triggered Count)

5.6.2.1 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 function 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 5.31.

High Current Sink Drivers

Frequency Resolution and Maximum Range

CLK	Resolution	Range
0	0.4	26.2
1	0.8	52.4
2	1.6	104.9
3	3.2	209.7
4	6.4	419.4
5	12.8	838.9
6	25.6	1678
7	51.2	3355

Symbol	Description	Typ @ 10 MHz
t_{fout}	Function call to output update	96 μs
t_{ret}	Return from function	13 μs
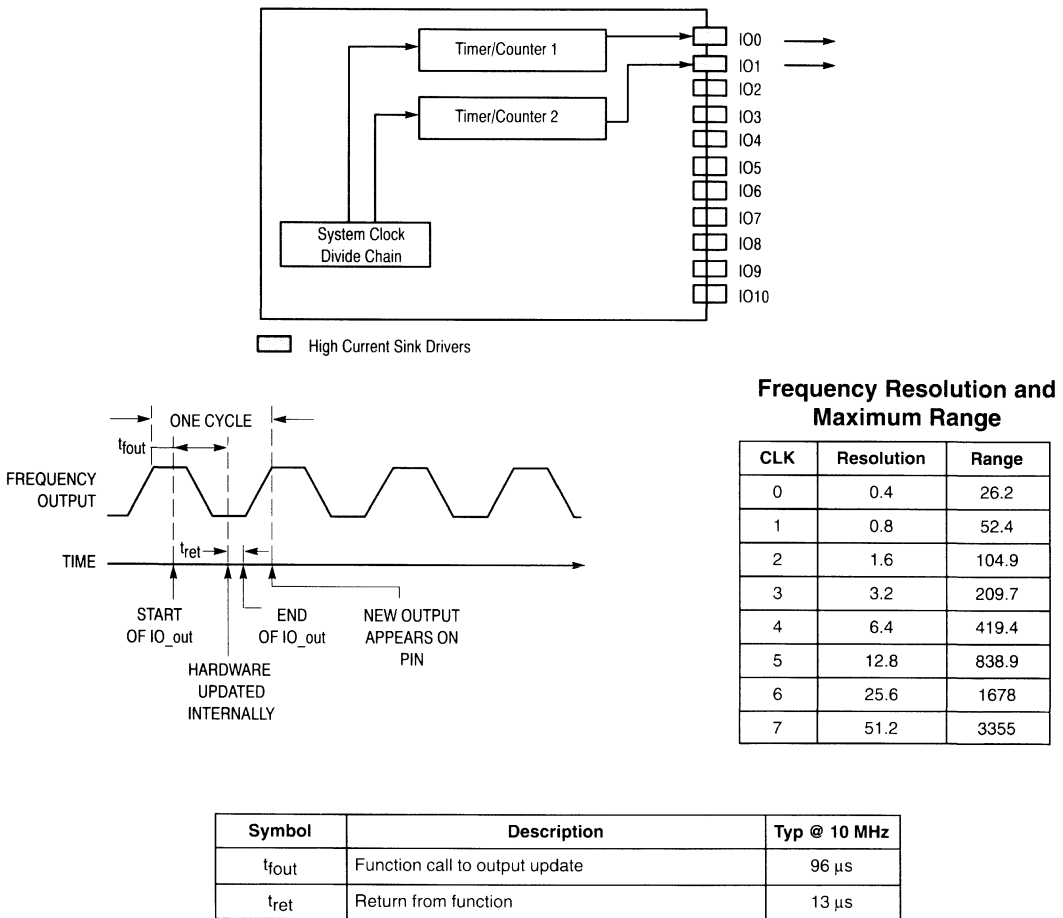
Figure 5.31. Frequency Output Latency Values

5.6.2.2 ONE-SHOT OUTPUT. A timer/counter may be configured to generate a single pulse of program-mable duration. The asserted state may be either logic high or logic low. Retriggering the one-shot before the end of the pulse causes it to continue for the new duration. Table 5.8 gives the resolution and maximum time of the pulse for various clock selections. This function is useful for generating a time delay without intervention of the application processor. See Figure 5.32.
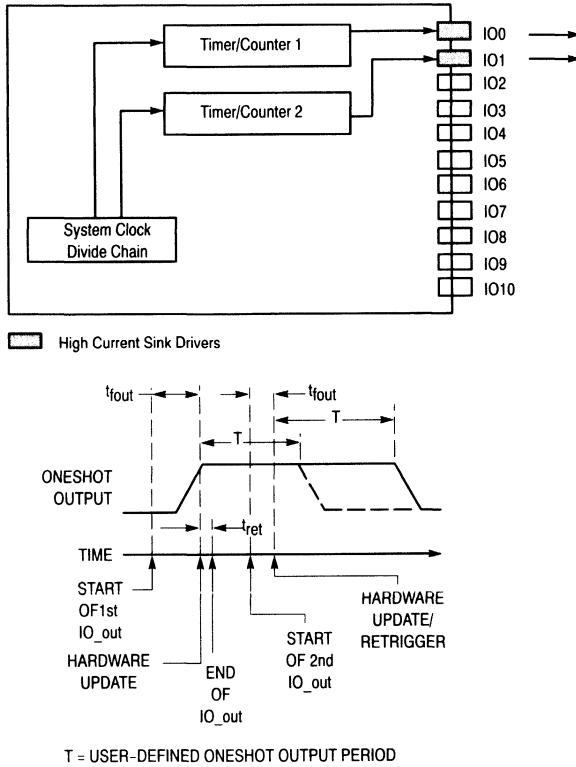
T = USER-DEFINED ONESHOT OUTPUT PERIOD

Symbol	Description	Typ @ 10 MHz
t_{fout}	Function call to output update	96 μs
t_{ret}	Return from function	13 μs

Figure 5.32. One-Shot Output Latency Values

5.6.2.3 PULSE COUNT 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 5.9, with clock select values of 1 through 7. This mode is useful for external counting devices that can accumulate pulse trains, such as stepper motors. See Figure 5.33.

Symbol	Description	Typ @ 10 MHz
t_{fout}	Function call to first active output pulse edge	115 μs
t_{ret}	Return from function	5 μs
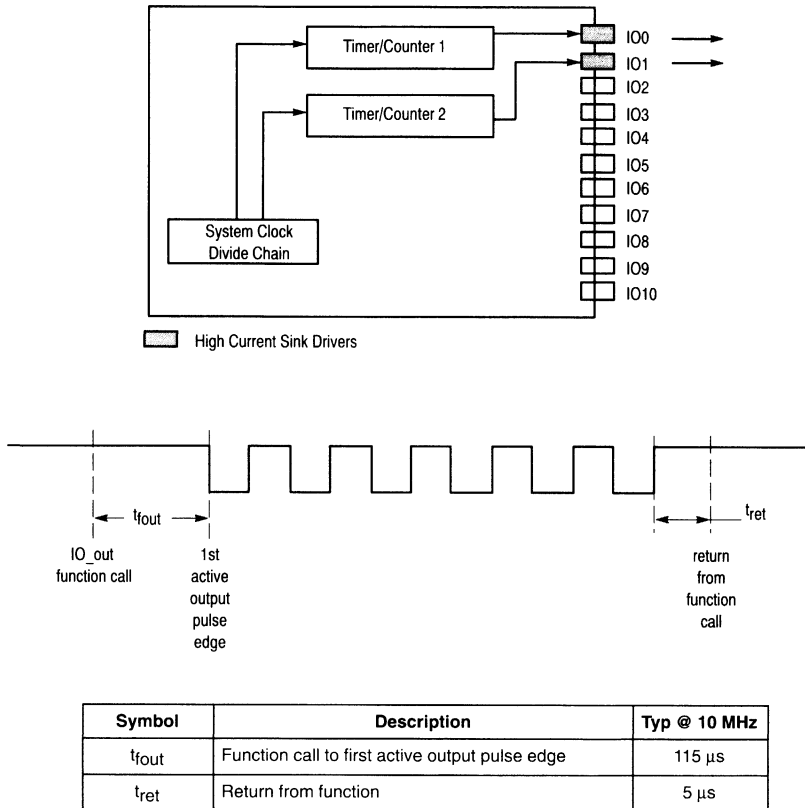
Figure 5.33. Pulse Count Output Latency Values

