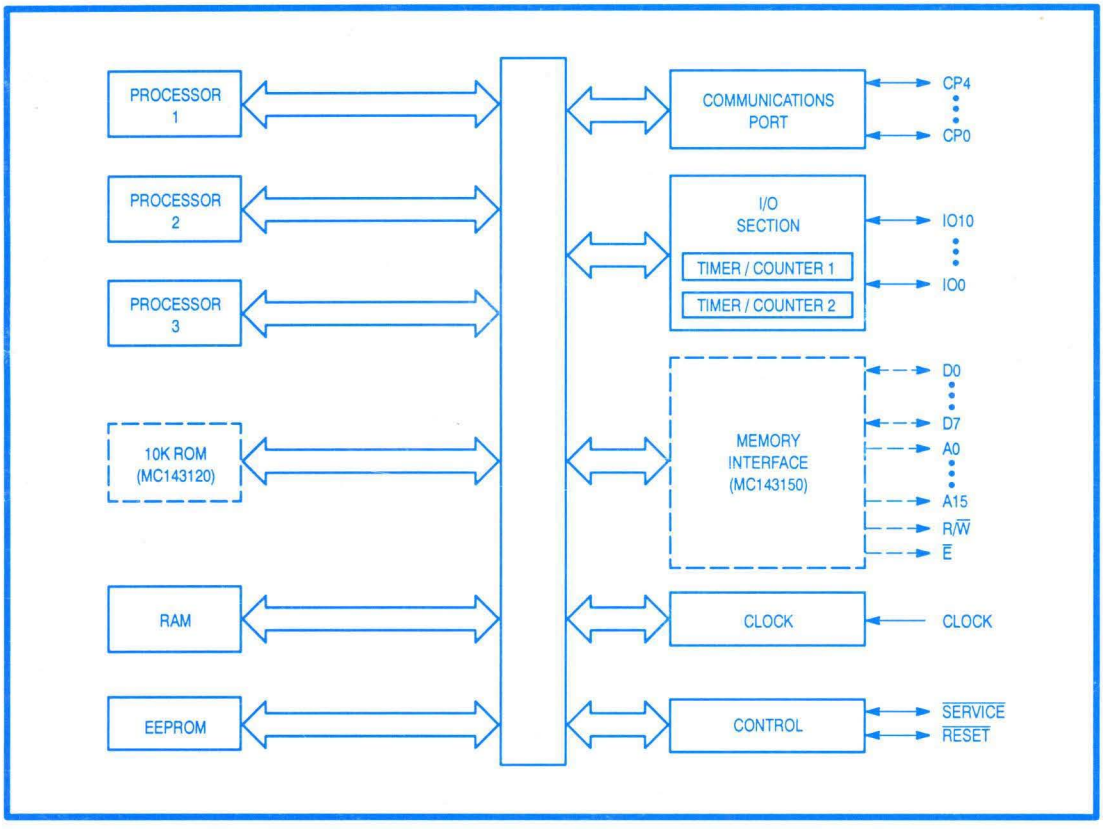
Order this document
by MC143150/D

MOTOROLA
SEMICONDUCTOR
TECHNICAL DATA

MC143150
MC143120

Advance Information

NEURON® CHIP Distributed Communications and Control Processors



MOTOROLA

REV 2

This document contains information on a new product. Specifications and information herein are subject to change without notice.

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.

CONTENTS

CONTENTS (Continued)

CONTENTS (Continued)

CONTENTS (Concluded)

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF FIGURES (Concluded)

LIST OF TABLES

SECTION 1
INTRODUCTION

The Motorola MC143150FU, MC143150FU1, and MC143120DW 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

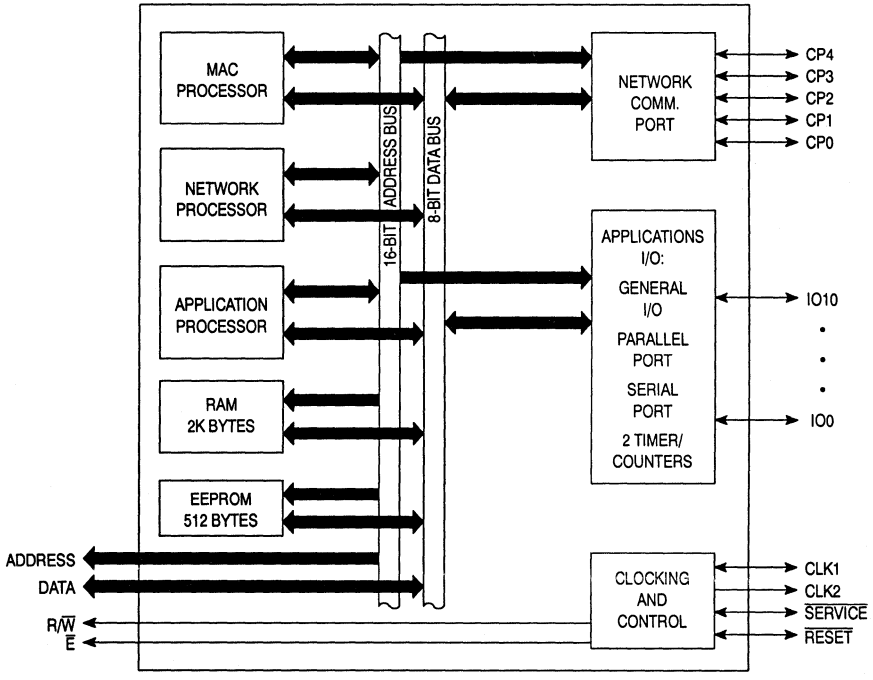


Figure 1-1. MC143150 NEURON CHIP Simplified Block Diagram

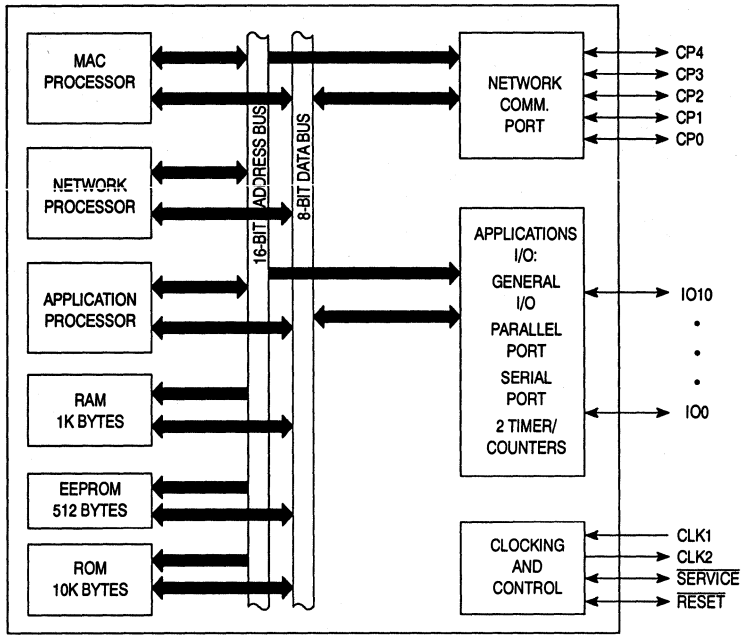


Figure 1-2. MC143120 NEURON CHIP Simplified Block Diagram

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 device. The communications protocol, developer's operating system, and 29 I/O device driver object code are provided with Echelon's LONBUILDER Workbench. The protocol and application object code can be stored in external ROM, EEPROM, NVRAM, or battery-backup static RAM.

The MC143150 NEURON IC is available in two versions (suffix FU and FU1). The FU device can be operated at different voltage ranges to achieve specific performance goals. This will optimize the overall system cost with the associated external memories. The devices are:

Device	Clock Speed Rate	Memory Access Time	Supply Voltage Range	EEPROM Programming Temperature Range	Typical I_{DD}
MC143150FU	10 MHz	90 ns	4.5 – 5.5 V	– 20 to + 85°C	32 mA
MC143150FU	10 MHz	105 ns	4.75 – 5.25 V	– 20 to + 85°C	32 mA
MC143150FU1	5 MHz	200 ns	4.5 – 5.5 V	– 40 to + 85°C	20 mA

As shown, the MC143150FU is designed for input clock operation up to and at 10 MHz clock rate over – 40 to + 85°C with writes to internal EEPROM guaranteed down to – 20°C. The MC143150FU1 is a *lower cost device* designed for 5 MHz operation and below, and operates over the full industrial temperature range of – 40 to + 85°C.

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 accesses the firmware in the masked ROM for performing most of the communications and control functions.

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

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

Features

- Three 8-bit pipelined processors
  - Selectable input clock rates: 625 kHz to 10 MHz
- On-chip memory
  - 2 Kbyte static RAM (MC143150)
  - 1 Kbyte static RAM (MC143120)
  - 512 byte EEPROM
  - 10 Kbyte ROM (MC143120)
- 11 programmable I/O pins
  - 29 selectable modes of operation
  - Programmable pull-ups
  - 20 mA current sink
- Two 16-bit timer/counters for frequency and timer I/O
- Sleep mode for reduced current consumption while retaining operating state
- Network communications port
  - Single-ended mode
  - Differential mode
  - Selectable transmission rates: 0.6 kbit/s to 1.25 Mbit/s
  - 280 packets/s throughput sustained; 700 packets/s peak
  - 40 mA current output for differentially driving twisted-pair networks
  - Optional collision detect input
- Firmware
  - LONTALK protocol conforming to 7-layer OSI reference model
  - I/O drivers
  - Event-driven task scheduler
- Service pin for remote identification and diagnostics
- Unique 48-bit internal NEURON ID

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.

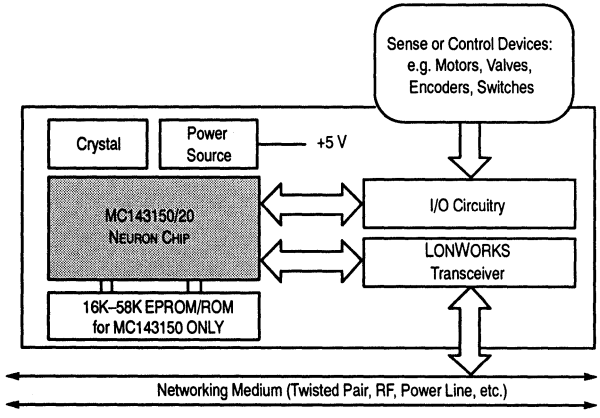


Figure 2-1. Typical Node Block Diagram

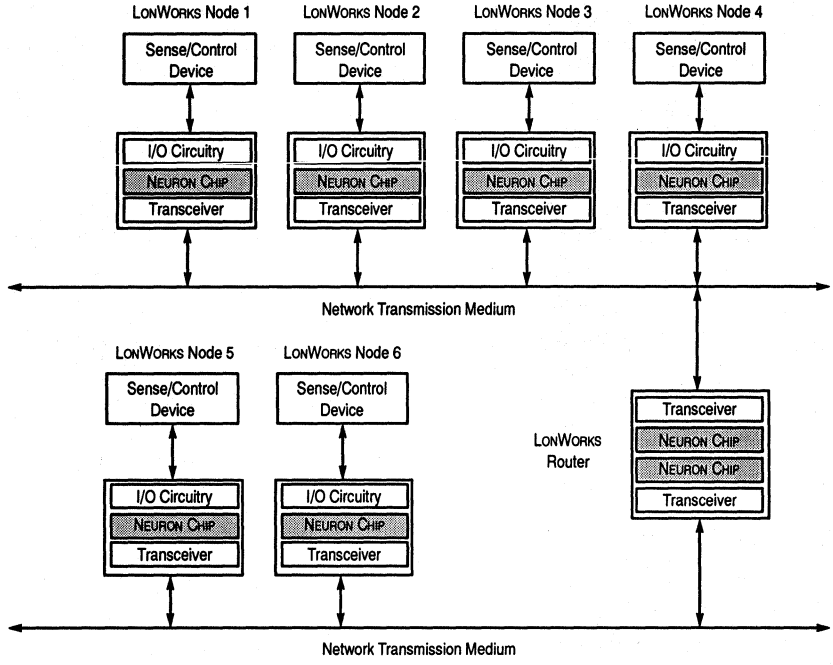


Figure 2-2. The MC143150 or MC143120 in a LONWORKS Network

The NEURON IC and LONTALK protocol are licensed products of Echelon Corporation. The following discussions are meant to give customers an awareness of the various restrictions and license fees that are associated with using LONWORKS technology. Contact either Motorola or Echelon for a copy of the complete license agreement. The key topics covered are:

A) Development License Agreement — no cost.
B) OEM License Agreement (Rev. F) — Contact Echelon for price.
C) Special Purpose Mode protocol (restrictions) — Generally no cost.
D) Router License Agreement — Various cost options.
E) Various Software License Agreements.

A) Development License Agreement

To get started, a customer needs a Development License Agreement. This enables them to receive NEURON CHIP samples from Motorola to be used in their product development.

There is no time limit or price associated with this license. It simply informs the customer of the various restrictions associated with LONWORKS technology. The following is a sample of one of the paragraphs in the Development License Agreement.

Development License Agreement (Rev. F)

Entitles the user to a nonexclusive license, under Echelon intellectual property, solely to use NEURON CHIPS to develop and design LONWORKS Applications.

Licensee's rights to use the LONTALK Protocol and NEURON CHIPS shall not extend to use of the LONTALK Protocol in devices that duplicate functions of all or part of the NEURON CHIPS, or to use of the NEURON CHIPS with any communications protocol other than the LONTALK Protocol. The foregoing limitation shall apply to all NEURON CHIPS used by Licensee, including NEURON CHIPS contained in products or equipment purchased by Licensee.

Contact either Motorola or Echelon for a copy of the complete license agreement if you are interested in receiving samples from Motorola.

B) OEM License Agreement

When the design is complete and ready for production, an OEM License Agreement will need to be signed to enable the customer to purchase volume quantities of NEURON ICs from Motorola. The cost is payable to Echelon Corporation (consult Echelon for pricing). The following is a sample of one of the paragraphs in the Development License Agreement.

OEM License Agreement (Rev. F)

Echelon grants Licensee a nonexclusive license, under Echelon Intellectual Property, to make, use, and sell LONWORKS Applications. Licensee agrees that whenever a NEURON CHIP is executing instructions, the NEURON CHIP Firmware shall be loaded into it starting at address location 0 (zero). Licensee's rights to use the LONTALK Protocol and NEURON CHIP shall not extend to use of the LONTALK Protocol in devices that duplicate the functions of all or part of the NEURON CHIPS, or to use the NEURON CHIP with any communications protocol other than the LONTALK Protocol. The foregoing limitations shall apply to all NEURON CHIPS used by Licensee, including NEURON CHIPS contained in products or equipment purchased by Licensee.

Contact either Motorola or Echelon for a copy of the complete license agreement.

C) Special Purpose Mode Protocol Restriction

The NEURON IC has three different communication modes for building transceivers. One specific mode, called Special Purpose Mode, uses a unique handshake protocol for communications to an external transceiver, such as a powerline or RF. There are restrictions associated with this mode of operation that limit the types of transceivers that can be interfaced to the NEURON IC without written approval from Echelon. The following is a sample of one of the paragraphs in the Development License Agreement.

Special Purpose Protocol Restriction

Unless otherwise approved in writing by Echelon, such transceivers must be of licensee's own proprietary design, and LONWORKS applications containing such transceivers must include licensee's proprietary software that will perform the end use functions in addition to network connections, for which the particular LONWORKS Application was designed. Contact either Motorola or Echelon for a copy of the complete license agreement.

D) Router License Agreement

For LONWORKS systems needing routers, bridges or gateways, customers can purchase complete or board based products from Echelon with shrink wrapped license but no license cost. For customers needing to develop their own routers, they can purchase the Source Code from Echelon and pay per run time license fees. Contact either Motorola or Echelon for a copy of the complete license agreement.

E) Various Software License Agreements

Echelon sells various software packages to help reduce customer's development times. There are various restrictions, along with upfront and run time cost. Contact Echelon for more information on product offering and cost.

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

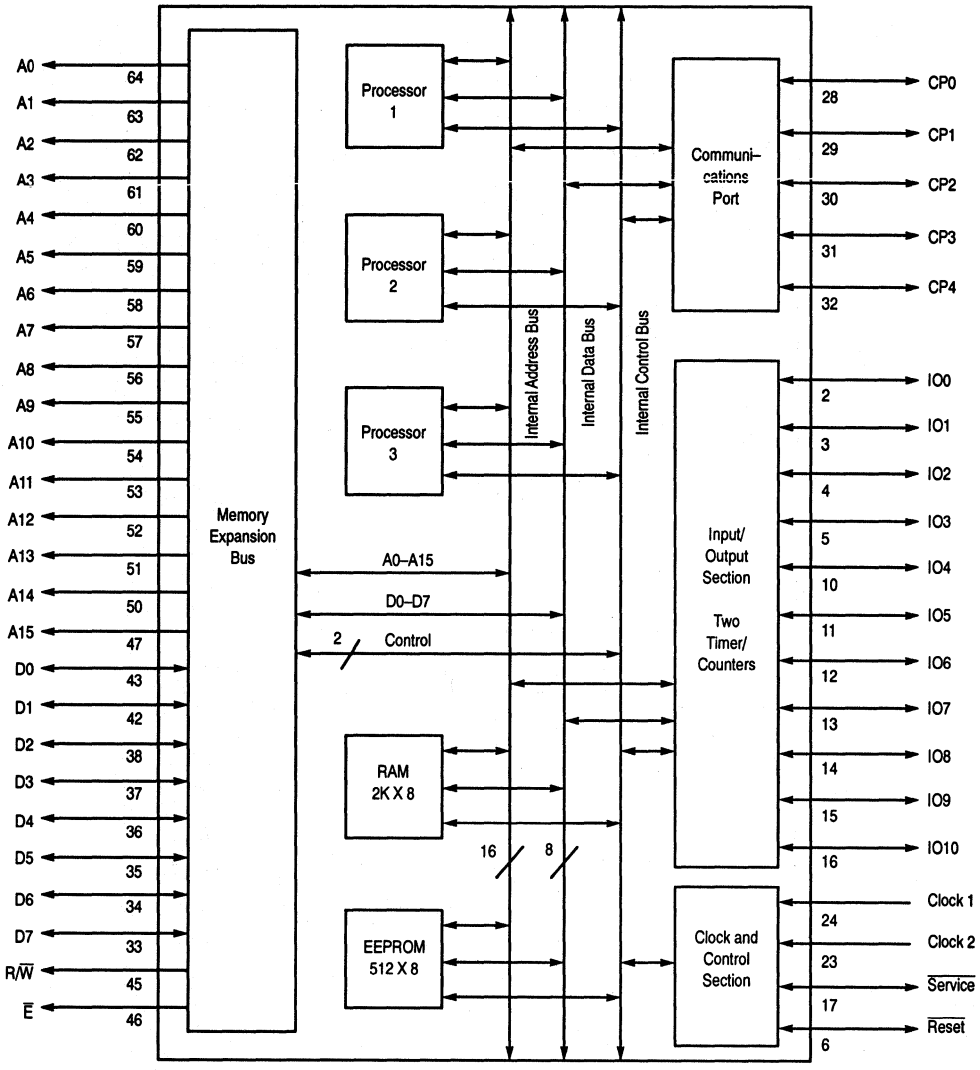
Characteristic	MC143150	MC143120
Processors	3	3
RAM Bytes	2,048	1,024
ROM Bytes	—	10,240
EEPROM Bytes	512	512
16-Bit Timer/Counters	2	2
External Memory Interface	Yes	No
Package	PQFP	SOG
Pins	64	32

3.1 PROCESSING UNITS

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

Processor 1 is the Media Access Control (MAC) layer processor that handles layers one and two 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 network buffers located in shared memory.