5.6.2.4 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 the eight values given by Table 5.9. 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 the eight values given by Table 5.10. 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 5.34.
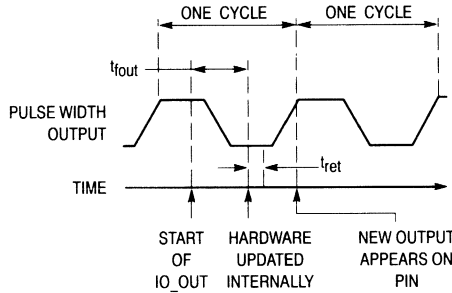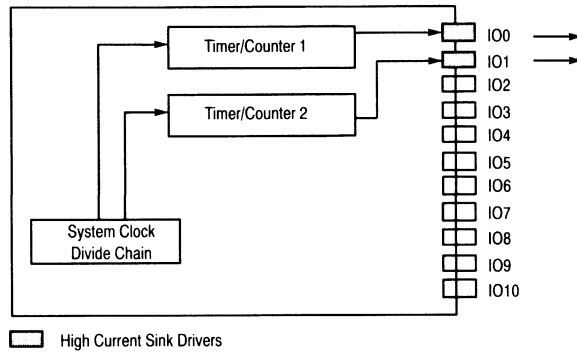
Symbol	Description	Typ @ 10 MHz
t_{fout}	Function call to output update	101 μs
t_{ret}	Return from function	13 μs

Figure 5.34. Pulse Width Output Latency Values

5.6.2.5 TRIAC OUTPUT.

On the MC143150, a timer/counter may be configured to control the delay of a 25 μs wide output pulse signal with respect to either a rising edge or a falling input edge as trigger. On the MC143120, this trigger may be either a rising edge, a falling edge, or both rising and falling edges. 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 5.8 shows the resolution and maximum range of the delay. See Figure 5.35.
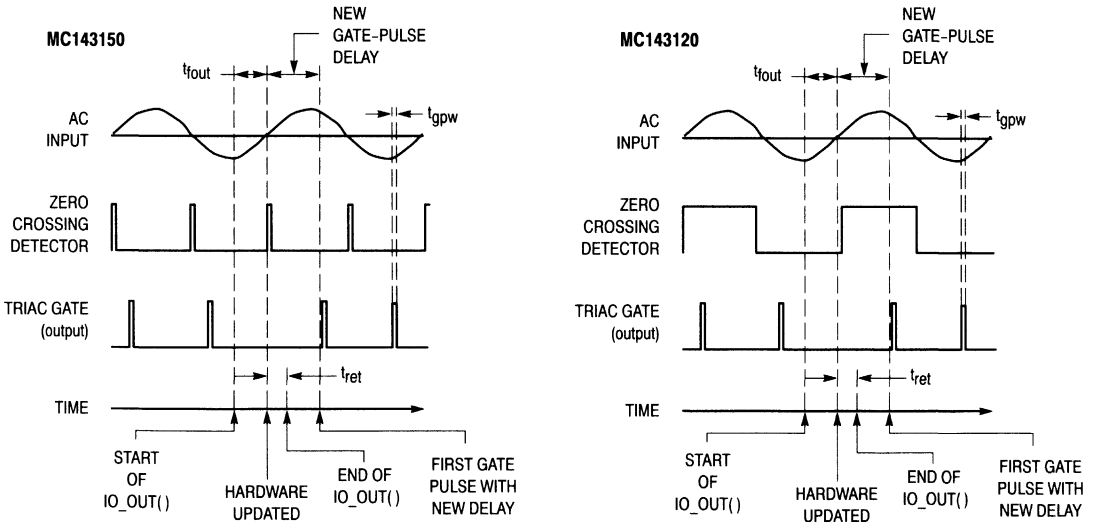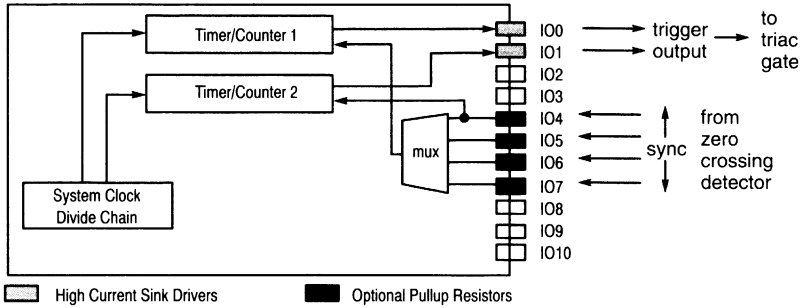
Symbol	Description	Typ @ 10 MHz
t_{fout}	Function call to first sync bit Min (first sample) Max (time out)	 185 μs 10.2 μs
t_{ret}	Return from function	17 μs
t_{gpw}	Trigger output pulse width	25 μs

Figure 5.35. Triac Output Latency Values

5.6.2.6 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 mode 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 5.36.
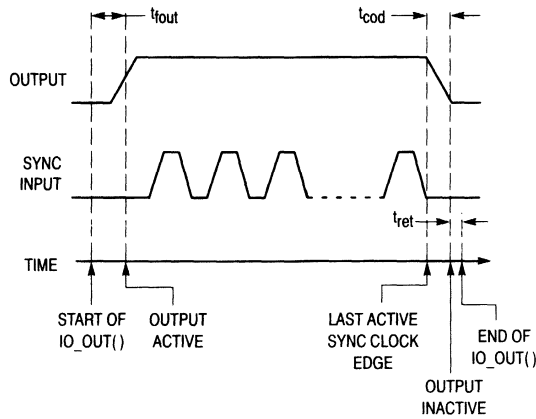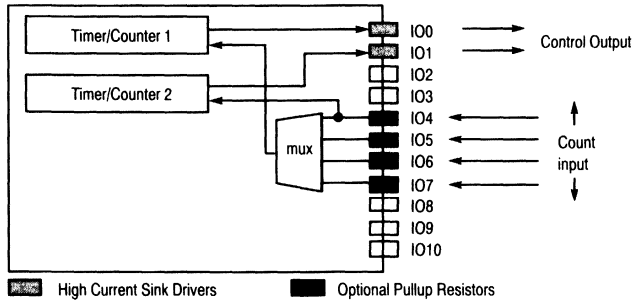
Symbol	Description	Typ @ 10 MHz
t_{fout}	Function call to output pulse	109 μs
t_{cod}	Last negative sync Clock edge to output inactive	min 550 ns max 750 ns
t_{ret}	Return from function	7 μs

Figure 5.36. Triggered Count Output Latency Values

5.7 NOTES

For on-time input and period input, the timer/counter returns a number in the range 0 to 65,535, representing elapsed times from zero up to the maximum range given in Table 5.8.

For one-shot output and triac output, the timer/counter may be programmed with a number in the range 0 to 65,535, representing waveform on-times from zero up to the maximum range given in Table 5.8. 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 5.8. Timer/Counter Resolution and Maximum Range

Clock Select (System Clock ÷)	Resolution (µs) (Clock Period)	Maximum Range (ms) (Max Count)
0 (÷1) (5 MHz)	0.2	13.11
1 (÷2) (2.5 MHz)	0.4	26.21
2 (÷4) (1.25 MHz)	0.8	52.42
3 (÷8) (625 kHz)	1.6	104.86
4 (÷16) (312.5 kHz)	3.2	209.71
5 (÷32) (156.25 kHz)	6.4	419.42
6 (÷64) (78.125 kHz)	12.8	838.85
7 (÷128) (39.06 kHz)	25.6	1,677.70

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

$$\text{Resolution (µs)} = 2^{(\text{Clock Select} + 1)} / \text{Input Clock (MHz)}$$
$$<\text{Maximum Range}> \text{(ms)} = 65.535 \times \text{Resolution (µs)}$$

For pulse-width short output and pulse-count output, Table 5.9 gives the possible choices for pulse-train repetition frequencies.

Table 5.9. Timer/Counter Square Wave Output

Clock Select (System Clock ÷)	Repetition Rate (Hz)	Repetition Period (µs)	Resolution (µs) of Pulse
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:

$$\text{Period (µs)} = 512 \times 2^{\text{Clock Select}} / \text{Input Clock (MHz)}$$
$$\text{Frequency (Hz)} = 1,000,000 / \text{Period (µs)}$$

For pulse-width long output, Table 5.10 gives the possible choices for pulse-train repetition frequencies.