Processor 2 is the Network Processor that implements layers three 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 network buffers in shared memory to communicate with Processor 1, and Application Buffers to communicate with Processor 3. The buffers are also lcoated in shared memory. Access to them is mediated with hardware semaphores to resolve contention when updating shared data.



NOTE: Pin numbers apply to 64-lead package.

Figure 3-1. MC143150

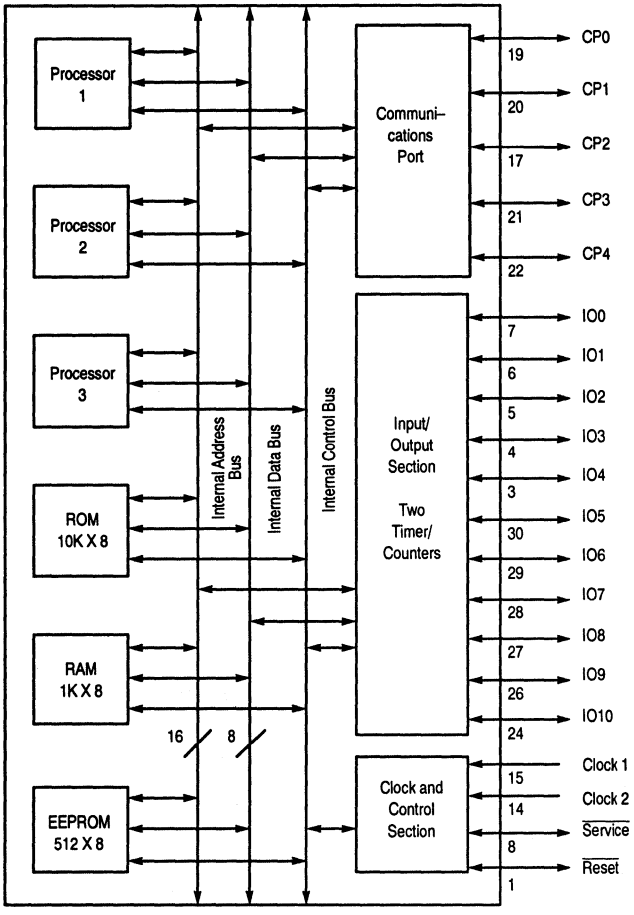


Figure 3-2. MC143120

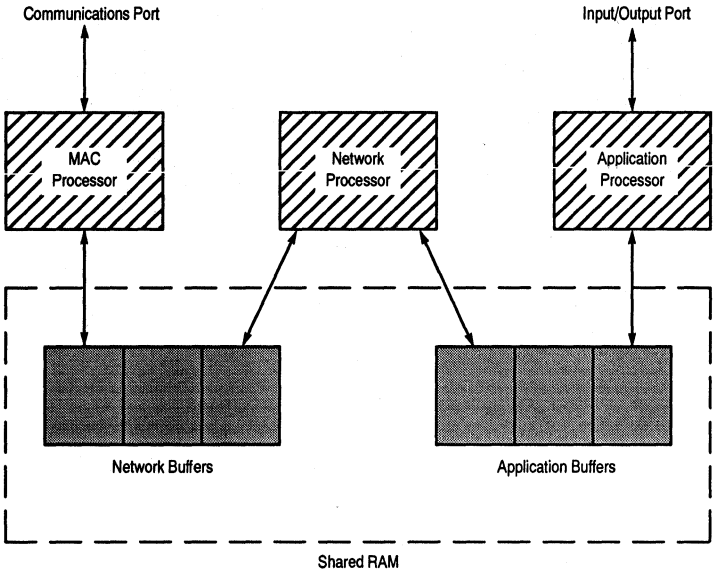


Figure 3-3. Processor Organization Memory Allocation

Processor 3 is the Application Processor. It executes the code written by the user, together with the operating system services called by user code. The primary programming language used by most applications is NEURON C, a derivative of the 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, manage input/output activities, 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.

Each of the three identical processors has its own register set (Table 3-2), but all three processors share data and address ALUs and memory access circuitry (Figure 3-4). Each CPU *minor* cycle consists of *three system clock* cycles, or phases; each system clock cycle is two input clock cycles. The minor cycles of the three processors are offset from one another by one system clock cycle, so that each processor can access memory and ALUs once during each instrument cycle. Figure 3-4 shows the active elements for each processor during one of the three phases of a minor cycle. The system thus pipelines the three processors, reducing hardware requirements without affecting performance. This allows the execution of three processes in parallel without time-consuming interrupts and context switching.

Table 3-2. Register Set

Mnemonic	Bits	Contents
FLAGS	8	CPU number, fast I/O select, and carry bit
IP	16	Next Instruction Pointer
BP	16	Address of 256-byte Base Page
DSP	8	Data Stack Pointer within base page
RSP	8	Return Stack Pointer within base page
TOS	8	Top Of Data Stack, ALU input

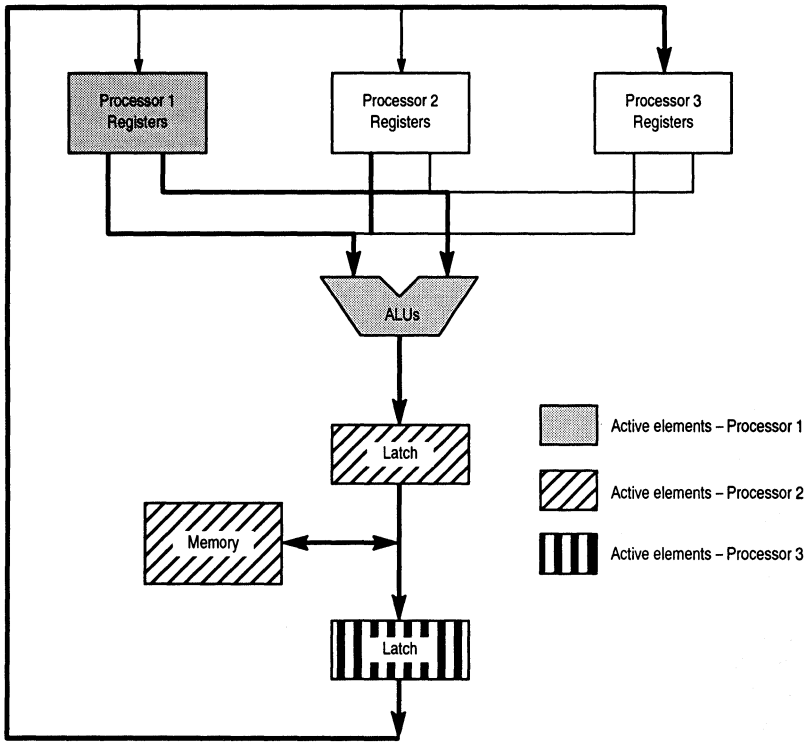


Figure 3-4. Processor/Memory Activity During One of the Three Phases of a Minor Cycle

The architecture is stack-oriented; one 8-bit wide stack is used for data references, and the ALU operates on the TOS (Top Of Stack) register and the next entry in the data stack which is in RAM. A second stack stores the return addresses for CALL instructions, and may also be used for temporary data storage. This zero-address architecture leads to very compact code. Tables 3-15, 3-16, and 3-17 outline the instruction set.

Figure 3-5 shows the layout of a base page, which may be up to 256 bytes long. Each of the three processors uses a different base page, whose address is given by the contents of the BP register of that processor. The top of the data stack is in the 8-bit TOS register, and the next element in the data stack is at the location within the base page at the offset given by the contents of the DSP register. The data stack grows from low memory towards high memory, and the return stack grows from high memory towards low memory.

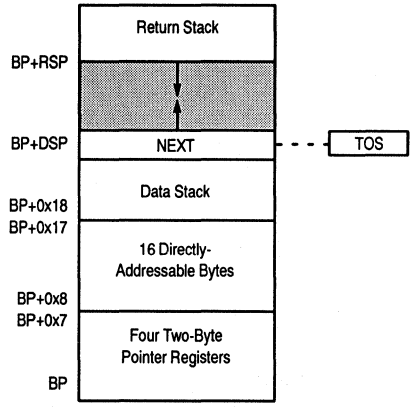


Figure 3-5. Base Page Memory Layout

3.2 MEMORY

3.2.1 Memory Allocation Overview

3.2.1.1 MC143150 MEMORY ALLOCATION (SEE FIGURE 3-6).

- 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

3.2.1.2 MC143120 MEMORY ALLOCATION (SEE FIGURE 3-7).

- 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

3.2.2 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 4.6.3). The EEPROM of the MC143150 can usually be corrected in the event of a fault by executing the program EE-Blank, which is supplied with the LONBUILDER Developer's Workbench.

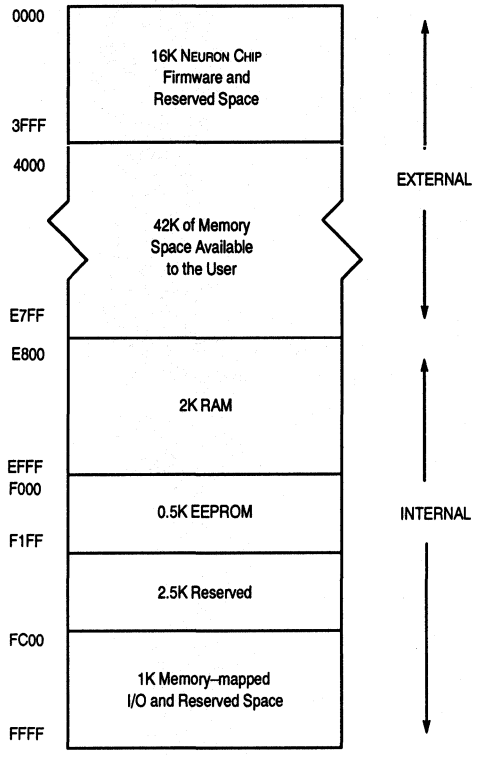


Figure 3-6. MC143150 Processor Memory Map

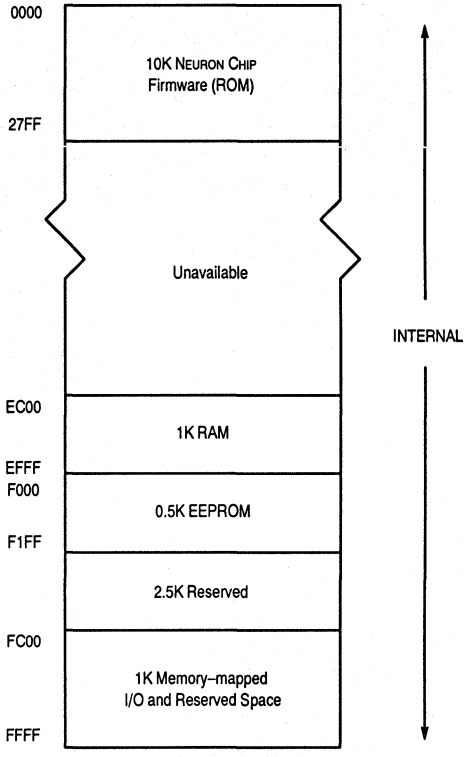


Figure 3-7. MC143120 Processor Memory Map

3.2.3 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. Reset will clear the RAM.

3.2.4 Preprogrammed ROM

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

3.2.5 External Memory (MC143150 Only)

This interface supports up to 42K bytes of external memory space for additional user program and data. The total address space is 64K bytes. However, 6K of address space is reserved for on-chip, leaving 58K of external address space. Of this space, 16K is used by the NEURON CHIP firmware, LONBUILDER development debugger, and reserved space. The external memory space can be populated with RAM, ROM, PROM, EPROM, or EEPROM in 256 byte increments. The memory maps are shown in Figure 3-6 and Figure 3-7. The bus has eight bidirectional data lines, and sixteen address lines driven by the processor. Two interface lines (R/$\overline{W}$ and $\overline{E}$) are used for external memory access (see Figure 3-1). At the maximum

clock rate (10 MHz input clock), 90 ns or better access time to external memory is required for the MC143150FU. If the input clock is scaled down, slower memory can be used. The suggested input clock rates are 10 MHz, 5 MHz, 2.5 MHz, 1.25 MHz, and 625 kHz. The Enable Clock ($\overline{E}$) runs at the system clock rate, which is one half the input clock rate. The Enable Clock is low whenever data is being transferred between the NEURON CHIP and external memory. All memory, both internal and external, may be accessed by any of the three processors at the appropriate phase of the instruction cycle. Since the instruction cycles of the three processors are offset by one third of a cycle with respect to each other, the memory bus is in use by only one processor at a time.

Appendix C 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 3-8). 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 $\overline{E}$ clock. The $\overline{E}$ clock signal is not typically used for *reading* external memory but can be gated with decoded address lines and the R/$\overline{W}$ signal to generate $\overline{WRITE}$ signals to external memory. A15 (address line 15) or a PAL decoded signal gated with R/$\overline{W}$ can be used to generate read signals to external memory. The MC143150FU needs to see data (setup time) 55 ns ahead of the rising edge of $\overline{E}$ clock. The data hold time required when reading data is 0 ns.

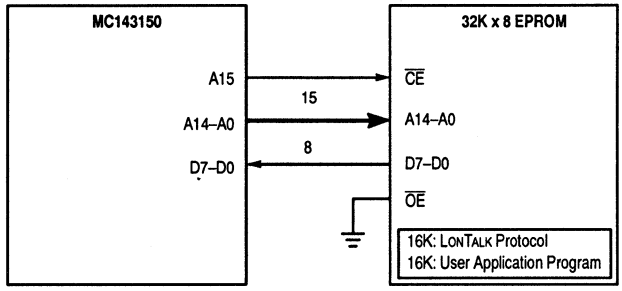


Figure 3-8. Minimum MC143150 Memory Interface

Table 3-3. External Memory Interface Pins

Pin Designation	Direction	Function
A0–A15	Output	Address pins
D0–D7	Input/Output	Data pins
$\overline{E}$	Output	Enable clock
R/$\overline{W}$	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. Memory mapped I/O is supported in the LONBUILDER environment.

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	7	—	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	55	—	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	—	60	ns
t_{WH}	R/$\overline{W}$ Hold Time Write	5	—	ns

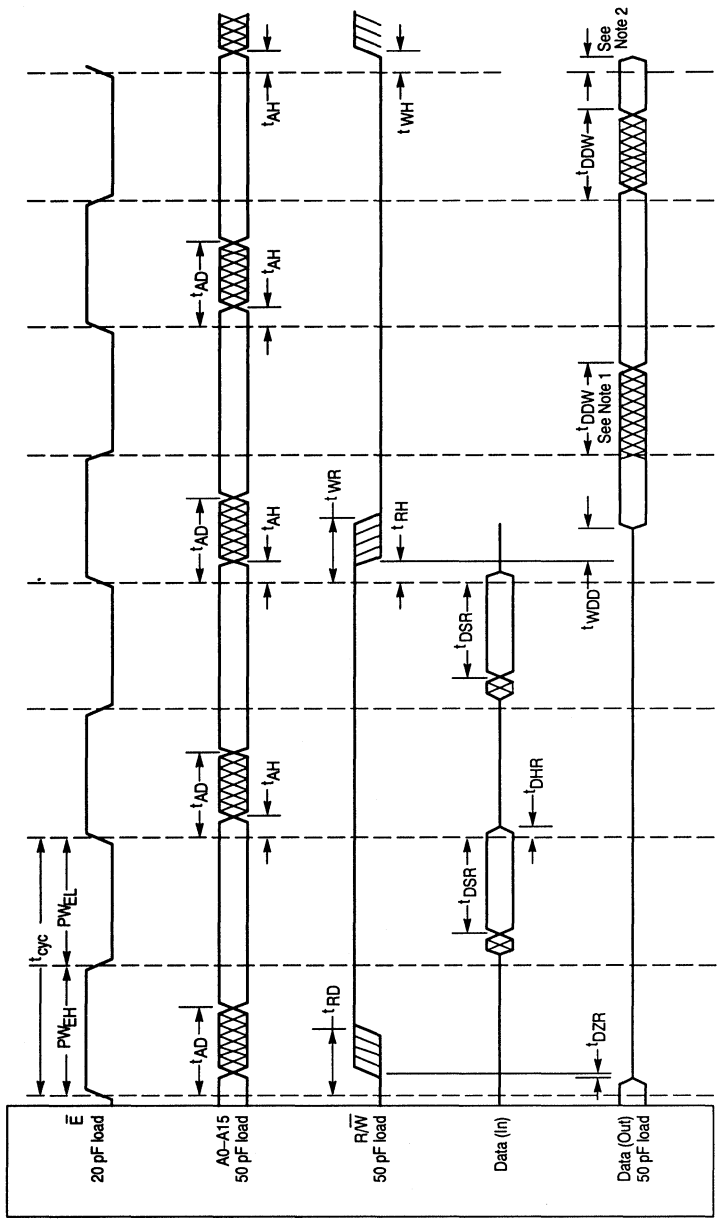
NOTES:
* Table 3-4 is for the MC143150FU. See Section 6 for FU1 timing.
1. $t_{cyc} = 2 \cdot 1/f$, where f is the input clock frequency.
2. See Figure 3-9. The NEURON CHIP drives the previously read data until after the falling edge of $\overline{E}$. Therefore, an external memory and the NEURON CHIP 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-up current sources. 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.



NOTES:
1. 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.
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.

Figure 3-9. External Memory Interface Timing Diagram

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 3-10). 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.

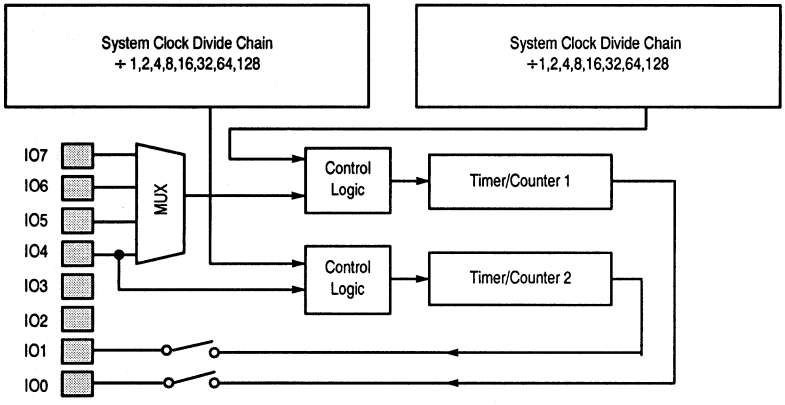


Figure 3-10. Timer/Counter Circuits

SECTION 4
NETWORK COMMUNICATIONS

4.1 INTRODUCTION

There are three published standard twisted pair interfaces for the NEURON IC.

A. The first is the "direct drive" interface using the NEURON device's internal transceiver with external resistors and diodes for current limiting and ESD protection. This is ideal for networks of nodes with a common power source that would be implemented **inside** many types of equipment. Varying data rates can be easily supported.

B. The second is for transformer coupled interfaces, which also use the IC's internal transceiver, for use in applications that require higher performance than direct drive can support. Optimized for specific data rates, the transformers require additional passive components such as inductors, capacitors, resistors, and isolation relays to support multiple power sourced networks. Changing data rates requires altering component values. U.L. level IV wiring is recommended for good performance with the specified 78 kbps and 1.25 Mbps interfaces.

C. A third standard interface uses commercially available EIA-485 transceivers to allow for cost/performance capability in between direct drive and transformer coupled interfaces. Multiple data rates and wire types can be supported with no changes in component values at lower loop distances than transformer coupled interfaces. Common mode voltage range is also reduced unless additional components (and cost) such as optoisolators are added.

One key consideration when implementing twisted pair transceivers is compatibility or interoperability with other LONWORKS based products. A developer has three choices. One is to adopt and purchase Echelon's OEM transceivers and transceiver modules. This ensures total compatibility or interoperability to other vendors' products, and using Echelon's LONMARK stamp gives visible indication of interoperability to other customers' LONWORKS products.

A second option is for a group of companies to jointly establish their particular NEURON CHIP twisted pair interface for a given industry/market, deciding on their own data rate and transceiver design. Obviously the more the proliferation of different designs within or across industry segments, the less the "standard" serves its purpose. The consortium would have to publish a set of specifications and institute a method to test and monitor for product conformance. Agreement among different companies with varying interests is many times difficult or impossible to achieve.

A third choice is for a company/developer to implement a unique transceiver design, not concerned with compatibility or interoperability. This would obviate the ability to use standard installation tools, routers or other products that may be available for the standard transceiver interfaced products. It may also limit potential product growth for lack of interoperability.

For direct drive and EIA-485, the external interface circuit is simple, allowing for a variety of data rate selections without having to adjust external components.

For transformer coupled transceiver applications, the external circuit configuration can be complex, depending on the performance needed. Echelon has developed two transformer designs, one at 78 kbps, the other at 1.25 Mbps. Both designs have been optimized in both component values and board layout to achieve optimum loop distance and loading performance. Echelon offers both stand-alone NEURON CHIP compatible transceivers and modules with power converters and other features such as FCC and U.L. certification. Contact Echelon for additional product information. The diagrams in Figure 4-10 illustrate the component values for 78 kbps and 1.25 Mbps. Performance of these circuits is very much layout dependent. For achieving other data rates, developers will need to modify the transformer and discrete component design to achieve good performance (long distances, number of nodes, low bit error rates, etc.).

The decision to make or buy transceivers must be made by the developer/OEM, and must take into consideration all of the above in addition to time to market product requirements, design resources available, what is the value added and the total cost (parts, design labor, manufacturing, compliance testing, etc.) of both alternatives.

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 three modes: single-ended, differential, or special purpose mode. See Table 4-1 and Figure 4-1.

Table 4-1. Communications Port Pin Characteristics

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

RX—Receiver TX—Transmitter CDet—Collision Detect

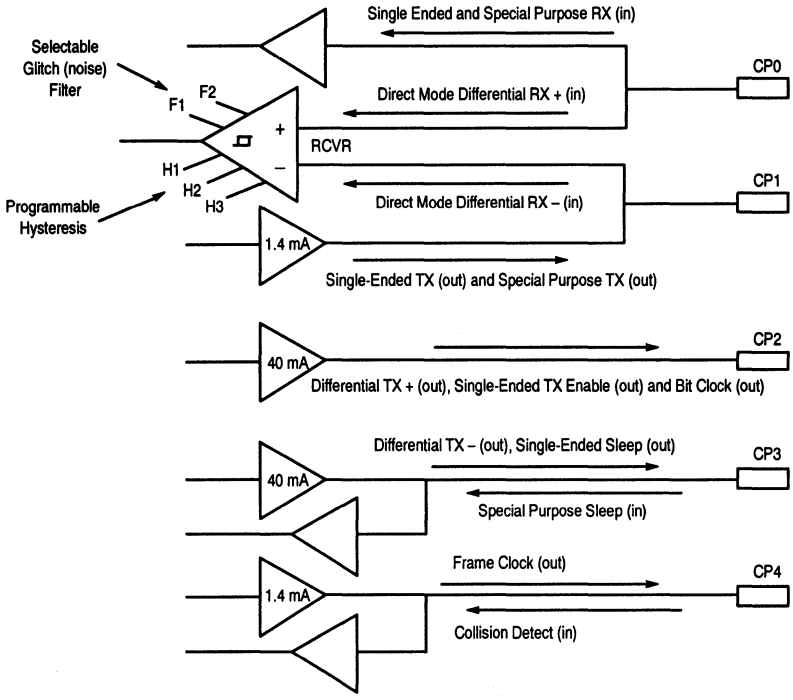


Figure 4-1. Internal Transceiver Block Diagram

In single-ended and differential modes, the processor communicates with the transceiver using a stream of Differential Manchester encoded data (see Figure 4-2). 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.

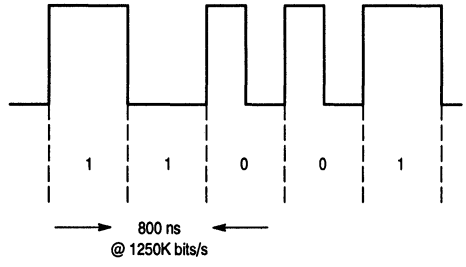


Figure 4-2. Differential Manchester Data Encoding

4.2 SINGLE-ENDED MODE

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

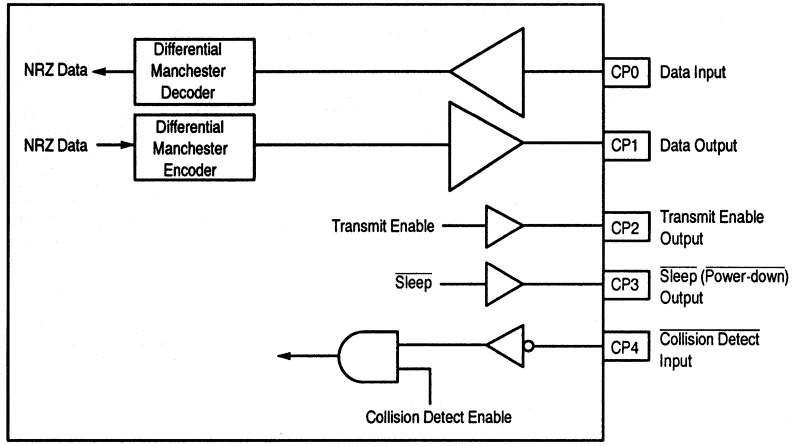


Figure 4-3. Single-Ended Mode Configuration

Table 4-2. 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
2.4	0.625	2.5
1.2	0.625	1.25
0.6	0.625	0.625

In single-ended mode, the communications port encodes transmitted data and decodes received data using Differential Manchester coding (also known as bi-phase space coding). This scheme provides a transition at the beginning of every bit period for the purpose of synchronizing the receiver clock. 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 4-4 shows a typical packet where T is the bit period, equal to 1/(bit rate). Note that clock transitions occur at the beginning of a bit period, and therefore, the last valid bit in the packet does not have a trailing clock edge.

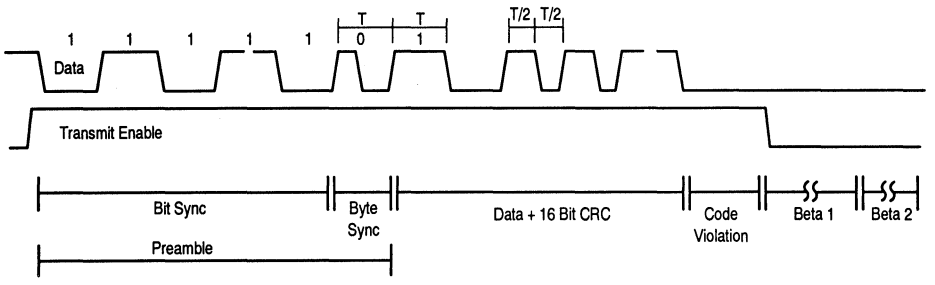


Figure 4-4. Single-Ended Mode Data Format

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

The transmitter transmits a *preamble* at the beginning of a packet to allow the other nodes to synchronize their receiver clocks. The preamble consists of a series of Differential Manchester 1's. Its duration is at least six bits long and is selectable by the user.

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

The NEURON CHIP terminates the packet by forcing a Differential Manchester code violation. After sending the last CRC bit, it holds the data output transitionless for 2-1/2 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. The collision detect flag is checked by firmware at the end of the preamble and end of packet.

If the node does not use collision detection, then the only way it can determine that a message has not been received is to request an 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.

The idle period between packets comprises the *Beta* 1 time and *Beta* 2 slots. The *Beta* 1 time is the fixed component in the idle period after a packet has been sent. This component is a function of the following:

- Oscillator frequencies and accuracies on the various network nodes
- Media indeterminate time — that time following packet transmission when the network can appear to be busy due to ringing on the line

- Minimum interpacket gap — transceiver dependent timing requirements
- Receive end delay — skew between transmitter and receiver NEURON CHIP's view of the end of the packet. This would be due to buffering in the transceivers and is typically found in special purpose mode transceivers.

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

Both priority (P) and non-priority slots are defined by the Beta 2 time. Nodes listen to the network prior to transmitting a packet. This prevents nodes from transmitting packets on top of each other except when the packets are initiated at nearly the same time. In addition, nodes randomize the time before they start transmitting on the network. When the network is idle, all nodes randomize over 16 slots. When the estimated network load increases, nodes start randomizing over more slots in order to lower the probability of a collision. The number of randomizing slots (R) varies from 16 up to 1008, based on "n", the estimated channel backlog (the number of slots is n•16 where "n" has a range of 1 to 63). See Figure 4-5.

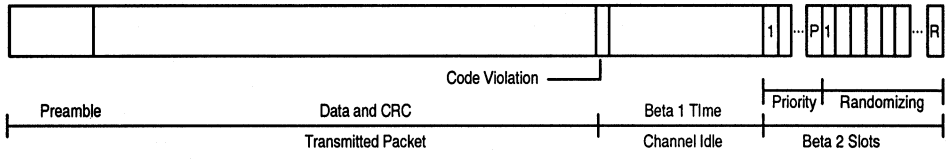


Figure 4-5. Packet Timing

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

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

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

When put in single-ended mode the NEURON CHIP can interface to an external EIA-485 transceiver IC. See Figure 4-6. While not having the performance of a transformer coupled circuit (Section 4.4) 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 unit loads (depends on EIA-485 transceiver IC, typically at least 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.

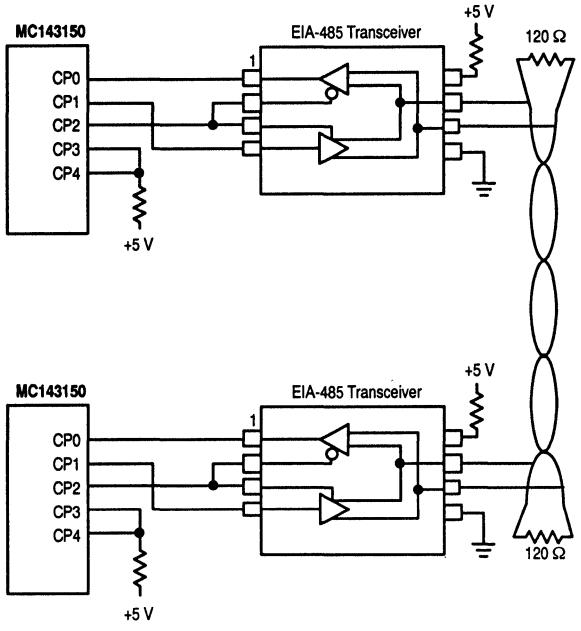


Figure 4-6. EIA-485 Twisted Pair Interface (Used with single-ended mode)

4.3 DIFFERENTIAL MODE

In the differential mode, the NEURON CHIP's built-in transceiver is able to differentially drive and sense a twisted-pair transmission line with external passive components. Differential mode is similar in most respects to single-ended mode; the key difference is that the driver/receiver circuitry is configured for differential line transmission. Data output pins CP2 and CP3 are driven to opposite states during transmission and put in a high-impedance (undriven) state when not transmitting. The differential receiver circuitry on pins CP0 and CP1 has selectable hysteresis with eight selectable voltage levels followed by a selectable low-pass filter with four selectable values of transient pulse (noise) suppression. See Figure 4-7. The selectable hysteresis and filter permit optimizing receiver performance to line conditions. See Tables 4-3 and 4-4 for specific values.

Figure 4-8 shows a typical packet waveform in differential mode. Note that the packet format is identical to that of the single-ended interface described earlier; the coding and jitter tolerance also apply identically.

The drivers can source 40 mA of current at 4.0 V and can sink 40 mA of current at 1.0 V (V_{DD} = 5 V). The programmable hysteresis values shown in Table 4-3 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 4-4 and 4-5.

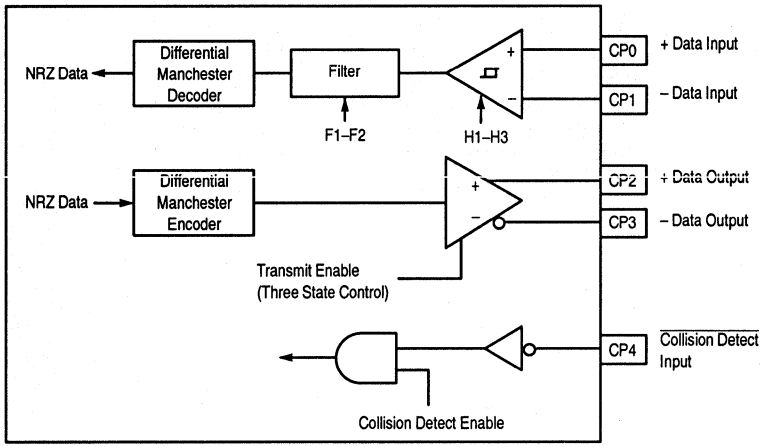


Figure 4-7. Differential Mode

Table 4-3. Communications Port Programmable Hysteresis Values (Expressed as differential peak to peak voltages in terms of V_{DD})

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

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

Table 4-4. 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

*Must be disabled if data rate is 1.25 Mbps.

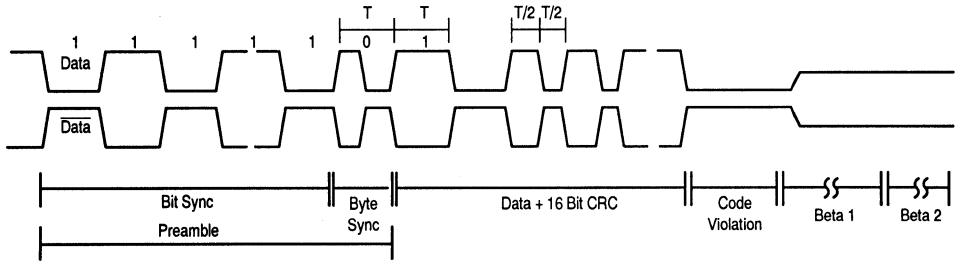


Figure 4-8. Differential Mode Data Format

Table 4-5. Differential Transceiver Electrical Characteristics

Characteristic	Min	Max
Receiver Common Mode Voltage Range to maintain hysteresis as specified in Table 3-7*	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

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

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 4-6 shows the width of this window as a function of the ratio of the NEURON CHIP input clock (MHz) and the network bit rate (Mbps) selected. If a transition falls outside of either window, it is not detected. Timing instability of the transitions, known as jitter, may be caused by changes in the communications medium, or instability in the transmitting or receiving node's input clocks. The jitter tolerance windows are expressed as fractions of the bit period, T, in Table 4-6.

Table 4-6. Receiver Jitter Tolerance Windows

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

The allowable network bit rates for differential mode are given in Table 4-2. All the active components necessary to implement twisted pair networks using differential 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 4-9, 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). The diodes provide some transient (ESD) protection.

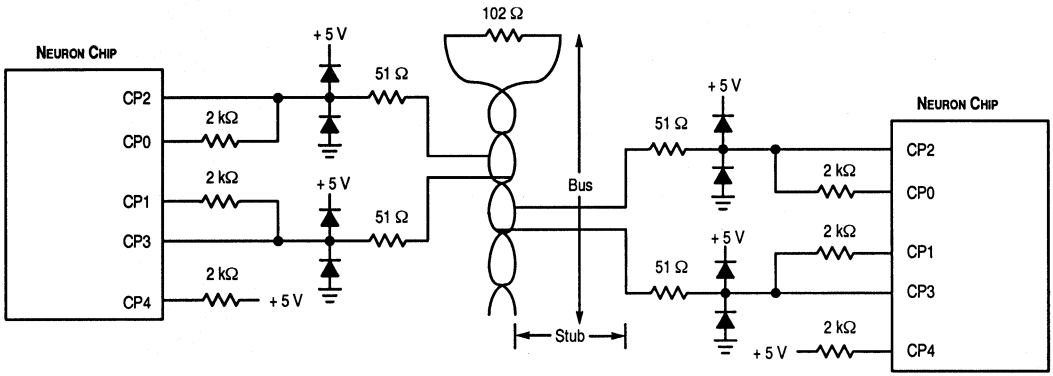


Figure 4-9. Simple Direct Connect Network Interface (Used with direct mode differential)

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. Examples are shown in Figures 4-10a and 4-10b.

When using the transformers described in Table 4-7a, 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. Echelon sells twisted pair transceiver modules using a 78 kbps transformer or a 1.25 Mbps transformer. They are also available in potted (seal) or unpotted versions as referenced in Table 4-7b.