Table 5.10. Timer/Counter Pulse Train Output

Clock Select	Frequency (Hz)	Period (ms)
0	76.29	13.11
1	38.15	26.21
2	19.07	52.43
3	9.54	104.86
4	4.77	209.72
5	2.38	419.43
6	1.19	838.86
7	0.60	1,677.72

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

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

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

MC143150•MC143120

SECTION 6
NEURON CHIP ELECTRICAL AND MECHANICAL SPECIFICATIONS

6.1 ELECTRICAL SPECIFICATIONS

6.1.1 Absolute Maximum Ratings (Voltages referenced to V_{SS})

Rating	Symbol	Value	Unit
Supply Voltage Range	V_{DD}	− 0.3 to 7.0 V	V
Input Voltage Range	V_{in}	− 0.3 to V_{DD} + 0.3	V
Maximum Source Current	I_{DD}	200	mA
Maximum Drain Current	I_{SS}	300	mA
Continuous Power Dissipation	P_D	1	W
Operating Temperature	T_A	− 40 to + 85	°C
Storage Temperature Range	T_{stg}	− 65 to + 150	°C

6.1.2 Recommended Operating Conditions (Voltages referenced to V_{SS}, T_A = − 40 to + 85°C)

Parameter	Symbol	Min	Max	Unit
Supply Voltage	V_{DD}	4.5	5.5	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}	0.8	V
CMOS High-Level Input Voltage	V_{IH}	V_{DD} − 0.8	V_{DD}	V
Operating Free-Air Temperature	T_A	− 40	+ 85	°C

6.1.3 Electrical Characteristics

(Excluding differential inputs CP0 and CP1 - TBD; V_{DD} = 4.5 to 5.5 V)

Parameter	Symbol	Min	Max	Unit
Input Low Voltage IO0–IO10, D0–D7, A0–A15, CP2–CP4 CP0, CP1 (Differential)	V_{IL}	— —	0.8 Programmable	V
Input High Voltage IO0–IO10, D0–D7, A0–A15, CP2–CP4 CP0, CP1 (Differential)	V_{IH}	2.0 Programmable	— —	V
Low-Level Output Voltage Standard Outputs* @ CMOS Level (I_{OL} = TBD) Standard Outputs* (I_{OL} = 1.4 mA) High Sink (IO0–IO3), $\overline{SERVICE}$, $\overline{RESET}$ (I_{OL}= 20 mA) Maximum Drive (CP2, CP3) (I_{OL}= 35 mA)	V_{OL}	— — — —	0.8 0.4 0.8 0.8	V
High-Level Output Voltage Standard Outputs* @ CMOS Level (I_{OH} = TBD) Standard Outputs* (I_{OH} = –1.4 mA) High Drive (IO0–IO3), $\overline{SERVICE}$ (I_{OH} = – 1.4 mA) Maximum Drive (CP2, CP3) (I_{OH} = – 35 mA)	V_{OH}	V_{DD} – 0.8 V 2.4 2.4 V_{DD} – 0.8 V	— — — —	V
Hysteresis (Excluding CLK1, $\overline{RESET}$)	V_{hys}	175	—	mV
Input Current (Excluding pullups)** (V_{SS} to V_{DD})	I_{in}	– 10	10	μA
Pullup Source Current (V_{out} = 0 V, Output = High-Z)	I_{in}	30	300	μA
Supply Current–Operating Mode (f = 10 MHz, quiescent)	I_{DD}	—	40	mA
Supply Current–Sleep Mode	I_{DD}	—	2	mA

* Standard outputs are A0–A15, D0–D7, IO4–IO10, CP0, CP1, CP4, $\overline{E}$, and R/$\overline{W}$. ($\overline{RESET}$ is a CMOS open drain input/output. CLK2 must not be used to drive external devices.)

** IO4–IO7 and $\overline{SERVICE}$ have configurable pullups. $\overline{RESET}$ has a permanent pullup.

6.2 MECHANICAL SPECIFICATIONS

6.2.1 MC143150 Pin Assignments

MC143150FU
64-LEAD QUAD FLAT-PACK

Top row (pins 48–33): NC, A15, $\overline{E}$, R/$\overline{W}$, V_{DD}, D0, D1, V_{DD}, V_{DD}, V_{SS}, D2, D3, D4, D5, D6, D7

Left side:
NC	49
A14	50
A13	51
A12	52
A11	53
A10	54
A9	55
A8	56
A7	57
A6	58
A5	59
A4	60
A3	61
A2	62
A1	63
A0	64

Top pin numbers: 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33

Right side:
32	CP4
31	CP3
30	CP2
29	CP1
28	CP0
27	NC
26	V_{DD}
25	V_{SS}
24	CLK1
23	CLK2
22	V_{DD}
21	V_{SS}
20	V_{DD}
19	V_{SS}
18	NC
17	$\overline{SERVICE}$

Bottom pin numbers: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Bottom row labels: NC, IO0, IO1, IO2, IO3, $\overline{RESET}$, V_{DD}, V_{SS}, V_{SS}, IO4, IO5, IO6, IO7, IO8, IO9, IO10

MC143150FN
68-LEAD CHIP CARRIER

Top row (pins 9–61): NC, A0, A1, A2, A3, A4, A5, A6, A7, A8, A9, A10, A11, A12, A13, A14, NC

Pin numbers top: 9 8 7 6 5 4 3 2 1 68 67 66 65 64 63 62 61

Left side		Right side	
NC	10	60	NC
IO0	11	59	A15
IO1	12	58	$\overline{E}$
IO2	13	57	$R/\overline{W}$
IO3	14	56	V_{DD}
$\overline{RESET}$	15	55	D0
V_{DD}	16	54	D1
V_{SS}	17	53	V_{DD}
V_{SS}	18	52	V_{DD}
IO4	19	51	V_{SS}
IO5	20	50	D2
IO6	21	49	D3
IO7	22	48	D4
IO8	23	47	D5
IO9	24	46	D6
IO10	25	45	D7
NC	26	44	NC

Pin numbers bottom: 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43

Bottom row: NC, SERVICE, NC, V_{SS}, V_{DD}, V_{SS}, V_{DD}, CLK2, CLK1, V_{SS}, V_{DD}, CP0, CP1, CP2, CP3, CP4, NC

6.2.2 MC143150 Package Dimensions

MC143150FU
PLASTIC
64-LEAD QUAD FLAT-PACK
CASE 840C-01

DETAIL "A"

DETAIL "C"

SECTION B–B

BASE METAL

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. ALLOWABLE DAMBAR PROTRUSION SHALL BE 0.08 (0.003) TOTAL IN EXCESS OF THE D DIMENSION AT MAXIMUM MATERIAL CONDITION. DAMBAR CANNOT BE LOCATED ON THE LOWER RADIUS OR THE FOOT.

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	2.067	2.457	0.081	0.097
D	0.30	0.45	0.012	0.017
E	2.00	2.40	0.078	0.094
F	0.30		0.012	
G	0.80 BSC		0.031 BSC	
H	0.067	0.250	0.003	0.009
J	0.130	0.230	0.005	0.009
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	9°	15°	9°	15°
V	16.20	16.60	0.638	0.654
W	0.042 NOM		0.016 NOM	
X	1.10	1.30	0.043	0.051

MC143150 Package Dimensions (continued)
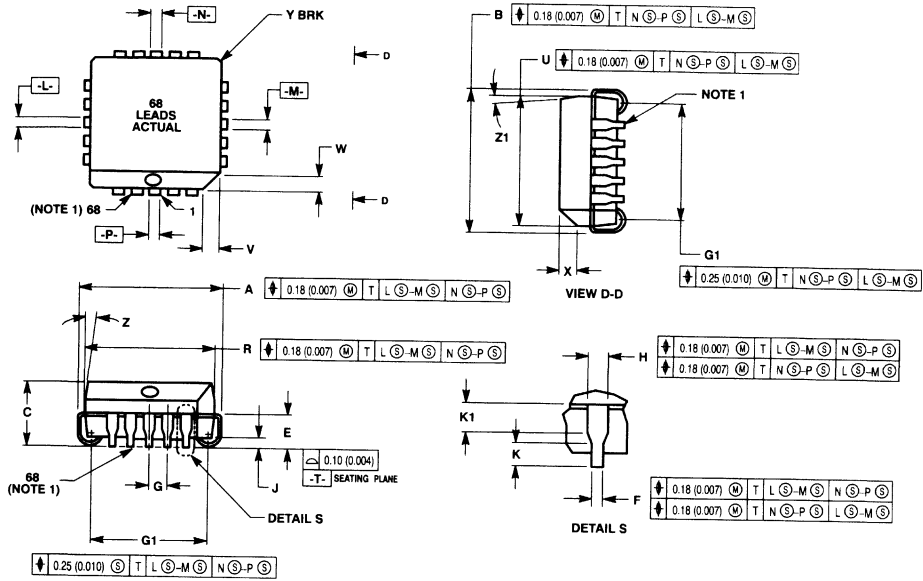
	MILLIMETERS		INCHES	
DIM	MIN	MAX	MIN	MAX
A	25.02	25.27	0.985	0.995
B	25.02	25.27	0.985	0.995
C	4.20	4.57	0.165	0.180
E	2.29	2.79	0.090	0.110
F	0.33	0.48	0.013	0.019
G	1.27 BSC		0.050 BSC	
H	0.66	0.81	0.026	0.032
J	0.51	—	0.020	—
K	0.64	—	0.025	—
R	24.13	24.28	0.950	0.956
U	24.13	24.28	0.950	0.956
V	1.07	1.21	0.042	0.048
W	1.07	1.21	0.042	0.048
X	1.07	1.42	0.042	0.056
Y	—	0.50	—	0.020
Z	2°	10°	2°	10°
G1	23.12	23.62	0.910	0.930
K1	1.02	—	0.040	—
Z1	2°	10°	2°	10°

NOTES:
1. **DUE TO SPACE LIMITATION, CASE 779-02 SHALL BE REPRESENTED BY A GENERAL (SMALLER) CASE OUTLINE DRAWING RATHER THAN SHOWING ALL 68 LEADS.**
2. DATUMS -L-, -M-, -N-, AND -P- DETERMINED WHERE TOP OF LEAD SHOULDER EXIT PLASTIC BODY AT MOLD PARTING LINE.
3. DIM G1, TRUE POSITION TO BE MEASURED AT DATUM -T-, SEATING PLANE.
4. DIM R AND U DO NOT INCLUDE MOLD PROTRUSION. ALLOWABLE MOLD PROTRUSION IS 0.25 (0.010) PER SIDE.
5. DIMENSIONING AND TOLERANCING PER ANSI Y14.5M, 1982.
6. CONTROLLING DIMENSION: INCH.