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

	78 kbps	1.25 Mbps
Part Number	0505-0542	PE-65948 Thru Hole PE-65848 Surface Mount
Turns Ratio Lineside Inductance	2:1 = 35 mH	1:1 = 3.5 mH
Company	Precision Components Inc.	Pulse Engineering, Inc.
Address	400 W. Davy Lane Wilmington, Illinois 60481	7250 Convoy Court San Diego, CA 92111
Phone Number	708-543-6448	619-674-8100
Contact	Bill Gray	John G. Schuler

Table 4-7b. Echelon Twisted-Pair Transceiver Modules

Product	Echelon Model Number
TPT/XF-78 Module (potted)	50010
TPT/XF-78 Module (unpotted)	50010-10
TPT/XF-250 Module (potted)	50020
TPT/XF-250 Module (unpotted)	50020-10

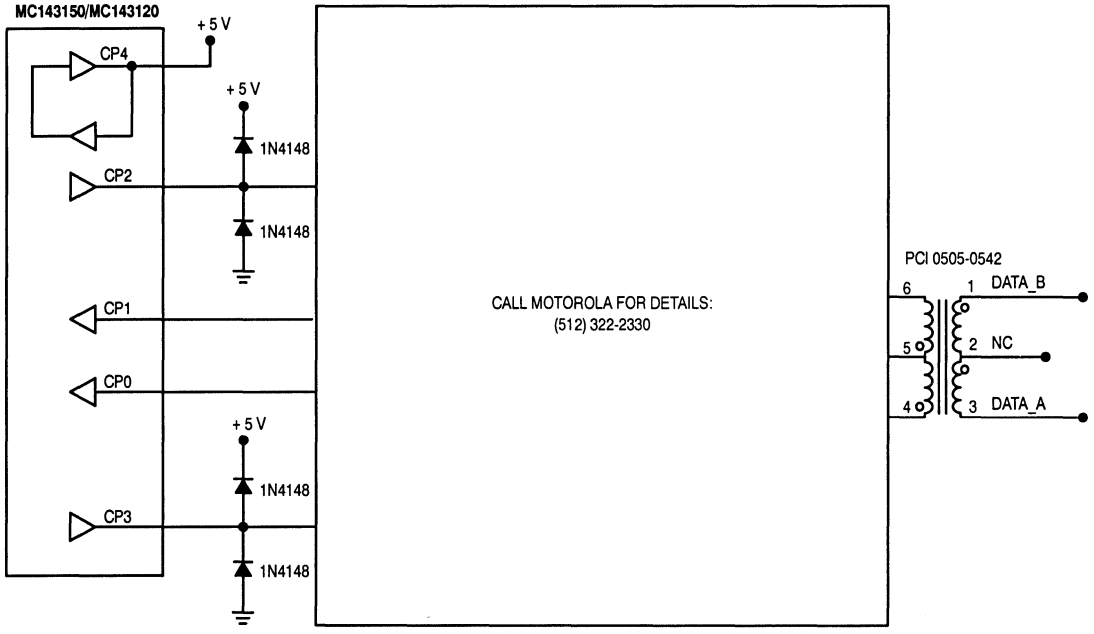


Figure 4-10a. Transformer Isolated Twisted Pair Interface (78 kbps)

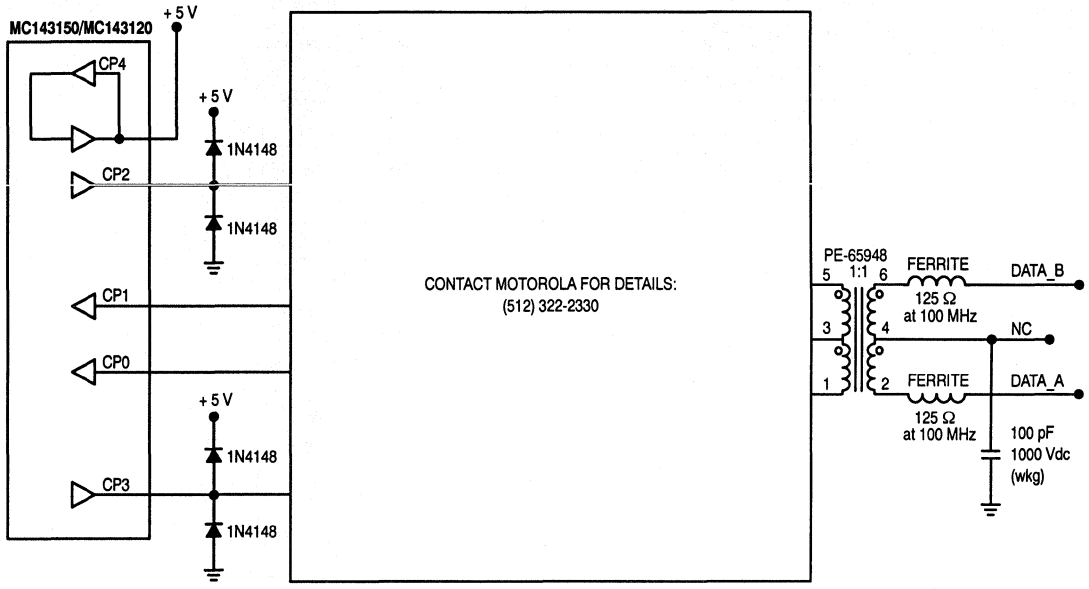


Figure 4-10b. Transformer Isolated Twisted Pair Interface (1.25 Mbps)

4.4 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

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

When the special-purpose mode is used, a protocol is utilized between the NEURON CHIP and the transceiver. This protocol consists of the NEURON CHIP and the transceiver, each exchanging 16 bits, 8 bits of status and 8 bits of data (see Figure 4-11) simultaneously and continuously at rates up to 1.25 Mbps (when the NEURON CHIP's input clock is 10 MHz). The 1.25 Mbps bit rate allows time-critical flags, such as a Carrier Detect, to be exchanged across the interface with network bit rates up to 156 kbps. The maximum bit rate is 156 kbps due to the overhead associated with the handshaking.

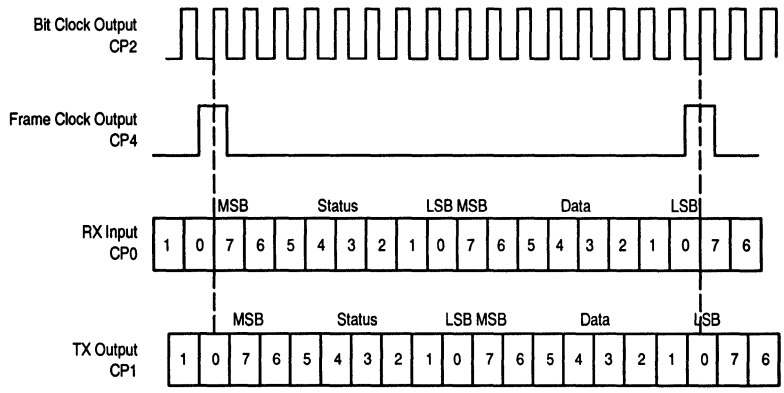


Figure 4-11. Special-Purpose Mode Data Format

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

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

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

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

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

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

The first 8 bits of each frame are interpreted as the status field, and the last 8 bits are the data field. The status field is used to control transceiver operation and to control passing data between the NEURON CHIP and the transceiver. The interpretation of each status bit is shown in Table 4-8.

Table 4-8. 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.

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

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

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

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

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

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

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

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

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

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

The NEURON CHIP performs a configuration write or status read by using TX_ADDR_R/W and TX_ADDR_[2:0]. TX_ADDR_[2:0] indicates which of 7 transceiver registers is being accessed, and TX_ADDR_R/W indicates whether the operation is a configuration register write (0) or status register read (1). Register 0 (TX_ADDR_[2:0] = 000) is unused, so that TX_ADDR_R/W = 0 and TX_ADDR_[2:0] = 000 indicates no read or write operation is to be performed.

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

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

4.5 CLOCKING SYSTEM

4.5.1 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 4-12 and Table 4-9. A 60/40 duty cycle is required when using an external oscillator.

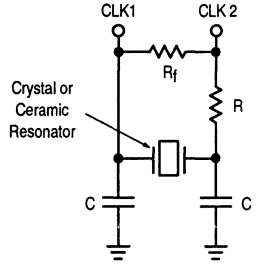


Figure 4-12. NEURON CHIP Clock Generator Circuit

Table 4-9. 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

NOTES:

- The capacitor values include stray capacitances; crystal or ceramic resonator manufacturers may recommend other values.
- R_f = 100 kΩ. R_f is required with ceramic resonator configurations and is not required but may be used with crystal configurations.
- Crystal or ceramic 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 source 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	1.25 MHz	CSB 1250I
	0.625 MHz	CSB 625J

When deciding what speed to operate the NEURON CHIP, the cost of the external memory will be an important consideration. There is a significant cost difference between a 200 ns memory (EPROM access time required at 5 MHz for the FU1 device) and a 90 ns memory (10 MHz access time required for the FU device) which is rated over the full industrial temperature range particularly in surface mount packages.

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

I/O response time will also double when operating at 5 MHz, and this must be considered. Proper prioritizing of "when" statements in the application program and use of timer counter objects can help to minimize latency. For nodes that require extremely fast I/O response time, if only a small number are needed in a network, using a 68HC11 or 68HC05 with their hardware interrupt capability and interfacing to the NEURON CHIP in one of the parallel modes is an option. The following electrical tables highlight the differences between the MC143150FU and FU1. Other characteristics which differ are noted.

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

Table 4-10. Typical Start-Up Times

Input Clock Frequency	Crystal	Ceramic Resonator
10.0 MHz	1.5 ms	8 μs
5.0 MHz	1.0 ms	15 μs
2.5,MHz	2.0 ms	30 μs
1.25 MHz	3.0 ms	60 μs
0.625 MHz	10 ms	380 μs

4.5.2 Assembly Instruction Set

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:

<Instruction Time> (μs) = <# Cycles> x 6 / <Input Clock> (MHz)

Tables 4-11, 4-12, and 4-13 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 4-11. Program Control Instruction Timings

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

Table 4-12. Memory/Stack 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
IN/OUT	7 + 4n	Transfer n bytes to I/O

Table 4-13. 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
(ADD/AND/OR/XOR) – R	7	Operate with next element on TOS and return

4.6 ADDITIONAL FUNCTIONS

4.6.1 Sleep/Wake-Up Circuitry

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

- I/O pins (maskable):
  One of IO4 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.

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

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

The $\overline{E}$ pin is held inactive (high) during sleep to disable memory operations during this period. All other memory interface signals are indeterminate during sleep.

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

The amount of time required for the NEURON CHIP to perform internal maintenance after waking up before it resumes execution of the application depends on several application parameters and on whether the Network Processor is servicing application timers during this period. The important application parameters are the comm ignore option,the number of receive transactions (if comm ignore is used), and the number of application timers (if the Network Processor services the application timers during this period).

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

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

4.6.2 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.

4.6.3 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 $\overline{\text{RESET}}$ 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 4-13. Figure 4-14 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, including stray capacitance
300 μA = maximum built-in pull-up current
I = maximum pull-up current due to external circuits (< 1.6 mA)
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 $\overline{\text{RESET}}$ pin

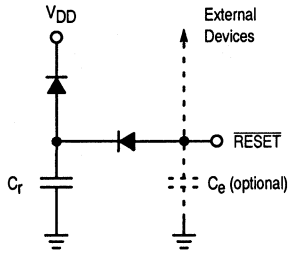


Figure 4-13. Power on Reset Circuit

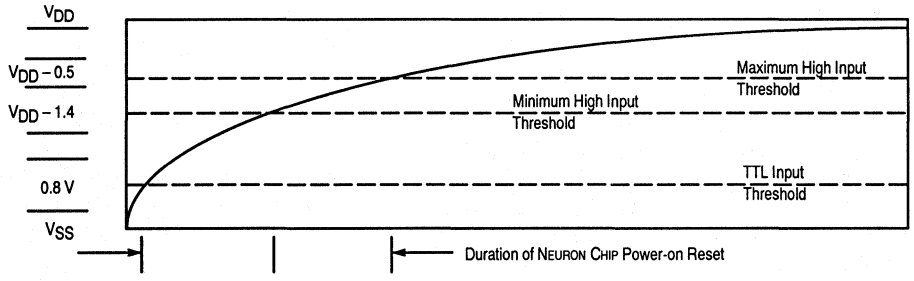


Figure 4-14. Typical Analog Waveform on Reset Pin at Power On

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 $\overline{\text{RESET}}$ pin, including stray and external device input capacitance, may not exceed 250 pF. A NEURON CHIP needs no capacitance on the $\overline{\text{RESET}}$ pin to reset itself or be reset by an external device. These components will provide proper reset for a power supply that will not power down faster than the reset pin voltage will decay, or power up slower than the reset pin will rise. *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 4-15, is recommended.*

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).

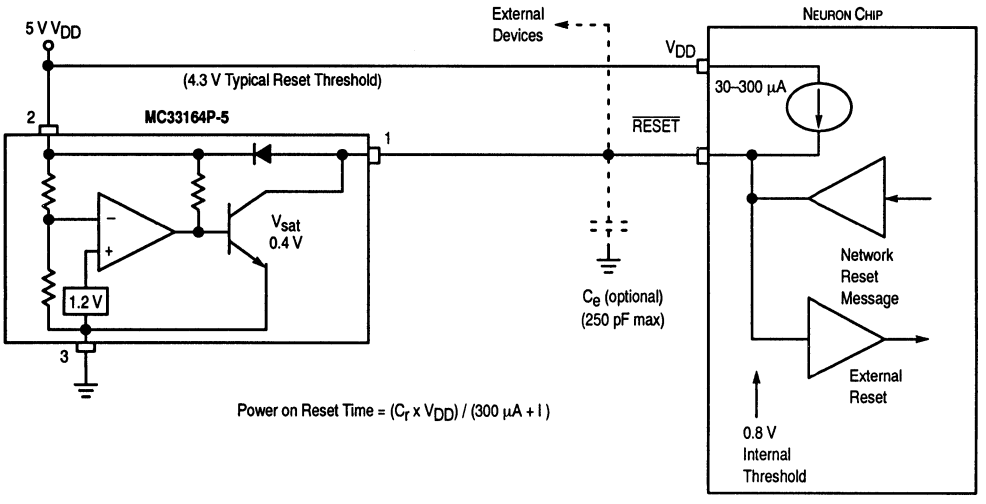


Figure 4-15. Example Low Voltage Detect/Reset Circuit

The state transition timing for NEURON CHIP I/O signals after reset depends on the external memory included in the node implementation. I/O state transitions during reset are bounded by the slowest input clock frequency. Figure 4-16 shows typical I/O state transition behavior for a 5 MHz node during reset.

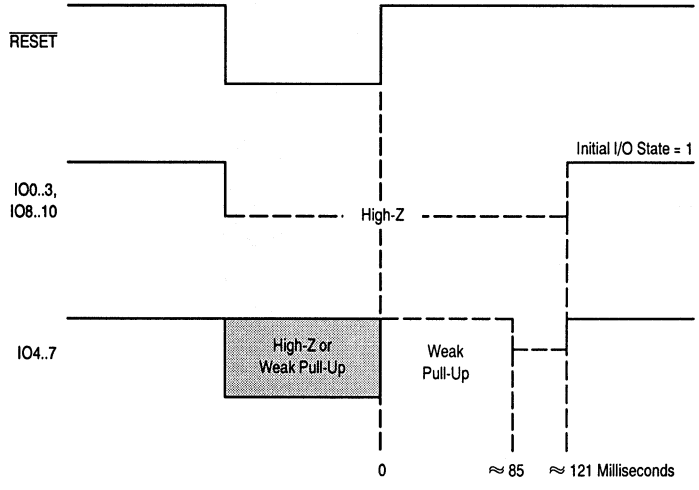


Figure 4-16. Output Pin State Transitions for 5 MHz Node

When a processor watchdog timer expires, or a software command to reset occurs, the output N-channel transistor turns on and pulls down the $\overline{\text{RESET}}$ 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 $\overline{\text{RESET}}$ pin is low, it sends a signal through the control logic to shut off the N-channel transistor. The $\overline{\text{RESET}}$ pin and optional capacitor (<250 pF) are allowed to begin their slow rise time, and provide the required duration of reset.

4.6.4 Reset Processes and Timing

During the reset period, the I/O pins, communication port pins,and service pin are in a high impedance state. The memory interface signals (address, data, $\overline{E}$, and R/$\overline{W}$) are undefined during reset. The steps followed in preparing the NEURON CHIP to execute application code are discussed below. These steps, and their effects on the NEURON CHIP pins, are summarized in Figure 4-17.

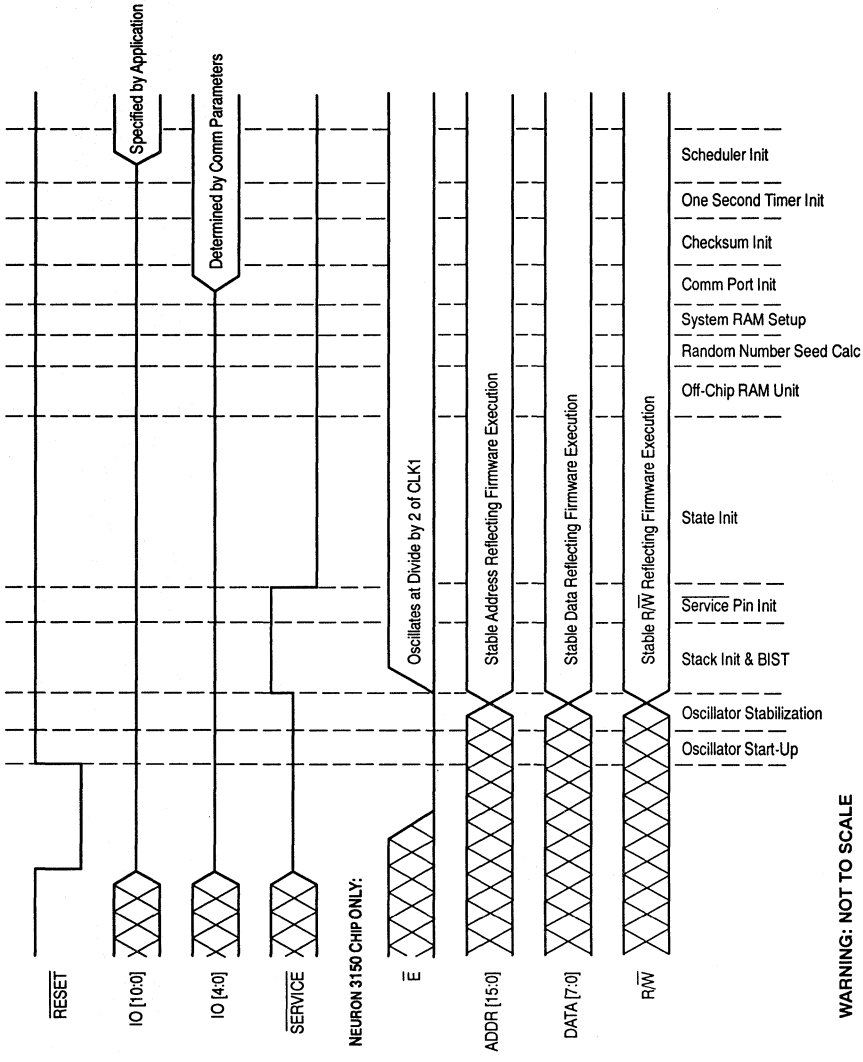


Figure 4-17. Reset Timeline

After the reset pin is released, the NEURON CHIP allows the oscillator to start up and stabilize and performs hardware and firmware initialization before executing application programs. Typical oscillator start-up times are discussed in Section 4.5, *Clocking System*. The NEURON CHIP allows for up to 15 transitions on CLK1 after the oscillator has started up for the oscillator to stabilize. Firmware initialization performs several tasks to initialize the NEURON CHIP and to ensure its integrity. These tasks are:

- Stack Initialization and Built-In Self-Test
- Service Pin Initialization
- State Initialization
- Off-Chip RAM Initialization
- Random Number Seed Calculation
- System RAM Setup
- Communication Port Initialization
- Checksum Initialization
- One Second Timer Initialization
- Scheduler Initialization

The Stack Initialization and Built-In Self-Test Task tests the on-chip RAM, the timer/counter logic, and the counter logic. For the test to pass, all three processors and the ROM must be functioning as well. A flag is set to indicate whether the NEURON CHIP passed or failed the Built-In Self-Test. The RAM is cleared to all 0's by the end of this step. At the beginning of this task, the pull-ups on IO[7:4] are enabled so that a weak high state can be observed on these pins. The Service Pin oscillates between a solid low and a weak high. The memory interface signals reflect execution of these tasks.