6.2.3 MC143150 Pad Layout

MC143150FU
64-LEAD QUAD FLAT PACK

0.5 mm
WIDTH

0.8 mm
PITCH

1.7 mm

14.0 mm
±0.1 mm

17.7 mm
±0.1 mm

MC143150FN
68-LEAD CHIP CARRIER

9   1

0.855 in.

0.075 in.

0.025 in.
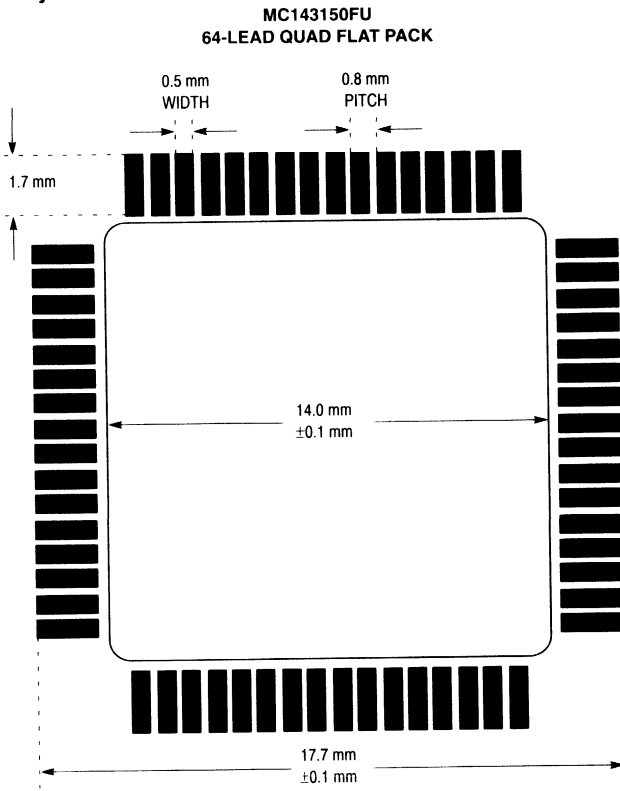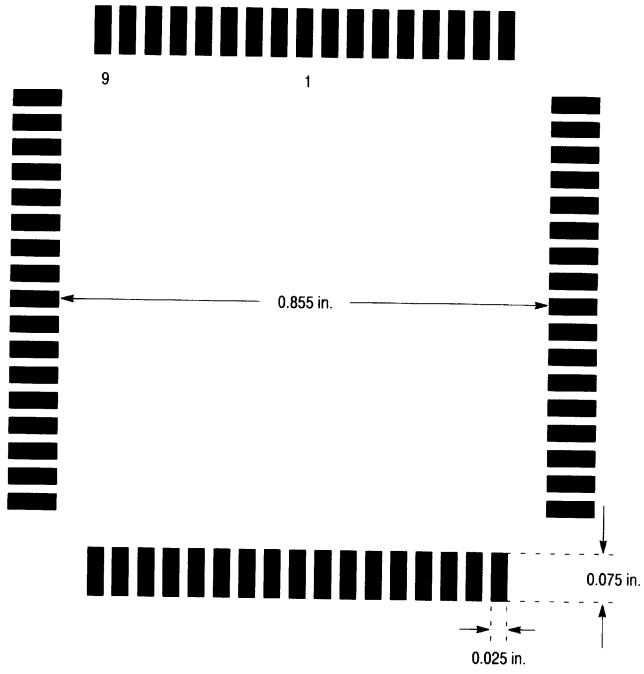
6.2.4 MC143120 Pin Assignment

MC143120DW
32-LEAD SOG

$\overline{\text{RESET}}$	1 •	32	V_{DD}
V_{DD}	2	31	V_{SS}
IO4	3	30	IO5
IO3	4	29	IO6
IO2	5	28	IO7
IO1	6	27	IO8
IO0	7	26	IO9
$\overline{\text{SERVICE}}$	8	25	V_{DD}
V_{SS}	9	24	IO10
V_{SS}	10	23	V_{SS}
V_{DD}	11	22	CP4
V_{DD}	12	21	CP3
V_{SS}	13	20	CP1
CLK2	14	19	CP0
CLK1	15	18	V_{DD}
V_{SS}	16	17	CP2
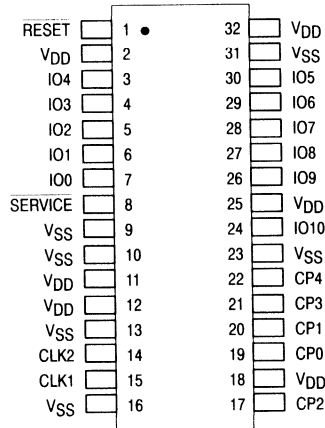
6.2.5 MC143120 Package Dimensions

MC 143120DW
PLASTIC
32-LEAD SOG
CASE 855-01
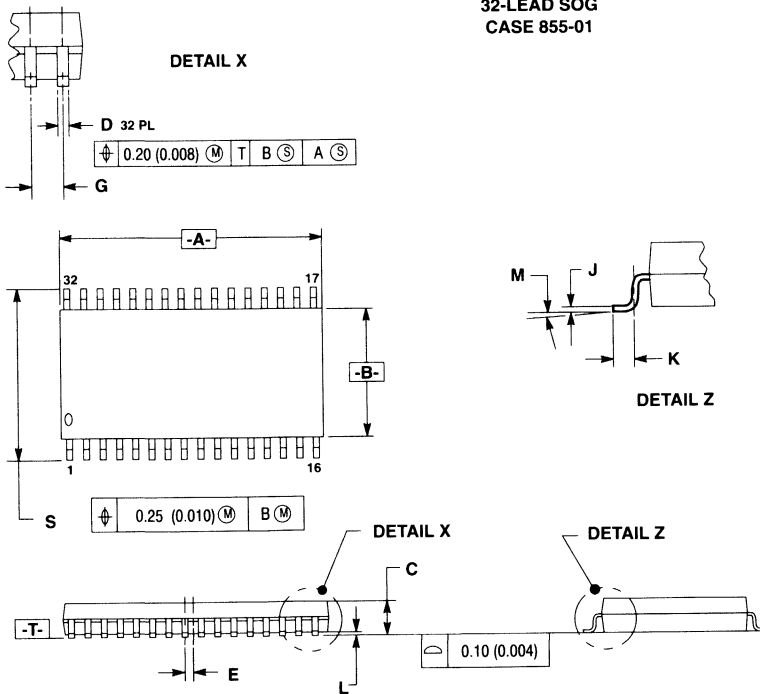
DETAIL X

D 32 PL

⊕ 0.20 (0.008) Ⓜ T B Ⓢ A Ⓢ

G

-A-

32   17

-B-

1   16

S   ⊕ 0.25 (0.010) Ⓜ B Ⓜ

DETAIL X   DETAIL Z

M J

K

DETAIL Z

C

-T-   0.10 (0.004)

E   L

NOTES:
1. DIMENSIONING AND TOLERANCING PER ANSI Y14.5M, 1982.
2. CONTROLLING DIMENSION: MILLIMETERS.
3. DIMENSION A AND B DO NOT INCLUDE MOLD PROTRUSION.
4. MAXIMUM MOLD PROTRUSION 0.15 (0.006) PER SIDE.