The Service Pin Initialization Task simply turns off the Service Pin.

The State Initialization Task determines if a NEURON CHIP boot is required, and, if so, performs it. The NEURON CHIP decides to perform a boot if it is blank, or if the boot ID does not match the boot ID in ROM (MC143150 only). The NEURON CHIP is assumed to be blank (i.e., the EEPROM is blank) if address 0xF02D is 0x00 or 0xFF; the manufacturer sets this address to the appropriate value during manufacture. The boot ID consists of two bytes starting at 0xF1FE which are set to the boot ID in ROM during the boot process; the user can force a boot process by clearing these two bytes and then resetting the NEURON CHIP.

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

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

The Random Number Seed Calculation Task creates a seed for the random number generator.

The System RAM Setup Task sets up internal system pointers as well as the linked lists of system buffers.

The Communication Port Initialization Task initializes the communication port according to the application specified communication port parameters and the MAC Processor begins handling packets. For special purpose mode, the configuration registers are initialized.

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

The One Second Timer Initialization Task initializes the one second timer. At this point, the Network Processor is available to accept incoming packets.

The Scheduler Initialization Task allows the Application Processor to perform application related initialization as follows:

- State Wait — wait for the node to leave the application-less state.
- Pointer Initialization — perform a global pointer initialization.
- Initialization Step — execute initialization task, which is created by the compiler/linker to handle initialization of static variables and the timer/counters.
- IO Pin Initialization Step — initialize IO pins based on application definition. Prior to this point, IO pins are high impedance.
- State Wait II — wait for the node to leave the unconfigured or hard-offline state. If waiting was required, a flag is set to indicate that the node should come up offline.
- Parallel IO Synchronization — nodes using parallel IO attempt to execute the master/slave synchronization protocol at this point.
- Reset Task — execute the application reset task (when (reset{ })).
- If the offline flag was set, go offline and execute the offline task. If the Built-In Self-Test flag indicated a failure, then the Service Pin is turned on and the offline task is executed. Otherwise the scheduler starts its normal task scheduling loop.

The amount of time required to perform these steps depends on many factors, including: NEURON CHIP model; input clock rate; whether or not the node performs a boot process; whether the node is application-less, configured, or unconfigured; amount of off-chip RAM; whether the off-chip RAM is tested, or simply cleared; the number of buffers allocated; and application initialization. Tables 4-14 and 4-15 summarize the number of input clock cycles (CLK1) required for each of these steps for the MC143120 and the MC143150. The times are approximate and are given as functions of the most significant application variables.

Table 4-14. Time Required for MC143120 to Perform Reset Sequence

Step	Number of CLK1 Cycles	Notes
Oscillator Start-Up	See Section 4.5, *Clocking System*	
Oscillator Stabilization	15	
Stack Initialization and Built-In Self-Test	200,000	
Service Pin Initialization	1000	
State Initialization	250 (for no boot) 2,275,000 (for boot)	
Off-Chip RAM Initialization	0	
Random Number Seed Calculation	0	1
System RAM Setup	21,000 + 600*B	2
Communication Port Initialization	0	1
Checksum Initialization	3400 + 175*M	3
One Second Timer Initialization	6100	
Scheduler Initialization	≥7400	4

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

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

Oscillator Start-Up	1.6 ms (from Table 4-10)
Oscillator Stabilization	0.002 ms
Stack Initialization and Built-In Self-Test	20 ms
Service Pin Initialization	0.1 ms
State Initialization	0.025 ms
Off-Chip RAM Initialization	0
Random Number Seed Calculation	0
System RAM Setup	2.7 ms
Communication Port Initialization	0
Checksum Initialization	10.8 ms
One Second Timer Initialization	0.61 ms
Scheduler Initialization	0.74 ms
Total	36.6 ms

Table 4-15. Time Required for MC143150 to Perform Reset Sequence

Step	Number of CLK1 Cycles	Notes
Oscillator Start-Up	See Section 4.5, *Clocking System*	
Oscillator Stabilization	15	
Stack Initialization and Built-In Self-Test	425,000	
Service Pin Initialization	1000	
State Initialization	1300 (for no boot) 70,000 + 25 ms*E (for boot)	 1
Off-Chip RAM Initialization	24,000 + 214*R (for test and clear) 24,000 + 152*R_a (for clear only)	2 3
Random Number Seed Calculation	50,000 max	
System RAM Setup	27,000 + 1500*B	4
Communication Port Initialization	0	5
Checksum Initialization	7200 + 175*M (for no boot) 82,000 + 100 ms + 175*M (for boot)	6 6, 7
One Second Timer Initialization	6100	
Scheduler Initialization	≥7400	8

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

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

Oscillator Start-Up	1.6 ms (from Table 4-10)
Oscillator Stabilization	0.002 ms
Stack Initialization and Built-In Self-Test	42.5 ms
Service Pin Initialization	0.1 ms
State Initialization	0.13 ms
Off-Chip RAM Initialization	353 ms
Random Number Seed Calculation	5 ms
System RAM Setup	4.2 ms
Communication Port Initialization	0
Checksum Initialization	11.2 ms
One Second Timer Initialization	0.61 ms
Scheduler Initialization	0.74 ms
Total	419 ms

4.7 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 4-18. During reset the $\overline{\text{Service}}$ pin state is indeterminate. The default state of the service pin pull-up is enabled.

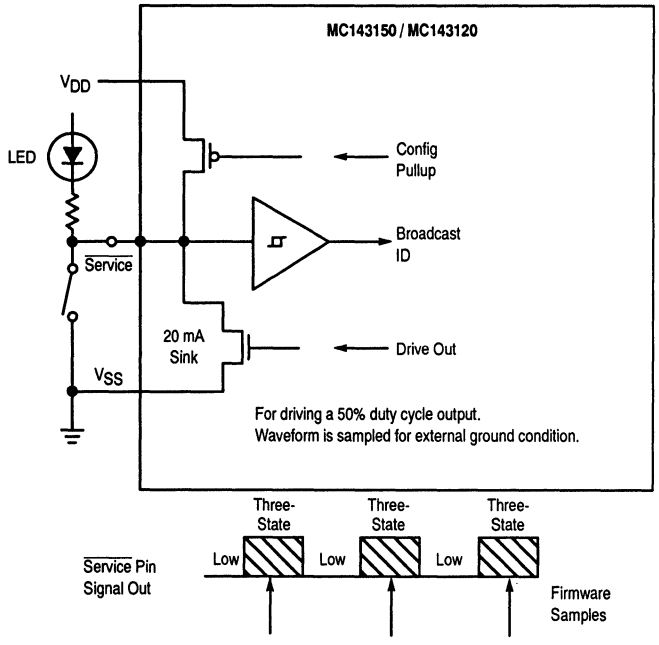


Figure 4-18. $\overline{\text{Service}}$ Pin Circuit

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 29 different I/O objects available in the MC143150 system image — 15 for input, 11 for output, and 3 bidirectional objects. There are 24 available in the MC143120 internal ROM image. The additional objects can be loaded into the MC143120 EEPROM.

The NEURON CHIP has two 16-bit timer/counters on-chip (see Figure 3-10). The input to timer/counter 1 is selectable among pins IO4 through IO7 via a programmable multiplexer (mux) and its output may be connected to pin IO0. The input to timer/counter 2 may be connected to pin IO4 and its output to pin IO1. The timer/counters are implemented as a 16-bit load register writeable by the CPU, a 16-bit counter, and a 16-bit latch readable by the CPU. The load register and latch are accessed a byte at a time. Note that no I/O pins are dedicated to timer/counter functions. If for example, timer/counter 1 is used for input signals only, then IO0 is available for other input or output functions. Timer/counter clock and enable inputs may be from external pins, or from scaled clocks derived from the system clock; the clock rates of the two timer/counters are independent of each other. External clock actions occur optionally on the rising edge, the falling edge, or both rising and falling edges of the input.

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

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, and illustrated in Figure 5-1. 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 at 10 MHz. The accuracy is ±10%. Most latency values scale up at lower input clock rates.

Table 5-1. Summary of Input Device Driver Objects

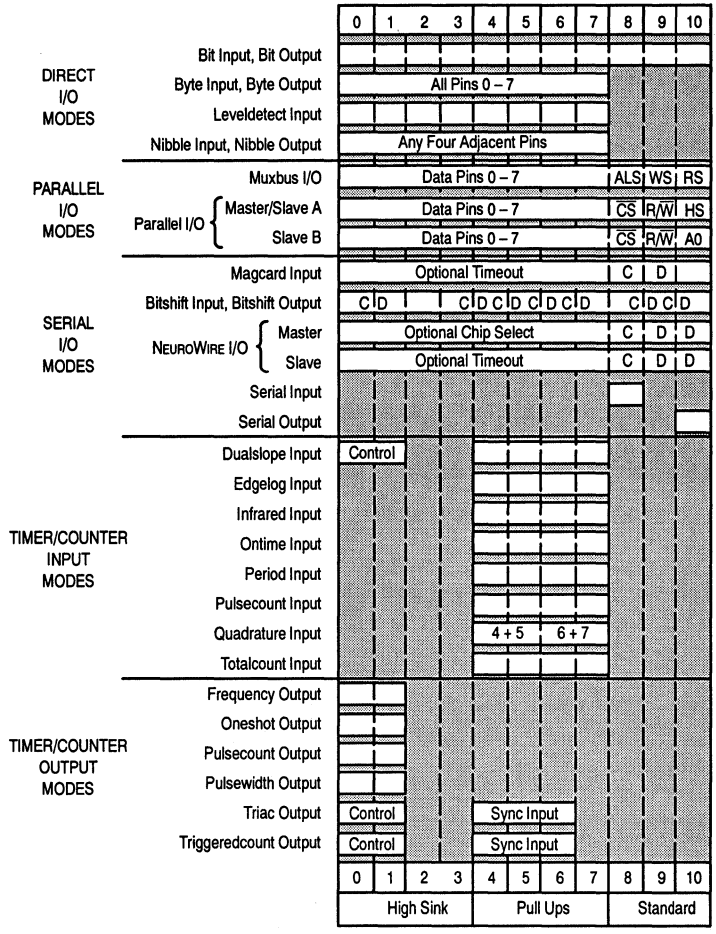
I/O Mode	Applicable I/O Pins	Input Value
Bit	IO0–IO10	0, 1 binary data
Bit Shift	Any adjacent pair (with even bit being the clock)	Up to 16 bits of clocked data
Byte	IO0–IO7	0 .. 255 binary data
Level Detect	IO0–IO7	Negative edge
Nibble	IO0..IO3 to IO4..IO7	0 .. 15 binary data
On-Time	IO4–IO7	Pulse width 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
Dualslope	IO0, IO1 + IO4–IO7	Analog signal
Edgelog	IO4	A stream of input transitions
Infrared	IO4–IO7	Encoded data stream from an infrared demodulator
Magcard	IO8 + IO9 + IO0–IO7	Encoded ISO7811 track 2 data stream from a magnetic card reader

Table 5-2. Summary of Output Device Driver Objects

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 Bidirectional Device Driver Objects

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
Muxbus	IO0–IO10	Parallel bidirectional port using multiplexed address technique



Timer/Counter 1 Devices
either: IO_6 input quadrature
or: IO_4 input edgelog
or: IO_0 output [triac | triggeredcount] sync([IO_4...IO_7])
or: IO_0 output [frequency | oneshot | pulsecount | pulsewidth]
or: IO_4...IO_7] input dualslope mux
or: Up to four of:
IO_4 input mux [infrared | ontime | period | pulsecount | totalcount]
[IO_5...IO_7] input [infrared | ontime | period | pulsecount | totalcount]

Timer/Counter 2 Devices
either: IO_4 input quadrature
or: IO_4 input edgelog
or: IO_1 output [triac | triggeredcount] sync(IO_4)
or: IO_1 output [infrared | frequency | oneshot | pulsecount | pulsewidth]
or: IO_4 input dualslope
or: IO_4 input ded [infrared | ontime | period | pulsecount | totalcount]

Figure 5-1. Summary of I/O Objects

To maintain and provide consistent behavior for external events and to prevent metastability, 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 5-2).

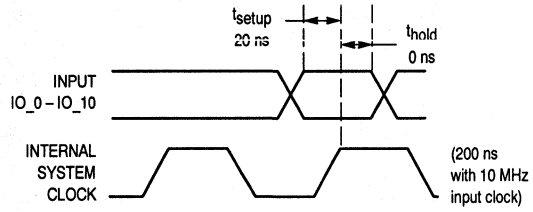


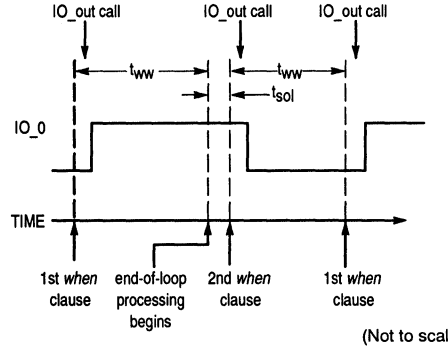
Figure 5-2. Synchronization of External Signals

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 Programmer's Guide* for more information). These latency values scale up proportionally at lower clock rates. Thus any event that lasts longer than 220 ns will be synchronized by hardware, but there will be latency in software sampling resulting in a delay detecting the event. If the state changes at a faster rate than software sampling can occur then the interim changes will go undetected. 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 Programmer's 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 5-3 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 5-3. *when* -Clause to *when* -Clause Latency, t_{ww} and Scheduler Overhead Latency, t_{sol}

Figure 5-3 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 objects 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 objects. They do not suspend network communication as this is handled by the NETWORK processor and the MEDIA ACCESS processor.

5.3 DIRECT OBJECTS (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. Refer to Figure 5-4. Figures 5-5 and 5-6 show the bit input and bit output latency times respectively. These are the times from which IO_in or IO_out is called until a value is returned. The direction of bit ports may be changed between input and output dynamically under application control.

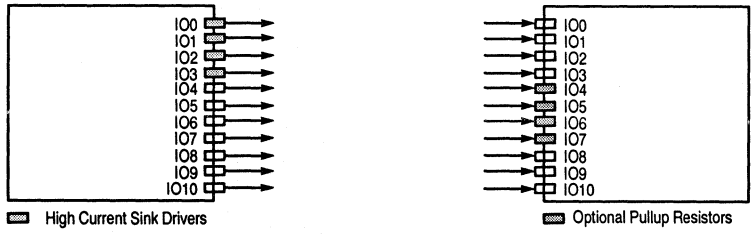
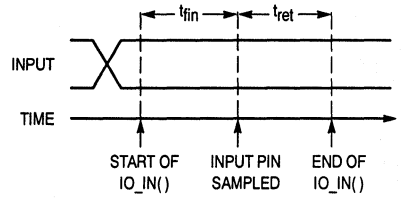
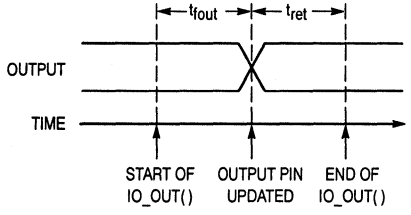


Figure 5-4. Bit I/O



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

Figure 5-5. Bit Input Latency Values



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

Figure 5-6. 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 5-7, 5-8, and 5-9. IO0 represents the LSB of data. The direction of a byte port may be changed between input and output dynamically under application control.

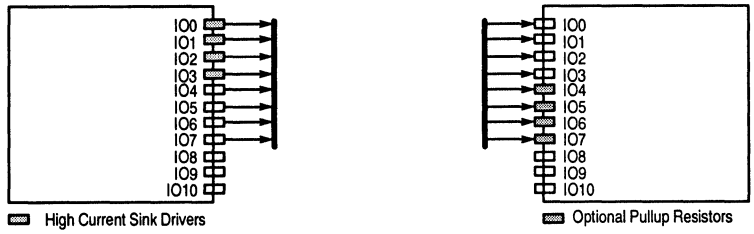
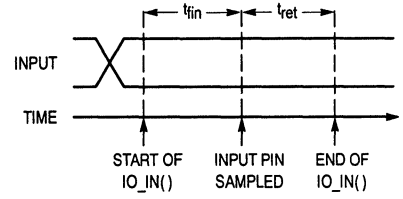
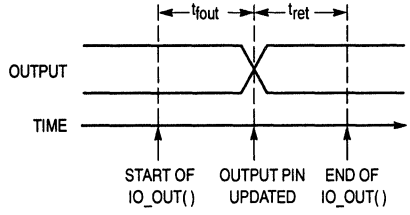


Figure 5-7. Byte I/O



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

Figure 5-8. Byte Input Latency Values

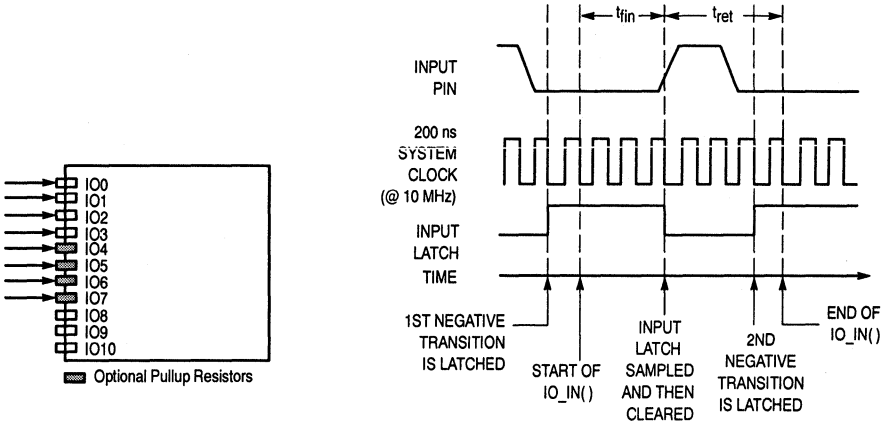


Symbol	Description	Typ @ 10 MHz
t_{fout}	Function call to update	57 µs
t_{ret}	Return from function	5 µs

Figure 5-9. 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 object which is latched before it is sampled. The latch is cleared during the *when* statement sampling and can be set again immediately after, if another transition should occur. See Figure 5-10.