DIM	MILLIMETERS		INCHES	
	MIN	MAX	MIN	MAX
A	20.40	20.70	0.803	0.815
B	11.10	11.30	0.437	0.445
C	2.75	3.04	0.108	0.120
D	0.35	0.50	0.014	0.020
E	0.64 BSC		0.025 BSC	
G	1.27 BSC		0.050 BSC	
J	0.14	0.32	0.006	0.012
K	0.60	1.00	0.024	0.039
L	0.10	0.35	0.004	0.014
M	0 °	8 °	0 °	8 °
S	13.80	14.40	0.543	0.567

6.2.6 MC143120 Pad Layout

MC143120DW
32-LEAD SOG
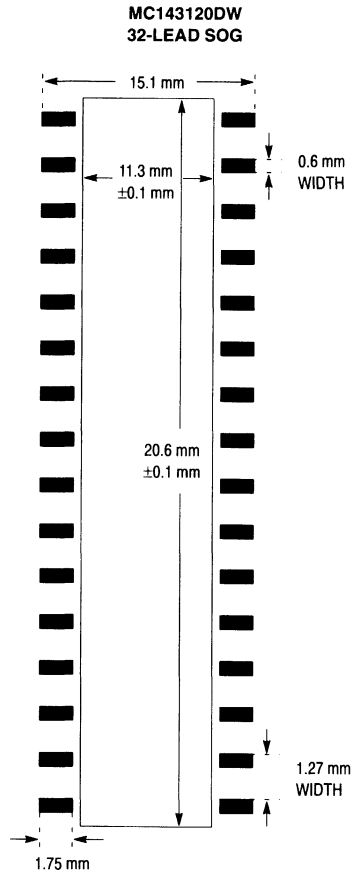
6.2.7 Sockets for NEURON CHIPS

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

Integrated Circuit	Manufacturer	Part Number
MC143120	Yamaichi	IC51-0322-667-2
MC143150	Enplas	FPQ-64-0.8-10A

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 run-time library supporting file I/O and other features common to larger target processors, such as floating point. However, NEURON C has a special run-time 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 Programming Guide*.

7.1 TIMERS

The application program can use up to fifteen 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 (see below). This event is called *timer_expires*. The value of a timer variable is an unsigned long (0–65,535).

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 LON-MANAGER Applications Programming Interface (API) both 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 EEPROM, and hence may be written after the node has been manufactured.

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 to it. 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 the network variable outputs from one node to inputs on other nodes 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 need not deal with message buffers, node addressing, request/response/retry processing, and other low-level details.

A node running a regular application program may have up to 62 network variables, including array elements. 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. 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). A node running a microprocessor interface program may have 4095 network variables. The Microprocessor Interface Program is a special system image used to attach the node to a host processor. In this case, the network variable information is located in the host's memory.

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

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

Network variables may be specified as authentic, 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 time slot is used to transmit their values (see Section 8.7).

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

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

7.3 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, 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:

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

Request/response service allows the node receiving a message to respond with data in the response message, as distinct from an acknowledgement, 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 time slot on the channel if the node has a priority slot allocated to it. In any case, 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 run-time library support routines:

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

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

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

msg_in	Currently received message
msg_out	Message to be sent
resp_in	Currently received response to a previous message

7.4 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 body is executed. The user has the capability of specifying one or more *when* clauses as having priority, and the scheduler checks priority *when* clauses, one per scheduler cycle, in order of their appearance in the NEURON C program, followed by the non-priority *when* clauses in a round-robin fashion. The task scheduler can handle up to eighty *when* clauses, depending on type.

Events fall into five classes:

> System Wide Events:

>> *reset*   Node has been reset
>> *offline*   Node has been set off-line
>> *online*   Node has been set on-line
>> *flush_completes*   Node has completed preparations to enter sleep mode
>> *wink*   Node has received 'wink' network management message

> Input/Output Events:

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

> Timer Events:

>> *timer_expires*   Software timer value has decremented to zero

> Message and Network Variable Events:

> The following events associated with the transmission of network variables and messages are discussed above 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

7.5 ADDITIONAL LIBRARY FUNCTIONS

The following miscellaneous functions are available from the application program:

Execution Control:

delay	Delay processing for a time independent of input clock rate
flush_cancel	Cancel a flush in progress
flush_wait	Wait for outgoing messages and updates to be sent before going off-line
go_offline	Cease execution of the application program
post_events	Defines a critical section for network variable and message processing
power_up	Determine whether last processor reset was due to power up
scaled_delay	Delay processing for a time that depends on the input clock rate
watchdog_update	Tickle the watch-dog timer to prevent node reset

Network Management Control:

access_address	Read node's address table
access_domain	Read node's domain table
access_nv	Read node's network variable configuration table
go_unconfigured	Reset this node to an uninstalled state
offline_confirm	Inform network management node that this node is going off-line
update_address	Write node's address table
update_domain	Write node's domain table
update_nv	Write node's network variable configuration table

Error Handling:

application_restart	Begin application program over again
clear_status	Clear error statistics accumulators and error log
error_log	Record software-detected error
node_reset	Activate NEURON CHIP reset pin, and reset all processors
retrieve_status	Read error statistics from protocol processor
retrieve_xcvr_status	Read transceiver status register

Sleep Mode:

flush	Flush all outgoing messages and network variable updates
sleep	Enter low-power mode by disabling system clock
timers_off	Turn off all software timers

Utilities:

abs	Arithmetic absolute value
bcd2bin	Convert binary coded decimal data to binary
bin2bcd	Convert binary data to binary coded decimal
max	Arithmetic maximum of two values
min	Arithmetic minimum of two values
muldiv(s)	Multiply/divide with 32-bit intermediate result–(un)signed
random	Generate eight-bit random number
refresh_memory	Rewrite contents of EEPROM memory
reverse	Reverse the order of bits in an eight-bit number

Input/Output (see Section 8 for details):

io_in	Input data from I/O object
io_out	Output data to I/O object
io_out_request	Request ready indication from parallel I/O object
io_change_init	Initialize reference value for *io_changes* event
io_select	Set timer/counter multiplexer
io_set_direction	Change direction of I/O pins
io_set_clock	Set timer/counter clock rate

7.6 BUILT-IN VARIABLES

input_value	Data read by last explicit or implicit *io_in( )* call
input_is_new	True if last input from a timer/counter object read an updated value
nv_array_index	Index of element in network variable array with updated value

SECTION 8
LONTALK PROTOCOL

The NEURON CHIP implements a complete networking protocol using two of the three on-chip processors. This networking protocol follows the ISO OSI reference model for network protocols; this 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. The three processors in the NEURON CHIP are used to execute the protocol software along with the application program. 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 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:

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, radio-frequency, infrared, coaxial cable, and fiber optics.

8.2 SUPPORT FOR MULTIPLE COMMUNICATION CHANNELS

A channel is a physical transport medium for packets. 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. These devices 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 multimedia networks may be constructed, and network loading can be optimized by localizing traffic.

8.3 COMMUNICATIONS RATES

Channels may be configured for different data rates to allow trade-offs of distance, throughput, and power consumption. The allowable bit rates are 4.9, 9.8, 19.5, 39.1, 78.1, 156.3, 312.5, 625, and 1,250 kb/s.

Channel throughput depends on the data rate, the average size of a packet, and the use of acknowledgments, priority, and authentication (see below). An average packet is in the range of 10 to 16 bytes long, depending on the length of the domain identifier, the addressing mode, size of the data field for a network variable update or an explicit message.

8.4 LONTALK ADDRESSING LIMITS

The 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.

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 255x127 = 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 interdomain gateway. This also allows, for example, a single sensor node to transmit its outputs into two different domains.

Nodes may also be grouped. Groups of nodes may span several subnets within a domain. Groups may also span several 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. Group addressing reduces the number of bytes of address information transmitted with each message, and also allows many nodes to receive a piece of 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 (transmission medium) of a node does not affect the way a node is addressed. Domains can contain several channels. Subnets and groups may also span several 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

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 the time-out 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 time-out 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 unacknowledcged. 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, in which the traffic generated by all the responses would overload the network.

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.

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. This is used to prevent unauthorized access to nodes and their applications. Authentication 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 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 time slots on a channel dedicated to priority nodes. Each priority time slot 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.

8.10 FOREIGN FRAMES

A special range of message codes is reserved for foreign frame transmission. Up to 229 bytes of data may be embedded in a message packet and transmitted like any other message. The LONTALK protocol applies no special processing to foreign frames. They 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 A for more detail on these services.

APPENDIX A
NEURON CHIP CONFIGURATION DATA STRUCTURES

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 NEURON 3120 CHIP, this software is in the on-chip 10K ROM. In the NEURON 3150 CHIP, this software is in an external ROM. For the NEURON 3150 CHIP, this software is provided as part of the LONBUILDER Developer's Workbench environment. With this tool, the user can produce Intel Hex or Motorola S-record files containing the system image so that EPROM devices may be programmed.

In the NEURON 3120 CHIP, the following NEURON C runtime library functions 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_domain`, `update_nv`, use of a signed bitfield, `memcpy` or structure assignment (length ≥ 256 bytes), `memset` (length ≥ 256 bytes), `memcpy from msg_in.data`, `memcpy to resp_out.data`, ability to send explicitly addressed messages, NEUROWIRE 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 external interface data
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 NEURON 3150 CHIP, the application image is typically programmed into an external ROM, or downloaded over the network to EEPROM memory. In the NEURON 3120 CHIP, the application image is downloaded into the on-chip EEPROM memory. The LONBUILDER Developer's Workbench supports creation of application images.