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

Figure 5-10. 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 5-11, 5-12, and 5-13.

The direction of nibble ports may be changed between input and output dynamically under application control. The LSB of the input data is determined by the object declaration and can be any of the IO0 to IO4 pins.

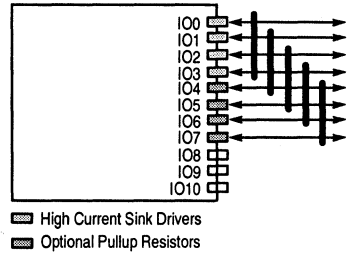
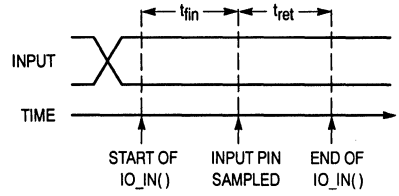
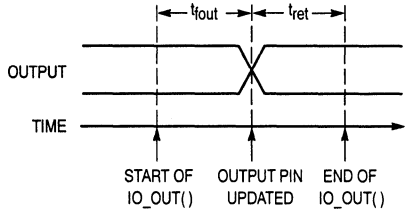


Figure 5-11. Nibble I/O



Symbol	Description	Typ @ 10 MHz
t_{fin}	Function call to sample IO0 to IO4	41 µs
t_{ret}	Return from function	
	IO0	18 µs
	IO1	22.75 µs
	IO2	27.5 µs
	IO3	32.25 µs
	IO4	37 µs

Figure 5-12. Nibble Input Latency Values



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

Figure 5-13. Nibble Output Latency Values

5.4 PARALLEL I/O INTERFACE OBJECT

For additional information consult AN1208/D, "Parallel I/O Interface to the NEURON CHIP."

5.4.1 Introduction

Pins IO0 through IO10 may be configured as a bidirectional 8-bit data and 3-bit control port for connecting to an external processor. The other processor may be a computer, microcontroller, or another NEURON CHIP (for bridge, router, gateway, or other applications). The parallel object may be configured in master, slave A, or slave B mode. Typically, two NEURON CHIPS interface in master/slave A mode and a NEURON CHIP interfaces with a non-NEURON CHIP processor in the slave B configuration with the non-NEURON CHIP as the master. Handshaking is used in both modes to control the instruction execution, and application

processing is suspended for the duration of the transfer (up to 255 bytes per transfer). Consult the *NEURON C Programmer's Guide* for detailed programming instructions. Upon a reset condition the master processor monitors the low transition of the handshake line from the slave, then passes a CMD_RESYNC (0x5A) for synchronization purposes. This must be done within 0.84 second after reset goes high with a NEURON CHIP slave running at 10 MHz to avoid a watchdog reset error condition (see *NEURON C Programmer's Guide*). The CMD_RESYNC is followed by the slave acknowledging with a CMD_ACKSYNC (0x07). This synchronization ensures that both processors are properly reset before data transfer occurs. When interfacing two NEURON CHIPS, these characters are passed automatically (refer to the flow table illustrated later in this section). However, a non-NEURON CHIP must duplicate the interface signals and characters that are automatically generated by the parallel I/O function.

5.4.2 Master/Slave A Mode

This mode is recommended when interfacing two NEURON CHIPS. In a master/slave A configuration, the master drives IO8 as a chip select and IO9 to specify a read or write cycle, and the slave drives IO10 as a handshake acknowledgement (see Figure 5-14). 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 (note that a master write is a slave read). Timing for the case where the NEURON CHIP is the master (Figure 5-15) 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. Slave A timing is shown in Figure 5-16.

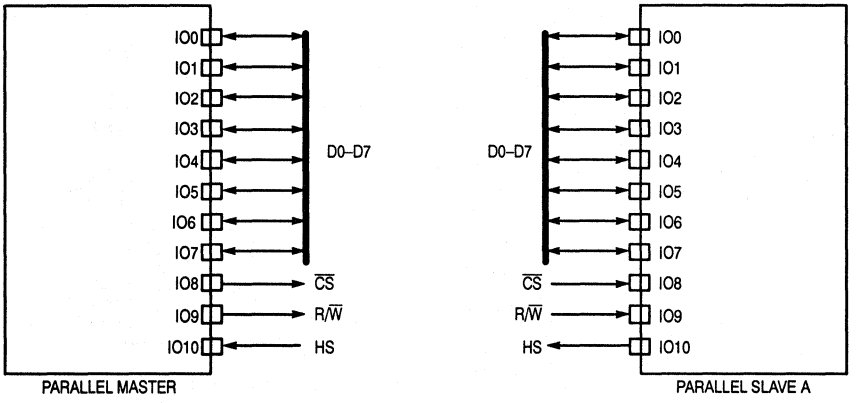
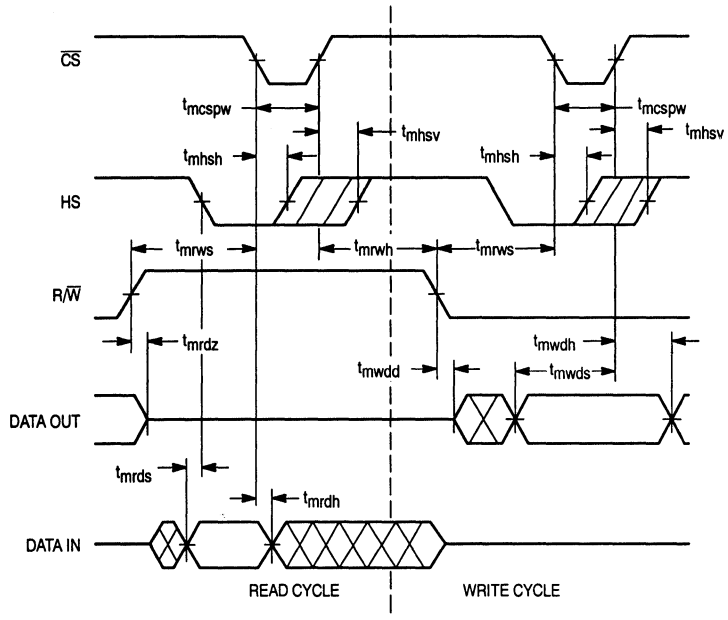


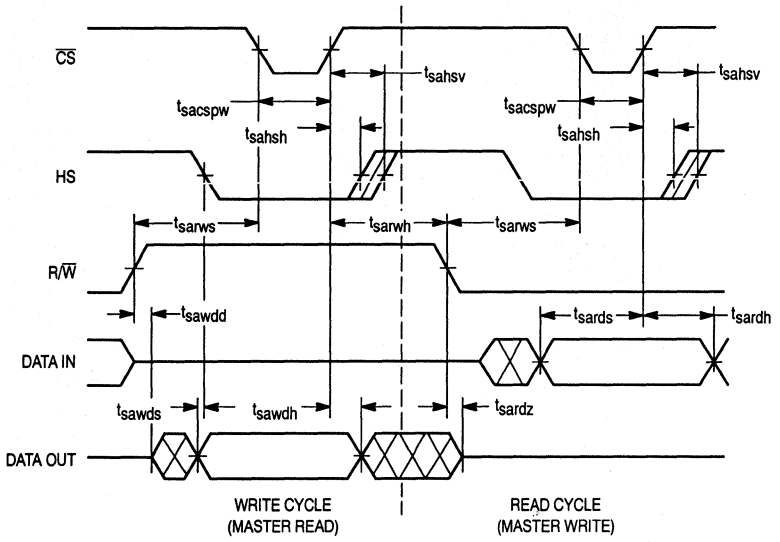
Figure 5-14. Parallel I/O — Master and Slave A



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

NOTES:
1. Master will hold output data valid during a write until the Slave device pulls HS low.
2. CLK1 represents the period of the NEURON CHIP input clock (100 ns at 10 MHz).

Figure 5-15. Master Mode Timing



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

NOTE: CLK1 represents the period of the NEURON CHIP input clock (100 ns at 10 MHz)

Figure 5-16. Slave-A Mode Timing

Following is an example program for transferring data in a parallel I/O master/slave A configuration. The code is for two NEURON emulators hardwired as shown in Figure 5-14. 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 Developer's Workbench to verify the transfer was complete. The master transmits [5,1,1,1,1,1] to the slave and the slave transmits [7,1,2,3,4,5,6,7,0,0,0,0,0,0] 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, &p_out);    // output buffer when slave is ready
}
when (io_in_ready(parallel_bus))
{
    p_in.len=MAX_IN;                 // declare the maximum input buffer acceptable
    io_in(parallel_bus, &p_in);      // store input data in buffer
}    //end of program

/* This is the slave program. After reset, the output buffer is filled with data and then the slave reads from
   the master. The slave then writes to the master. The slave's input buffer should contain [5,1,1,1,1,1]; the
   output buffer should contain [7,1,2,3,4,5,6,7,0,0,0,0,0,0].
 /*
IO_0 parallel slave parallel_bus;
#define MAX_IN 5                     // maximum length of input data expected
#define OUT_LEN 7                    // output length can be equal to or less than the max
#define MAX_OUT 13                   // maximum 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, &p_out);   // output buffer
}

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

Debugging the Above Programs: 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 Slave B Mode

The slave B mode is recommended for interfacing a NEURON CHIP (as the slave) to a non-NEURON CHIP processor as the master. When configured in slave B mode, the NEURON CHIP accepts IO8 as a chip select and IO9 to specify whether the master will read or write, and accepts IO10 as a register select input. When $\overline{CS}$ is asserted and either IO10 is low or IO10 is high and R/$\overline{W}$ is low, pins IO0 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 status register. Therefore, reads by the master to an odd address access the status register for handshaking 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 5-17 and 5-18.

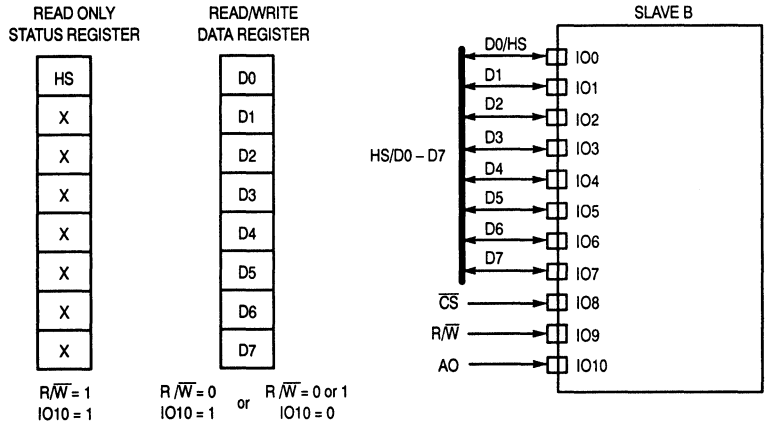


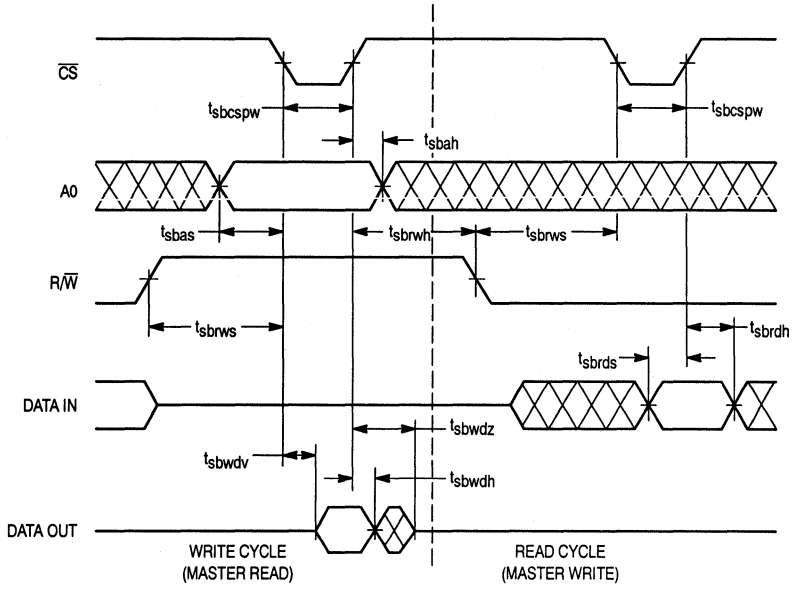
Figure 5-17. Parallel I/O Master/Slave B
(NEURON CHIP as Memory-Mapped I/O Device)

5.4.4 Token Passing

Virtual token passing is implemented to eliminate the possibility of data bus contention. The token is owned by the master after synchronization and is passed between the master and slave nodes. After each data transfer is completed the token owner writes an EOM (0x00) to indicate transfer is complete. The EOM is never read. Instead "processing the EOM" indicates passing of the token. Token passing can be achieved by executing either a data packet or a NULL transfer. Only the owner of the token can write to the bus. Therefore, when the master performs two writes of data (1 – 255 bytes each) a dummy read cycle (NULL character = 0x00) must be inserted between them in order to pass the token. Token passing is executed automatically in a NEURON CHIP to NEURON CHIP interface. Refer to 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 or slave A mode, the NEURON CHIP handshake line is pin IO10. In slave B mode, the NEURON CHIP handshake bit is monitored on IO0 which corresponds to the least significant data bit of the status register.



Symbol	Description	Min	Typ	Max
t_{sbrws}	R/$\overline{W}$ setup before falling edge of $\overline{CS}$	25 ns	—	—
t_{sbrwh}	R/$\overline{W}$ hold after rising edge of $\overline{CS}$	0 ns	—	—
t_{sbcspw}	$\overline{CS}$ pulse width	50 ns*	—	—
t_{sbas}	A0 setup to falling edge of $\overline{CS}$	50 ns	—	—
t_{sbah}	A0 hold after rising edge of $\overline{CS}$	0 ns	—	—
t_{sbwdv}	$\overline{CS}$ to write data valid	—	—	50 ns
t_{sbwdh}	Write data hold after rising edge of $\overline{CS}$	0 ns	30 ns	—
t_{sbwdz}	$\overline{CS}$ rising edge to Slave-B release data bus	—	—	50 ns
t_{sbrds}	Read data setup before rising edge of $\overline{CS}$	25 ns	—	—
t_{sbrdh}	Read data hold after rising edge of $\overline{CS}$	5 ns	—	—

*The t_{sbwdv} value is 50 ns worst case. The data hold time is 0 ns worst case. Therefore the actual $\overline{CS}$ minimum value is dependent on the master's read data setup time.

Figure 5-18. Slave-B Mode Timing

5.4.6 Data Transferring

Resynchronization Procedure: The following procedure applies to master/slave A and master/slave B configuration. The master initiates the resynchronization with a RESYNC (0x5A) command, and the slave acknowledges with 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=0x00)
	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 OBJECTS

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 object is useful for transferring data to external logic employing shift registers. This function suspends application processing until the operation is complete. See Figures 5-19, 5-20, and 5-21.

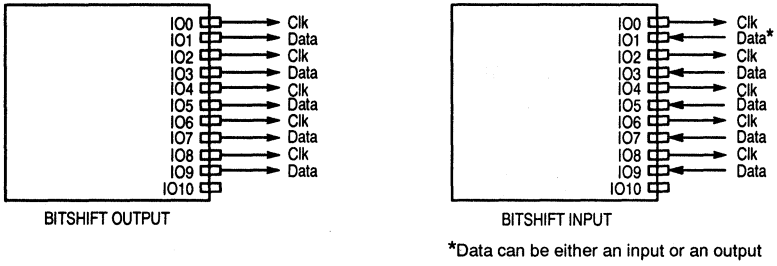
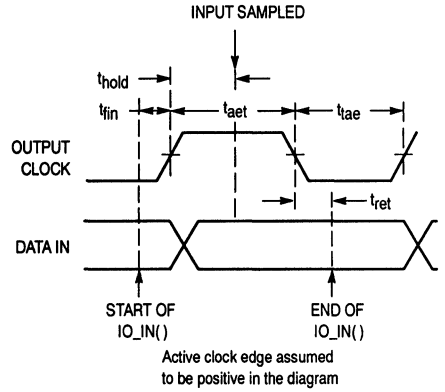
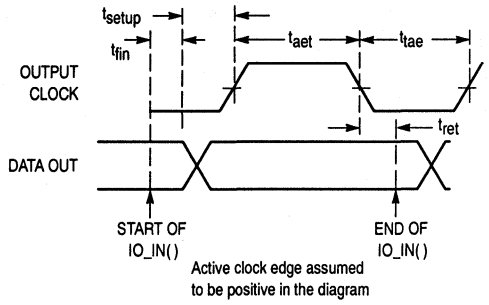


Figure 5-19. Bitshift I/O



Symbol	Description	Typ @ 10 MHz
t_{fin}	Function call to first edge	156.6 µs
t_{ret}	Return from function	5.4 µs
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 = 1 / (t_{aet} + t_{tae}) 15 kbaud data rate 10 kbaud data rate 1 kbaud data rate	 21.6 kHz 12.8 kHz 1.03 kHz

Figure 5-20. Bitshift Input Latency Values



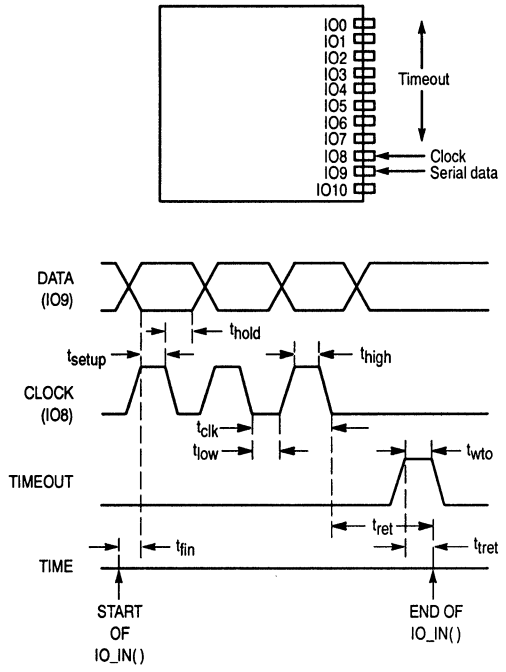
Symbol	Description	Typ @ 10 MHz
t_{fin}	Function call to first data out stable	
	16 bit shift count	185.3 µs
	1 bit shift count	337.6 µs
t_{ret}	Return from function	10.8 µs
t_{setup}	Data out stable to active clock edge	
	15 kbaud data rate	10.8 µs
	10 kbaud data rate	10.8 µs
	1 kbaud data rate	10.8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

Figure 5-21. Bitshift Output Latency Values

5.5.2 Magcard Input

This I/O object is used to transfer synchronous serial data from an ISO 7811 Track 2 magnetic stripe card reader in real time. The data is presented as a data signal input on pin IO_9, and a clock, or a data strobe, signal input on pin IO_8. The data on pin IO_9 is clocked on or just following the falling (negative) edge of the clock signal on IO_8, with the least significant bit first. In addition, any one of the pins IO_0 to IO_7 may be used as a timeout pin to prevent lockup in case of abnormal abort of the input bit stream during the input process.