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 run-time library calls and declarations for the purpose of node self-installation. In the LONBUILDER 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 API application programmer's interface 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.

The NEURON CHIP configuration data structures may be divided into six main sections:

1. A fixed structure, whose size is independent of the application on the node.
2. A domain table, with an entry for every domain this node belongs to.
3. An address table, with an entry for every network address referenced by this node.
4. Network variable tables, with entries for every network variable defined by this node.
5. Optional self-identification and self-documentation information describing this node and its network variables.
6. A channel configuration structure, defining the transceiver interface of the node.

A.1 FIXED READ-ONLY DATA STRUCTURE

This structure is physically located at the start of on-chip EEPROM, at location 0xF000. It 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 ];
      unsigned   model_num;
      unsigned   minor_model_num : 4;
      unsigned                   : 4;
const nv_fixed_struct * nv_fixed;
      unsigned   read_write_protect : 1;
      unsigned                   : 1;
      unsigned   nv_count        : 6;
const snvt_struct * snvt;
      unsigned   id_string[ ID_STR_LEN ];
      unsigned                   : 1;
      unsigned   two_domains     : 1;
      unsigned                   : 6;
```

```
        unsigned   address_count   : 4;
} read_only_data_struct;

const read_only_data_struct read_only_data;
```

The application program may read, but not write this structure, using the global declaration `read_only_data`. The structure is 21 bytes long, and it may be read and mostly written (except for the first eight bytes) over the network using the Read Memory and Write Memory network management messages with `address_mode=1`.

A.1.1 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. The first byte is a manufacturer number (0 for Toshiba and 1 for Motorola). All other values are reserved. The next four bytes form a manufacturer-assigned sequence number. The last byte is an Echelon-assigned block number.

```
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 these fields is: NEURON 3150 CHIP = 0 , NEURON 3120 CHIP = TBD.

```
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, this pointer is undefined.

```
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. The protected data includes the application code and the network variable fixed table. This includes the `read_write_protect` bit itself, so that once set, the bit may not be reset over the network. The application program may set this bit with the NEURON C compiler directive `#pragma read_write_protect`.

```
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.

```
const snvt_struct * snvt;              // offset 0x0B
```

This field is a pointer to the data structure giving 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 executing a microprocessor interface program and acting as a communications chip, 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:cs:ss:nn:nn
```

The second format is reserved for Echelon-certified application nodes. The bit assignment for this format is as follows:

The first 4 bits are the format code (0x8 – 0xF).
The next 20 bits are the manufacturer code.
The next 12 bits are the device class.
The next 12 bits are the device sub-class.
The last 16 bits are the manufacturer-assigned model number.

The encoding of all these fields is TBD. 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    two_domains    : 1;      // offset 0x15
```

This bit specifies that the domain table has two entries (see Section A.2). If this bit is zero, the domain table has one entry. This bit is set unless the NEURON C compiler directive `#pragma one_domain` was specified in the application program.

```
unsigned    address_count  : 4;
```

This field specifies the number of entries (0–15) in the address table (see Section A.3).

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;
      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 );
```

The application program may read or write any entry in this table using the access routines `access_domain` and `update_domain`. 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.

A.2.1 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 six bytes, it is left justified in this field. In the development environment, the user specifies the size and value of the domain ID when creating the domain object.

```
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.

```
unsigned    node;
```

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 six-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. 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 five 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 addressing is not used by the LONBUILDER binder or the API binder 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 transmit 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). 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. Zero is not a valid node ID.

```
unsigned   subnet    : 8;
```
This field specifies the subnet ID (1–255) within the specified domain. Zero 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             : 7;
        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   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 acknowledgement is received. Subsequent responses or acknowledgements are discarded.

A.3.7 Declaration of Turnaround Address Format

```
    typedef struct {
        addr_type  type;                     // offset 0x00
        unsigned   turnaround;               // offset 0x01
    } turnaround_struct;
```

A.3.8 Turnaround Address Field Descriptions

```
    addr_type  type;
```

Contains the value UNBOUND (0).

```
    unsigned   turnaround;
```

This field contains the value one. If the turnaround field is zero, this address table entry is not in use.

A.3.9 Declaration of Unique 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 Unique 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.

```
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 zero 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 in Table A.1.

```
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 in Table A.1.

```
unsigned   tx_timer  : 4;
```

This field specifies the time interval between retries when acknowledged or request/response service is used. The retry timer is restarted when each attempt is made, and also when any acknowledgment or response (except for the last one) is received. The encoding of this field is specified in Table A.1.

Table A.1. 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 TABLES

There are two tables associated with network variables: the network variable configuration 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, and is part of the network image written during node installation. The network variable fixed table defines the compile-time attributes of the network variables in this node. It may be located in read-only memory, and is part of the application image written during node manufacture.

Declarations from ACCESS.H

```
#define MAX_NVS 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;

const nv_struct * access_nv( int index );
void update_nv( const nv_struct * nv_entry, int index );
int nv_table_index( network_variable_name );

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 and write any entry in the network variable configuration table using the access routines `access_nv` and `update_nv`. The index of the table entry corresponding to any network variable declared in the program may be determined with the function `nv_table_index`. The base address of the network variable fixed table may be retrieved from the configuration data field `config_data.nv_fixed`.

Each table consists of up to 62 entries, each three bytes in length in either table. 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 not limited to 62. 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 Config network management messages. The network variable fixed table may not be written, except when downloading the application image.

A.4.1 Network Variable Configuration Table Field Descriptions

```
unsigned    nv_priority      : 1;
```

This bit is set to one if the network variable uses priority messaging. Specified by `bind_info(priority | nonpriority)` in the NEURON C declaration of the network variable. In the development environment, this may be overridden in the connection screen.

```
unsigned    nv_direction     : 1;
```

This bit is set to one if this is an output network variable, zero if an input. This bit must not be changed by the application program or a Write Memory network management message.

```
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. The network variables on any one node must all have different selectors, unless they are turnaround network variables bound to each other. The binder in the development system ensures this by assigning a network variable selector to the union of all the connection sets that this network variable participates in. This, however, is not required by the protocol.

```
unsigned    nv_turnaround    : 1;
```

This bit is set to one 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. Specified by `bind_info(ackd | unackd_rpt | unackd)` in the NEURON C declaration of the network variable. In the development environment, this may be overridden in the 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 one if this network variable uses authenticated transactions. Specified by `bind_info(authenticated | nonauthenticated)` in the NEURON C declaration of the network variable. In the development environment, this may be overridden in the connection screen.

```
    unsigned  nv_addr_index    : 4;
```

This field specifies the index into the address table for this network variable (0–14). If the network variable is not associated with an address table entry, then the value 15 is used.

A.4.2 Network Variable Fixed Table Field Descriptions

```
    unsigned  nv_sync          : 1;
```

This bit is set to one if this is a synchronous network variable. 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 (SNVT) STRUCTURES

There are four structures associated with self-identification and self-documentation: a fixed size SNVT structure, a table containing a self-identification descriptor for each network variable, a self-documentation string for the node, and self-documentation information for network variables. If the NEURON C compiler directive #pragma disable_snvt_si is specified in the application program, none of this information is present. These tables may be located in read-only memory, and form part of the application image written during node manufacture.

Declarations from ACCESS.H

```
    typedef struct {
        unsigned long length;            // offset 0x00
        unsigned          num_netvars;   // offset 0x02
        unsigned          version;       // offset 0x03
        unsigned          mtag_count;    // offset 0x04
    } 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;
```

The base address of the SNVT structure may be retrieved from the configuration data field config_data.snvt. This structure is five bytes long. The SNVT descriptor table follows immediately after the SNVT structure, and it has one entry for every network variable declared in the application program. Each element of a network variable array has its own entry. Each entry in the SNVT descriptor table is two 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 255 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. All the self-identification and self-documentation information may be read over the network with the Read Memory network management command using `address_mode=0`.

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.

```
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        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 one if this network variable has an extension record following the node's self-documentation string.

```
unsigned   nv_sync               : 1;
```

This bit is set to one if this is a synchronous network variable. Specified with the modifier sync in the NEURON C declaration of the network variable.

```
unsigned   nv_polled             : 1;
```

This bit is set to one if this network variable is polled. If it is an output network variable, this is specified with the modifier polled 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 one if the network management node should take the node off-line before this network variable is updated. 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 one if the service type of this network variable (ACKD, UNACKD, UNACKD_RPT) may be modified by a network management message. 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 one if the priority of this network variable may be modified by a network management message. 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 one if the authentication of this network variable may be modified by a network management message. 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 one if this is a configuration network variable whose value is stored in EEPROM. 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 Standard Network Variable Types, see *The SNVT Guide*. If this field is zero, the network variable is of a non-standard type.