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



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

Figure 5-22. Magcard Input Object

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

5.5.3 NEUROWIRE (SPI Interface) I/O Object

The NEUROWIRE object implements a full-duplex synchronous transfer of data to some peripheral device. It can operate as the master (drive a clock out) or as the slave (accept a clock in). In both master and slave modes, up to 255 bits of data may be transferred at a time. The NEUROWIRE I/O suspends application processing until the operation is completed. NEUROWIRE object is useful for external devices, such as A/D, D/A converters, and display drivers incorporating serial interfaces that conform with Motorola's SPI interface. See Figure 5-23.

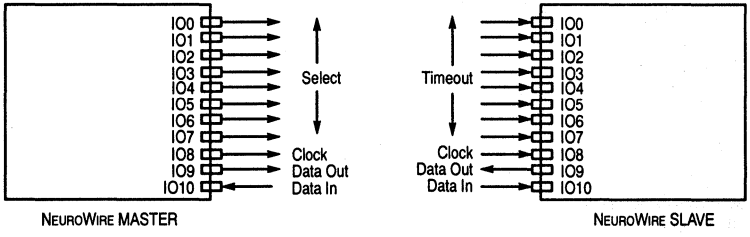
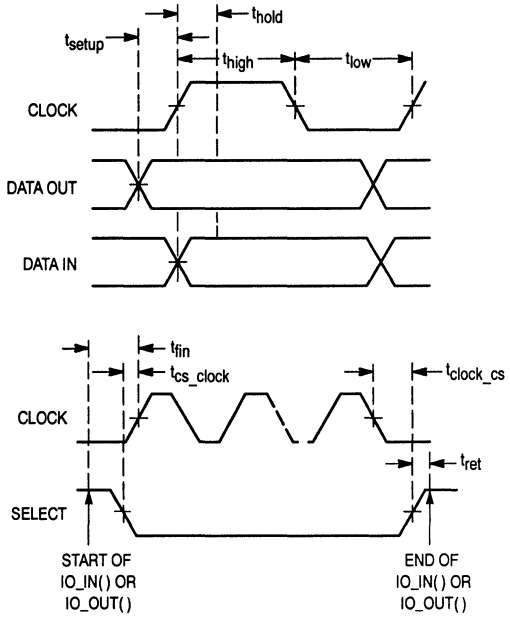


Figure 5-23. NEUROWIRE I/O

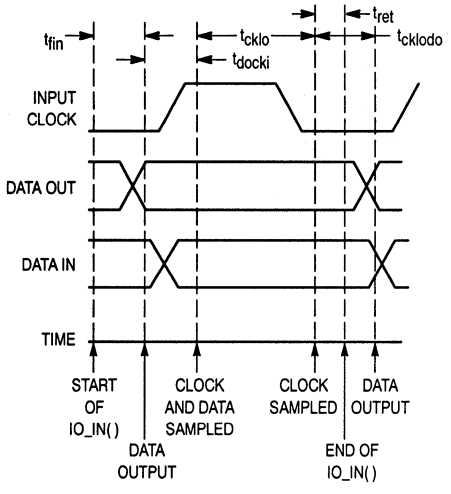
5.5.3.1 NEUROWIRE MASTER MODE. In NEUROWIRE master mode, pin IO8 is the clock (driven by the NEURON CHIP), IO9 is serial data output, and IO10 is serial data input. Serial data is clocked out on pin IO9 at the same time as data is clocked in from pin IO10. Data is clocked by the rising edge of the clock 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 5-24.



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

Figure 5-24. NEUROWIRE (SPI) Master Timing

5.5.3.2 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 5-25.

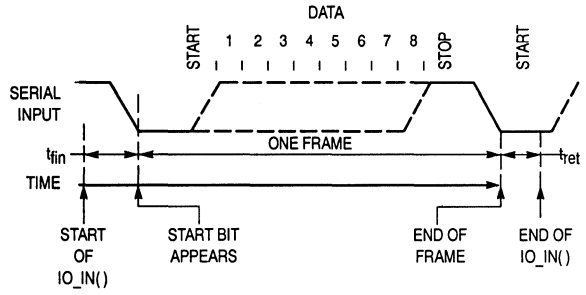


Parameter	Description	Typ
t_{fin}	Function call to data bit out	41.4 µs
t_{ret}	Return from function	19.2 µs
t_{docki}	Data out to inout clock and data sampled	4.8 µs
t_{cklo}	Data sampled to clock low sampled	24.0 µs
t_{cklodo}	Clock low sampled to data output	25.8 µs
f	Clock frequency (max)	18.31 kHz

Figure 5-25. NEUROWIRE (SPI) Slave Timing

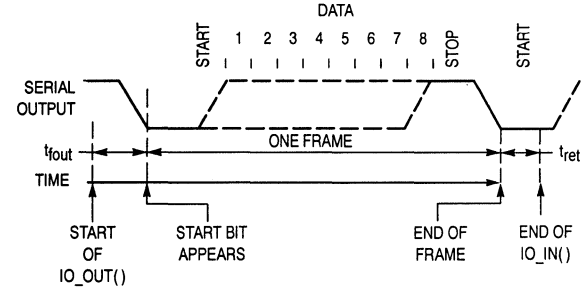
5.5.4 Serial I/O

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 character 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 5-26 and 5-27.



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 5-26. Serial Input



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

Figure 5-27. Serial Output

5.6 TIMER/COUNTER OBJECTS

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

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

Input timer/counter objects have the advantage (over non-timer/counter objects) in that input events will be captured even if the application processor is occupied doing something else when the event occurs. A true *when* statement condition for an event being measured by a timer/counter is the completion of the measurement and a value being returned to an event register. If the processor is delayed due to software processing and 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 5-28 shows an example of how the timer/counter objects are processed with NEURON C *when* statement.

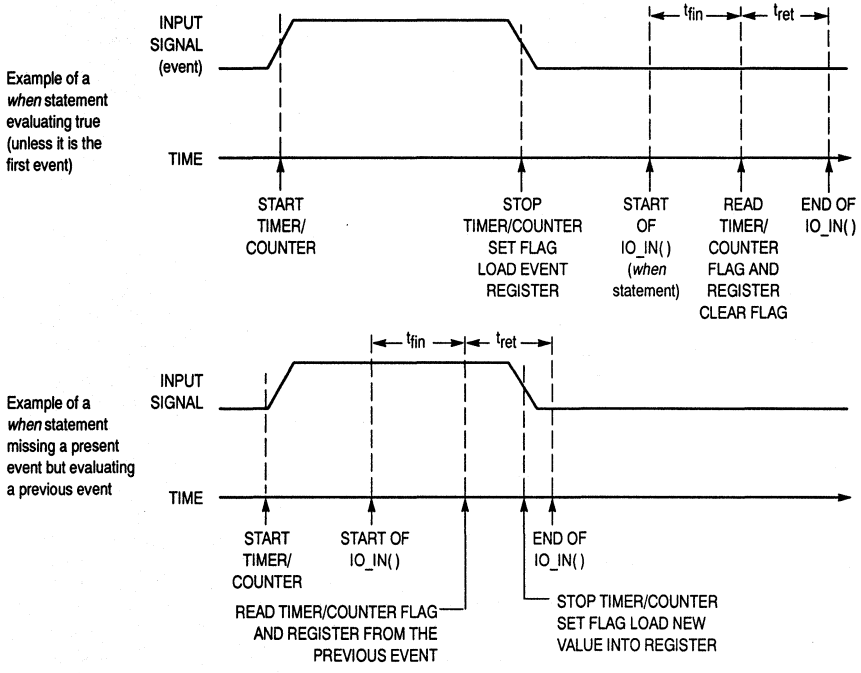
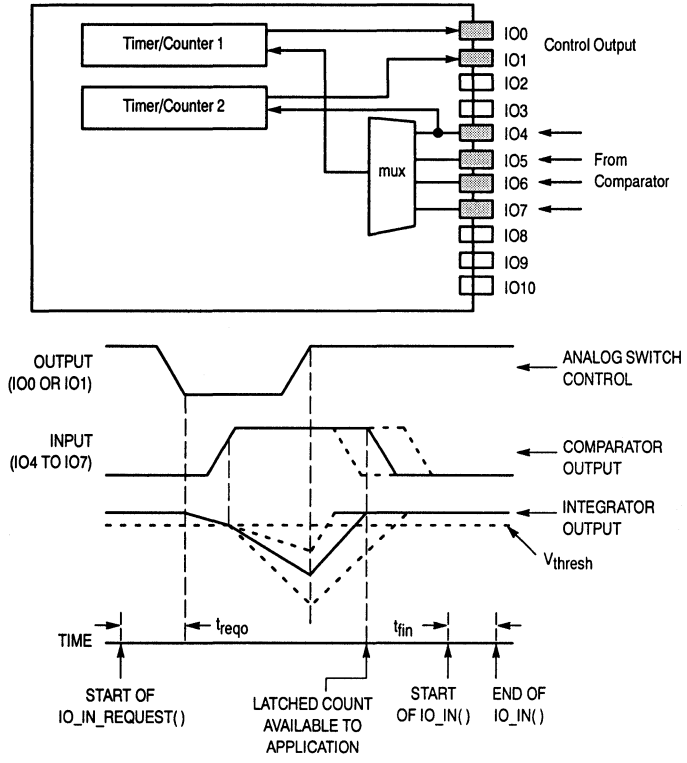


Figure 5-28. *when* Statement Processing Using the On-Time Input Function

5.6.1.1 DUALSLOPE INPUT. This input object uses a timer/counter to control and measure the integration periods of a dual slope integrating analog to digital converter (see Figure 5-29). The timer/counter provides the control out signal and senses a comparator output signal. The control output signal controls an external analog multiplexer which switches between the unknown input voltage and a voltage reference. The timer/counter's input pin is driven by an external comparator which compares the integrator's output with a voltage reference. A high level on the comparator input indicates the end of the conversion cycle, unless the *invert* keyword is used in the I/O declaration.

The resolution and range of the timer/counter period options is shown by Table 5-4 in Section 5.8.

For additional information regarding the dualslope A/D conversion and the NEURON CHIP, see EB155/D, "Analog to Digital Conversion with the NEURON CHIP."



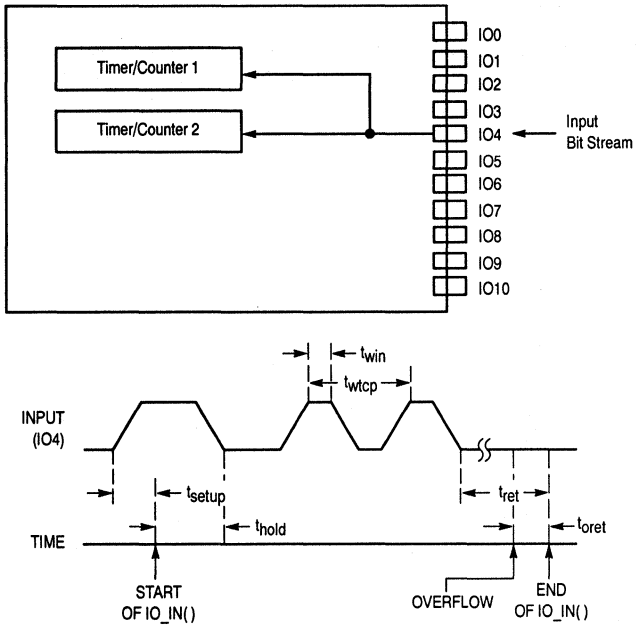
Symbol	Description	Min	Typ	Max
t_{reqo}	*io_in_request()* to output toggle	—	75.6 µs	—
t_{fin}	Input function call and return	—	82.8 µs	—

Figure 5-29. Dualslope Input Object

5.6.1.2 EDGELOG INPUT. The edgelog input object can record a stream of input pulses measuring the consecutive low and high periods at the input and storing them in user-defined storage (see Figure 5-30). The values stored represent the units of clock period between input signal edges, both rising and falling. Both timer/counters of the NEURON CHIP are used for this object.

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

The resolution and range of the timer/counter period options are shown in Table 5-4 in Section 5.8. This object is useful for analyzing an arbitrarily-spaced stream of input edges (or pulses), such as the output of a UPC bar-code reader.



Symbol	Description	Min	Typ	Max
t_{setup}	Input data setup	0	—	—
t_{win}	Input pulse width	1 T/C clk	—	65534 T/C clks
t_{hold}	*io_in()* call to data input edge for inclusion of that pulse	26.4 µs	—	—
t_{wtcp}	Two consecutive pulse widths	104 µs	—	—
t_{oret}	Return on overflow	—	42.6 µs	—
t_{ret}	Return on count termination	—	49.6 µs	—

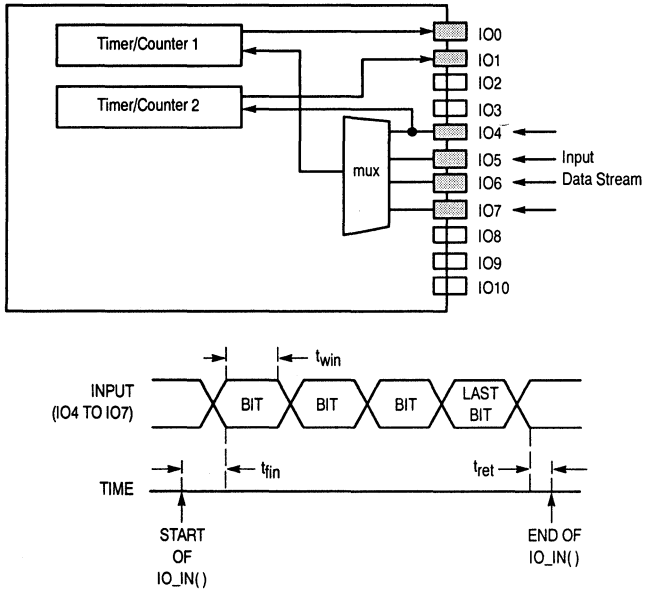
NOTE: T/C clk represents the period of the clock used during the declaration of the I/O object.

Figure 5-30. Edgelog Input Object

5.6.1.3 INFRARED INPUT. The infrared input object is used to capture a stream of data generated by a class of infrared remote control devices (see Figure 5-31). The input to the object is the demodulated series of bits from infrared receiver circuitry. The period of the on/off cycle determines the data bit value, a shorter cycle indicating a one, and a longer cycle indicating a zero. The actual threshold for the on/off determination is set at the time of the call of the function. The measurements are made between the negative edges of the input bits unless the *invert* keyword is used in the I/O declaration.

The infrared input object, based on the input data stream, generates a buffer containing the values of the bits received. The resolution and range of the timer/counter period options is shown by Table 5-4 in Section 5.8.

This function can be used with an off-the-shelf IR demodulator (e.g., NEC μPD1913 or Sharp GP1U50X) to quickly develop an infrared interface to the NEURON CHIP.

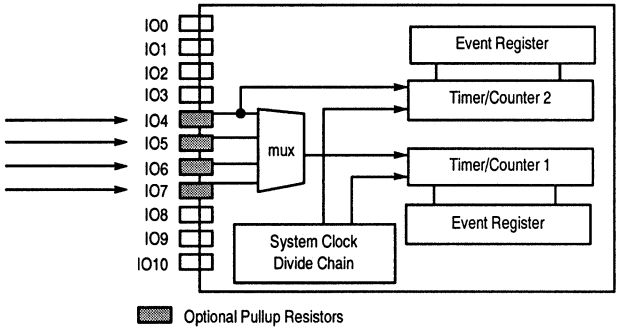


Symbol	Description	Min	Typ	Max
t_{fin}	Function call to start of input sampling	—	82.2 μs	—
t_{ret}	End of last valid bit to function return	max-period	max-period	—

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

Figure 5-31. Infrared Input Object

5.6.1.4 ON-TIME INPUT. A timer/counter may be configured to measure the time for which its input is asserted. Table 5-4 in Section 5.8 shows the resolution and maximum times for different I/O clock selections. Assertion may be defined as either logic high or logic low. This object may be used as a simple analog-to-digital converter with a voltage-to-time circuit, or for measuring velocity by timing motion past a position sensor. See Figures 5-28 and 5-32.