A.5.3 SNVT Table Extension Records

Extension records may be present for only some of the network variables. If an extension record is present, the field ext_rec is set to one 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 one-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 most significant bit.

```
unsigned    mre : 1;
```
If this bit is set to one, 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 one, 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 one, 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

21 bytes if a network variable array element. Each array element has its own extension record containing the name of the array, a left square bracket, one or two decimal digits denoting the index of the element, and a right square bracket.

```
unsigned   sd : 1;
```

If this bit is set to one, 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.

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;
```

The application program may read, but not write this structure using the global declaration `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`.

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 development environment, this is assigned automatically when the channel object is created. The NEURON CHIP firmware does not reference this field, but it is available to assist in recreation of the network topology data structure by interrogating the nodes.

```
char            location[ LOCATION_LEN ];
```

The location field is used to pass a six-byte ASCII string describing the physical location of the node to the network management node. In the development environment, this 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.2 shows the transceiver bit rate as a function of the input_clock field and the comm_clock field.

Table A.2. Transceiver Bit Rate (kbit/s) as a Function of comm_clock and input_clock

comm_clock	ratio	input_clock				
		5	4	3	2	1
0	8:1	1,250	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	—
6	512:1	19.5	9.8	4.9	—	—
7	1,024:1	9.8	4.9	—	—	—

```
unsigned        input_clock         : 3;
```

This field specifies the NEURON CHIP input clock (oscillator frequency). In the development environment, the user specifies this value in the hardware properties screen for the node. The encoding is as follows:

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 development environment, this is specified in the channel screen. The encoding is as follows:

1	Single-ended
5	Differential
2	Special-purpose

```
unsigned     comm_pin_dir      : 5;
```
This field specifies the direction of the NEURON CHIP's communications port pins. Zero indicates an input, one indicates an output with respect to the NEURON CHIP. The least significant bit corresponds to pin CP0. Values used in this field include:

0x0E	Direct mode — single-ended
0x0C	Direct mode — differential
0x1E	Special-purpose — wake-up pin is an output
0x17	Special-purpose — wake-up pin is an input

```
unsigned     reserved          [ 5 ];
```
This field specifies the raw transceiver parameters in units of processor cycles. In the development environment, these parameters are specified as the Raw Data in the channel screen. The descriptions are:

reserved[0]	preamble length pad — determines the length of the preamble for direct mode transceivers
reserved[1]	packet cycle pad — determines the packet cycle duration for counting down the backlog
reserved[2]	beta pad — the pad for the beta loop
reserved[3]	transmit beta 1 pad — the pad for the beta 1 period after transmitting
reserved[4]	receive beta 1 pad — the pad for the beta 1 period after receiving

```
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 zero. In the development 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 one. In the development 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 development environment, these are specified as General Purpose Data in the channel screen. The most significant bit of the first transceiver parameter is defined to be the alternate channel bit. The alternate channel is used for the later retries with acknowledged or request/response service. In the case of the powerline transceiver, this bit determines whether the transceiver uses QPSK modulation at 9600 bps (bit is zero), or BPSK modulation at 4800 bps (bit is one). All other transceiver parameters are user-defined. For direct-mode transceivers, the first two bytes are overlaid by the direct mode parameter structure defined below.

```
unsigned     non_group_timer  : 4;
```
When the node receives a unicast (non-group) message requiring a response or acknowledgement, 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. In the development environment, this is implicitly set to the maximum of the non-group receive timers specified in the connection screens. The encoding is of this field is in Table A.1.

```
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 node. In the development environment, this is defined in the node specification screen.

```
unsigned     preemption_timeout  :3;
```

This field specifies the maximum time the node should wait for a free buffer. In the development 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 development environment, these parameters are specified in the channel screen.

```
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 one 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 12.6) 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 12.5) for details of the available hysteresis filter settings.

```
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
Application Responses	0x00 – 0x3E
Response if node is off-line	0x3F
Foreign Messages	0x40 – 0x4E
Foreign Responses	0x40 – 0x4E
Response if node is off-line	0x4F

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 application or foreign message and response codes.

Network Diagnostic Messages	Request Code	Success Response	Failed Response
Query Status	0x50	0x30	0x10
Clear Status	0x51	0x31	0x11
Proxy Command	0x52	0x32	0x12
Query XCVR Status	0x53	0x33	0x13

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	—	—
Memory Refresh	0x71	0x31	0x11
Query SNVT	0x72	0x32	0x12
Network Variable Fetch	0x73	0x33	0x13

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 cannot 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.

Network management and network diagnostic messages may be delivered using Request/ Response service (except for *mode on-line*, *mode off-line* and *wink*, which 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 acknowledgement 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.

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 B.1.5) to retrieve the 6-byte NEURON ID from another NEURON CHIP at offset 0x0000 into the Read-Only Structure:

```
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 cannot 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.

B.1 NETWORK MANAGEMENT MESSAGES

These messages are described in six main groups: node identification messages, domain table messages, address table messages, messages related to network variables, node memory 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 relative to the Read-Only Structure (see A.1), or the Configuration Structure (see A.6). 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).

Message declarations:

```
typedef struct {
    enum {
    unconfigured        = 0,
        selected        = 1,
        selected_uncnfg = 2,
    } selector;
    enum {                          // Begin optional fields
        read_only_relative= 1,
        config_relative   = 2,
    } mode;
    unsigned long offset;
    unsigned count;
    unsigned data[ ];
} NM_query_id_request;

typedef struct {
    unsigned neuron_id[ NEURON_ID_LEN ];
    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 both. A node may be selected with the Respond to Query message. The response message contains the 6-byte NEURON ID and the 8-byte program ID.

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.

Service Pin Message (Unsolicited)

This is an unsolicited message sent by a node when service pin is grounded. It contains the node's unique ID followed by the program ID.

Message declaration:

```
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 A.1.1 for details on these values.

B.1.2 Domain Table Messages

Update Domain

This message overwrites a domain table entry with a new value, and recomputes the configuration check-sum. This assigns a domain, subnet and node identifier to the node, as well as an authentication key for that domain. The authentication key is transmitted in the clear, so that this message should not be used on an open network if authentication protection is desired. 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 A.2.

Message declaration:

```
typedef struct {
    unsigned domain_index;
    unsigned id[ DOMAIN_ID_LEN ];
    unsigned subnet;
    unsigned must_be_one : 1; // 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 \LB\INCLUDE \ACCESS.H. The most significant bit of the byte containing the node ID must be set.

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 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 A.2.

Message declaration:

```
    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 in the clear on the network. This message is honored even if the node is read/write protected. For a description of the domain table, see A.2.

Message declaration:

```
    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 six 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 A.3.

Message declaration:

```
    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 `\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 most significant bit 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 A.3.

Message declarations:

```
    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 most significant bit 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 A.3.

Message declaration:

```
    typedef struct {
        unsigned type           : 1;    //  must be one
        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 B.3.

Update Net Variable Config

This message overwrites a network variable configuration 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 A.4. For a node running the Microprocessor Interface Program 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 two bytes in the message form a 16-bit network variable index, allowing up to 16,383 network variables to be bound.

```
    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 `\LB\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, see A.4. For a node running the Microprocessor Interface Program 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 two bytes in the message form a 16-bit network variable index, allowing up to 16,383 network variable configuration table entries to be queried.

Message declaration:

```
    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 \LB\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 running the Microprocessor Interface Program. This program is a special application used to attach the node to a host processor at Layer 4 of the protocol. In this case, 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 a NEURON CHIP running a regular application program, 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 A.1). For a description of the SNVT Structure, see A.5.

Message declarations:

```
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 off-line. For a description of the network variable tables, see A.3. The normal way to poll a network variable value is with an application message specifying the network variable's selector (see B.3). For a node running the Microprocessor Interface Program, 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 two bytes in the message form a 16-bit network variable index, allowing up to 16,383 network variables to be fetched.

Message declarations:

```
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 A.1), relative to the Configuration Structure (see A.6), 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 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, the SNVT Structures (see A.5) and the Configuration Structure.

Message declaration:

```
typedef struct {
    enum {
        absolute          = 0,
        read_only_relative= 1,
        config_relative   = 2,
      } 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, two 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.

Write Memory

This message writes data to the node's memory. Addresses may be specified relative to the Read-Only Structure (see A.1), relative to the Configuration Structure (see 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. 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,
     } 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, two 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.

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 application-less state. 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.

Message declaration:

```
typedef enum {
    both_cs = 1,
    cnfg_cs = 4,
} 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. Up to eight bytes may be written if the node is on-line (executing the application program) and up to 38 bytes if it is off-line. 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 cannot 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:

```
typedef struct {
    unsigned long offset;
    unsigned count;
    enum {
        on_chip  = 0,
        off_chip = 1,
    } which;
} NM_memory_refresh_request;
```

The request message consists of two 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)

This message puts the application in an off-line or on-line mode, or resets the node. This message may also be used to change the state of the node. The state may be:

Application-less and unconfigured	(Service LED on full)
Unconfigured (but with an application)	(Service LED flashing)
Configured, hard off-line (a reset leaves the node off-line)	(Service LED off)
Configured, soft off-line (a reset puts the node on-line)	(Service LED off)
Configured, on-line	(Service LED off)

When the application is off-line, the scheduler is disabled. Polling a network variable will return no data, but network variable updates will be processed. However, if the application is off-line, nv_update_occurs events will be lost. If this message changes the mode to off-line or on-line, the appropriate NEURON C task (if any) will be executed. *Mode on-line* and *mode off-line* messages should not be delivered with Request/Response service.

Message declaration:

```
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
    } 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 off-line state, put it on-line, reset it, or change the state of the NEURON CHIP. In this last case, there is a second byte specifying which state the node should enter.

Wink (Any service class except Request/Response)

This message causes the node to execute the *wink* clause in application program (if any). This may be used to identify nodes visually or audibly if it is more convenient than grounding the service pin. There is no data associated with this message.

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 run-time 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;
    unsigned long transaction_timeouts;
    unsigned long rcv_transaction_full;
    unsigned long lost_msgs;
    unsigned long missed_msgs;
    unsigned reset_cause;
    enum {
        appl_uncnfg    = 2,
        no_appl_uncnfg = 3,
        cnfg_online    = 4,
        cnfg_offline   = 6,         // hard offline state
        soft_offline   = 0xC,
    } node_state;
     unsigned version_number;
    enum {
        bad_event                = 1,
        nv_length_mismatch       = 2,
        nv_msg_too_short         = 3,
        eeprom_write_fail        = 4,
        bad_address_type         = 5,
        preemption_mode_timeout  = 6,
        already_preempted        = 7,
        sync_nv_update_lost      = 8,
        invalid_resp_alloc       = 9,
        invalid_domain           = 10,
        read_past_end_of_msg     = 11,
        write_past_end_of_msg    = 12,
        invalid_addr_table_index = 13,
        incomplete_msg           = 14,
        nv_update_on_output_nv   = 15,
        no_msg_avail             = 16,
        illegal_send             = 17,
```

```
        unknown_PDU                      = 18,
        invalid_nv_index                 = 19,
        divide_by_zero                   = 20,
        invalid_appl_error               = 21,
        memory_alloc_failure             = 22,
        write_past_end_of_net_buffer     = 23,
        appl_cs_error                    = 24,
        cnfg_cs_error                    = 25,
        invalid_xcvr_reg_addr            = 26,
        xcvr_reg_timeout                 = 27,
        write_past_end_of_appl_buffer    = 28,
        io_ready                         = 29,
        self_test_failed                 = 30,
        subnet_router                    = 31,
    } error_log;
    unsigned model_number;
} 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 acknowledgements 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. These may be due to excessively long receive timers (see 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. 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.

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.

The response message also contains a byte with the cause of the last reset as follows (X = don't care):

Power–up reset	0bXXXXXXX1
External reset	0bXXXXXX10
Watchdog reset	0bXXXX1100
Software reset	0bXXX10100

This is followed by a byte containing the current state of the node. The state may be:

Application-less and unconfigured	(Service LED on full)
Unconfigured (but with an application)	(Service LED flashing)
Configured, hard off-line (a reset leaves the node off–line)	(Service LED off)
Configured, soft off-line (a reset puts the node on–line)	(Service LED off)
Configured, on-line	(Service LED off)

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. Zero means that no error has been detected since the last reset.   For a description of the firmware errors, see the *NEURON C Programmer's Guide*, Appendix E.

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 the target node is out of earshot 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 `\LB\INCLUDE\MSG_ADDR.H`. See 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 most significant bit set. The response message is the response appropriate to the specified status request.

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 7.3).

Message declaration:

```
typedef struct {
    unsigned xcvr_params[ NUM_COMM_PARAMS ];
} ND_query_xcvr_response;
```

The request message contains no data. The response message contains seven bytes with the transceiver status register values.

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 most significant bit 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.

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.

Network Variable Update (Acknowledged, Unacknowledged, or Unacknowledged/Repeated)

The code field of this message contains the most significant six bits of the network variable selector. The most significant bit of the code field is set, indicating a network variable message, and the second most significant bit is clear, indicating that the update is addressed to an input network variable. The first data byte of the message contains the least significant eight 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.

Example of updating a two-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 cannot 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 Microprocessor Interface Program can be used.

Network Variable Poll (Request/Response Only)

For completeness, the network variable poll message format is described here, although this message is not very useful in a normal NEURON C application program. This is because the response to the poll will be processed by the network processor, and cannot 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. 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 B.2. Alternatively, a node running the Microprocessor Interface Program can handle the response to the poll explicitly.

The code field of the Network Variable Poll request message contains the most significant six bits of the network variable selector. The most significant bit of the code is set, indicating a network variable message, and the second most significant bit 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 eight bits of the network variable selector. The response message contains the same code, except that the second most significant bit is clear, indicating that the response is addressed to an input network variable. The first data byte of the response contains the least significant eight 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.

MC143150•MC143120

APPENDIX C
EXTERNAL MEMORY INTERFACING
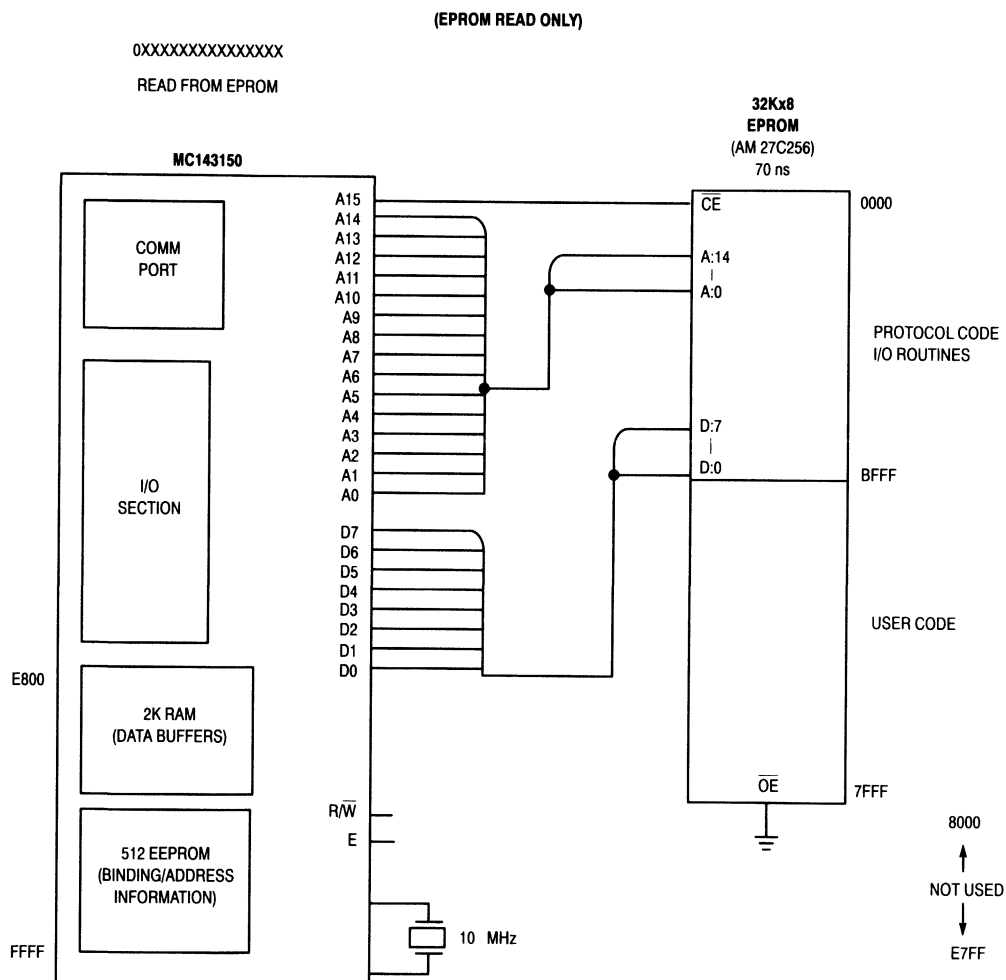
(EPROM READ ONLY)

0XXXXXXXXXXXXXXX

READ FROM EPROM

MC143150•MC143120

MOTOROLA

MC143150/D

◇   1ATX30337-0 PRINTED IN USA 4/92 IMPERIAL LITHO 84994 25,000 MOS D-A YFAAAA