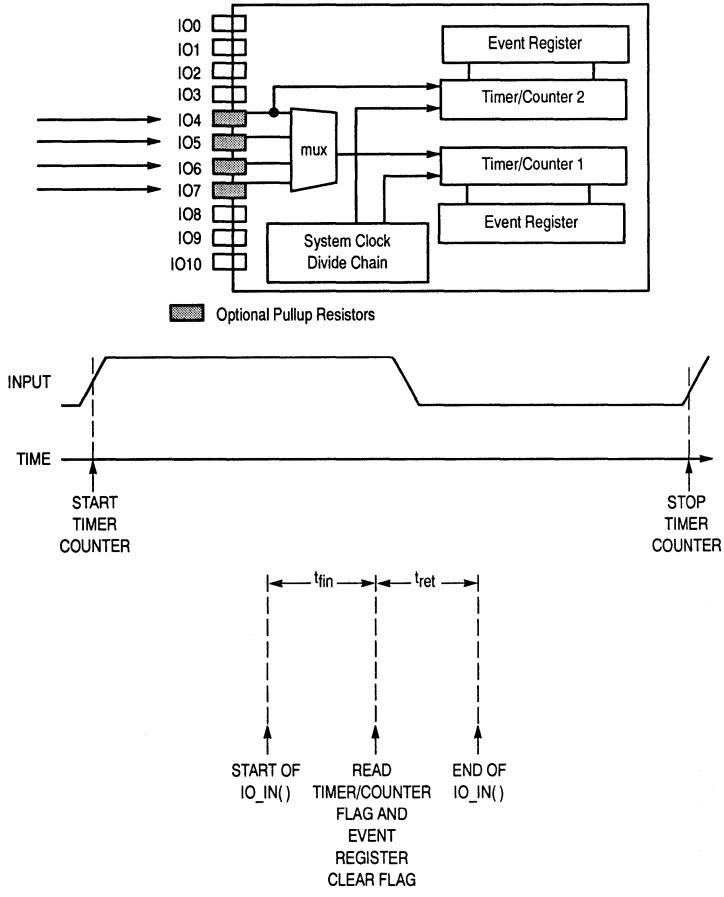
Reference Figure 5-28

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

*If the measurement is new, t_{ret} = 52 μs. If a new time is not being returned, t_{ret} = 22 μs.
NOTE: This is a level-sensitive function. The active level of the input signal gates the clock driving the internal counter in the NEURON CHIP.
The actual active level of the input depends on whether or not the invert option was used in the declaration of the function block. The default is the high level.

Figure 5-32. On-Time Latency Values

5.6.1.5 PERIOD INPUT. A timer/counter may be configured to measure the period from one rising or falling edge to the next corresponding edge on the input. Table 5-4 in Section 5.8 shows the resolution and maximum time measured for various clock selections. This object is useful for instantaneous frequency or tachometer applications. Analog-to-digital conversion can be implemented using a voltage-to-frequency converter with this function. See Figure 5.33.



Reference Figure 5-28

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

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

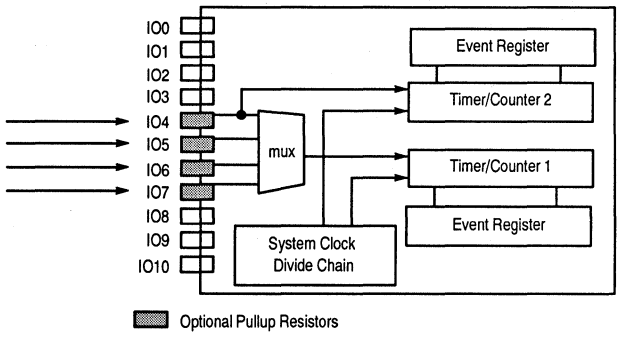
NOTE: This is a edge-sensitive function. The clock driving the internal counter in the NEURON CHIP is free running. The detection of active input edges stops and resets the counter each time.

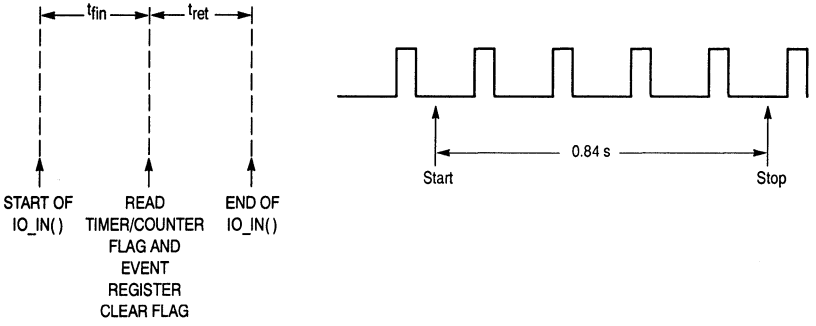
The actual active edge of the input depends on whether or not the invert option was used in the declaration of the function block. The default is the negative edge.

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

Figure 5-33. Period Input Latency Values

5.6.1.6 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 object is useful for average frequency measurements, or tachometer applications. See Figure 5-34.





Reference Figure 5-28

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

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

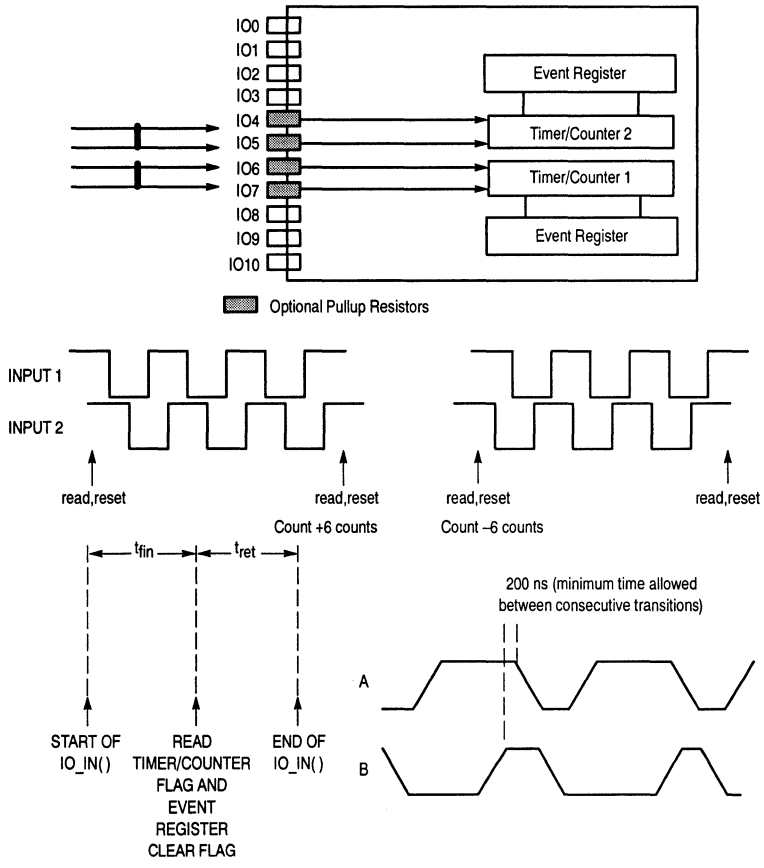
NOTE: This is a edge-sensitive function. The clock driving the internal counter in the NEURON CHIP is the actual input signal. The counter is reset automatically every 0.839 second.

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

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

Figure 5-34. Pulse Count Input Latency Values

5.6.1.7 QUADRATURE INPUT. A timer/counter may be configured to count transitions of a binary Gray code input on two adjacent input pins. The Gray code is generated by devices such as shaft encoders and optical position sensors which generate the bit pattern (00,01,11,10,00...) for one direction of motion and (00,10,11,01,00...) for the opposite direction. Reading the value of a quadrature object gives the arithmetic net sum of the number of transitions since the last time it was read (– 16,384 to 16,383). The maximum frequency of the input is one quarter of the input clock rate, for example 2.5 MHz at the maximum 10 MHz NEURON CHIP input clock. Quadrature devices may be connected to timer/counter 1 via pins IO6 and IO7, and timer/counter 2 via pins IO4 and IO5. See Figure 5-35. If the second input transitions low while the first input is low and high while the first input is high the counter counts up. Otherwise the count is down.



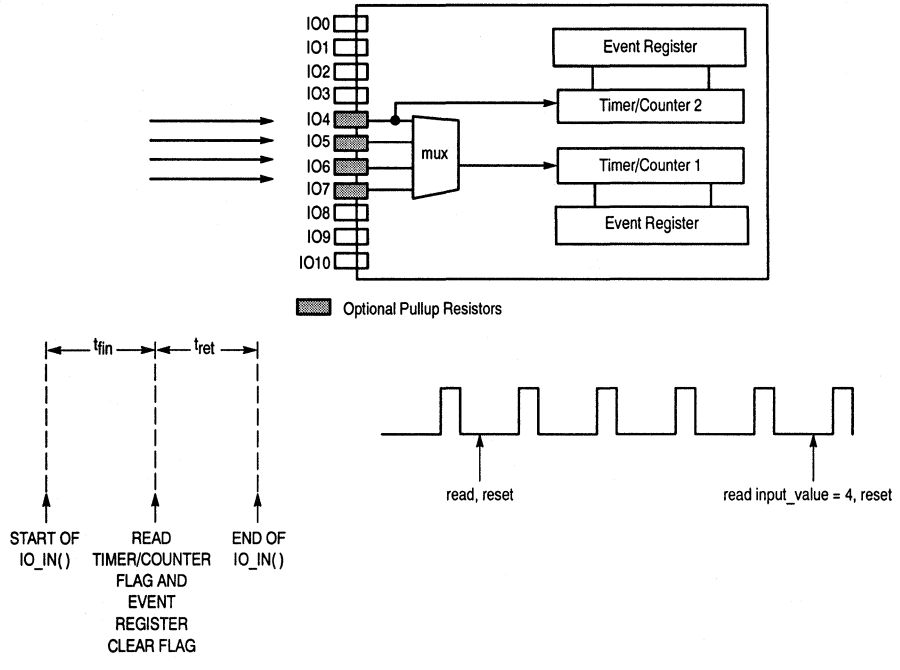
Reference Figure 5-28

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

NOTE: A call to this function returns the current value of the quadrature count since the last read operation. The counter is then reset and ready for the next series of input transitions.
The count returned is a 16-bit signed binary number, capped at ± 16K.
The number shown in the diagram above is the minimum time allowed between consecutive transitions at either input of the quadrature function block. For more information, see EB146/D, "NEURON CHIP Quadrature Input Function Interface."

Figure 5-35. Quadrature Input Latency Values

5.6.1.8 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 object is useful for counting external events such as contact closures, where it is important to keep an accurate running total. See Figure 5-36.



Reference Figure 5-28

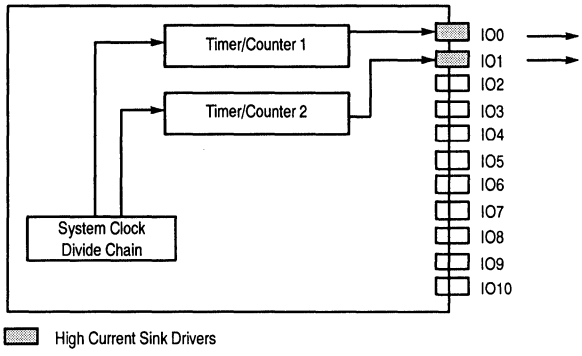
Symbol	Description	Typ @ 10 MHz
t_{fin}	Function call to input sample	92 μs
t_{ret}	Return from function	61 μs

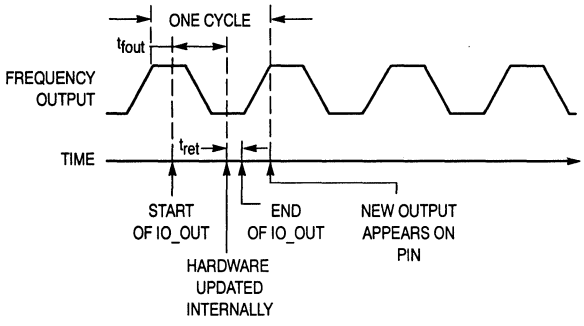
NOTE: A call to this function returns the current value of the totalcount value corresponding to the total number of active clock edges since the last call. The counter is then reset and ready for the next series of input transitions.
The actual active edge of the input depends on whether or not the invert option was used in the declaration of the function block. The default is the negative edge.

Figure 5-36. Total Count Input Latency Values

5.6.2 Timer/Counter Output Objects (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 object is useful for frequency synthesis to drive an audio transducer, or to drive a frequency to voltage converter to generate an analog output. See Figure 5-37.





Frequency Resolution and Maximum Range

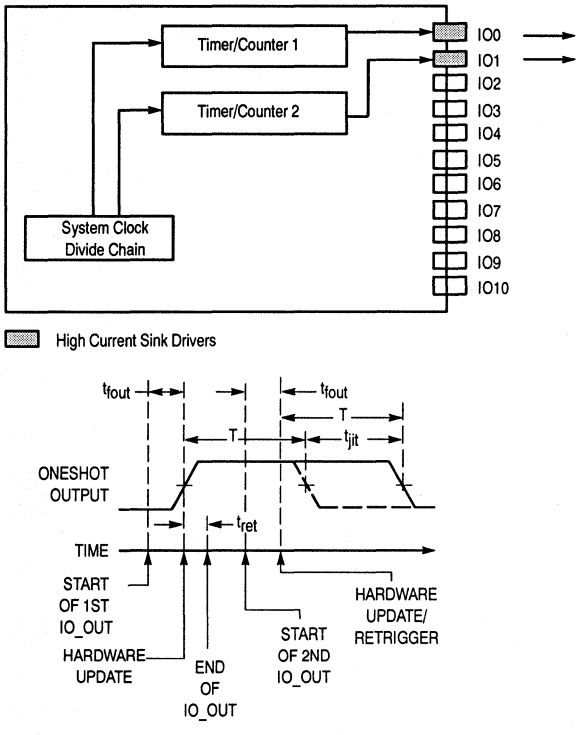
CLK	Resolution	Range	Unit
0	0.4	26.21	µs
1	0.8	52.42	µs
2	1.6	104.86	µs
3	3.2	209.71	µs
4	6.4	419.42	µs
5	12.8	838.85	µs
6	25.6	1677	µs
7	51.2	3355	µs

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

NOTE: A new frequency output value will not take effect until the end of the current cycle. There are two exceptions to this rule. If the output is disabled, the new (non-zero) output will start immediately after t_{fout}. Also, for a new output value of zero, the output is disabled immediately and not at the end of the current cycle.
A disabled output is a logic zero by default unless the *invert* keyword is used in the I/O object declaration.

Figure 5-37. Frequency Output Latency Values

5.6.2.2 ONE-SHOT OUTPUT. A timer/counter may be configured to generate a single pulse of programmable duration. The asserted state may be either logic high or logic low. Retriggering the one-shot before the end of the pulse causes it to continue for the new duration. Table 5-4 in Section 5.8 gives the resolution and maximum time of the pulse for various clock selections. This object is useful for generating a time delay without intervention of the application processor. See Figure 5-38.



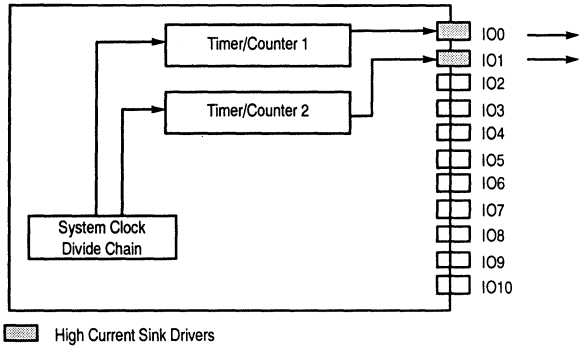
Symbol	Description	Typ @ 10 MHz
t_{fout}	Function call to output update	96 µs
t_{ret}	Return from function	13 µs
t_{jit}	Output duration jitter	1 timer/counter clock period*

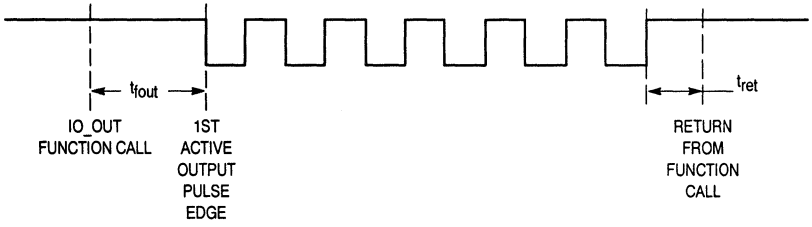
*Timer/counter clock period = 2000 * 2^(clock)/input clock (MHz).

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

Figure 5-38. 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 8 values given by Table 5-5 in Section 5.8, with clock select values of 0 through 7. This object is useful for external counting devices that can accumulate pulse trains, such as stepper motors. See Figure 5-39.



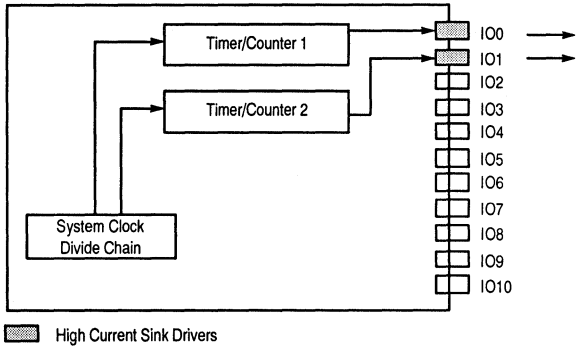


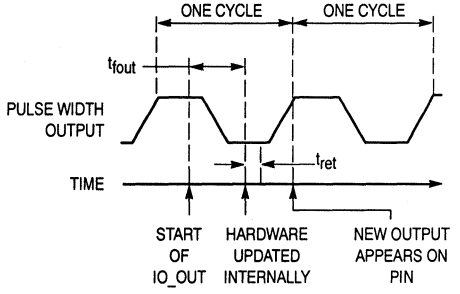
Symbol	Description	Typ @ 10 MHz
t_{fout}	Function call to first active output pulse edge	115 μs
t_{ret}	Return from function	5 μs

NOTE: The return from this function does not occur until all output pulses have been produced.
t_{fout} is the time from function call to first output pulse. Therefore, the calling of this function ties up the application processor for a period of N x (pulse period) + t_{fout} + t_{ret}, where N is the number of specified output pulses.
The polarity of the output depends on whether or not the invert option was used in the declaration of the function block. The default is low with high pulses.

Figure 5-39. 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 5-5 in Section 5.8. In pulsewidth long function, the duty cycle ranges from 0% to almost 100% (0/65,536 to 65,535/65,536) of a cycle in steps of 15.25 ppm (1/65,536). In pulsewidth long function, the frequency of the waveform may be one of the eight values given by Table 5-6 in Section 5.8. The asserted state of the waveform may be either logic high or logic low. Writing a new pulsewidth value to the device takes effect at the end of the current cycle. A pulsewidth modulated signal provides a simple means of digital to analog conversion. See Figure 5-40.





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

NOTE: The new output value will not take effect until the end of the current cycle. There are two exceptions to this rule. If the output is disabled, the new (non-zero) output will start immediately after t_{fout}. Also, for a new output value of zero, the output is disabled immediately and not at the end of the current cycle.
A disabled output is a logic zero by default unless the *invert* keyword is used in the I/O object declaration.

Figure 5-40. Pulsewidth 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 5-4 in Section 5.8 shows the resolution and maximum range of the delay. See Figure 5-41.

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

The output gate pulse may be configured to be either a pulse or a level. In the case of a pulse output, the gate signal is a 25 μs wide pulse, independent of the NEURON CHIP input clock. In the case of a level output, the output gate signal remains in the active state until the next zero crossing. This, for example, can be useful when the triac is driving a highly inductive load and the gate signal must remain active for the duration of the half cycle. Table 5-4 in Section 5.8 shows the resolution and maximum range of delay.

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

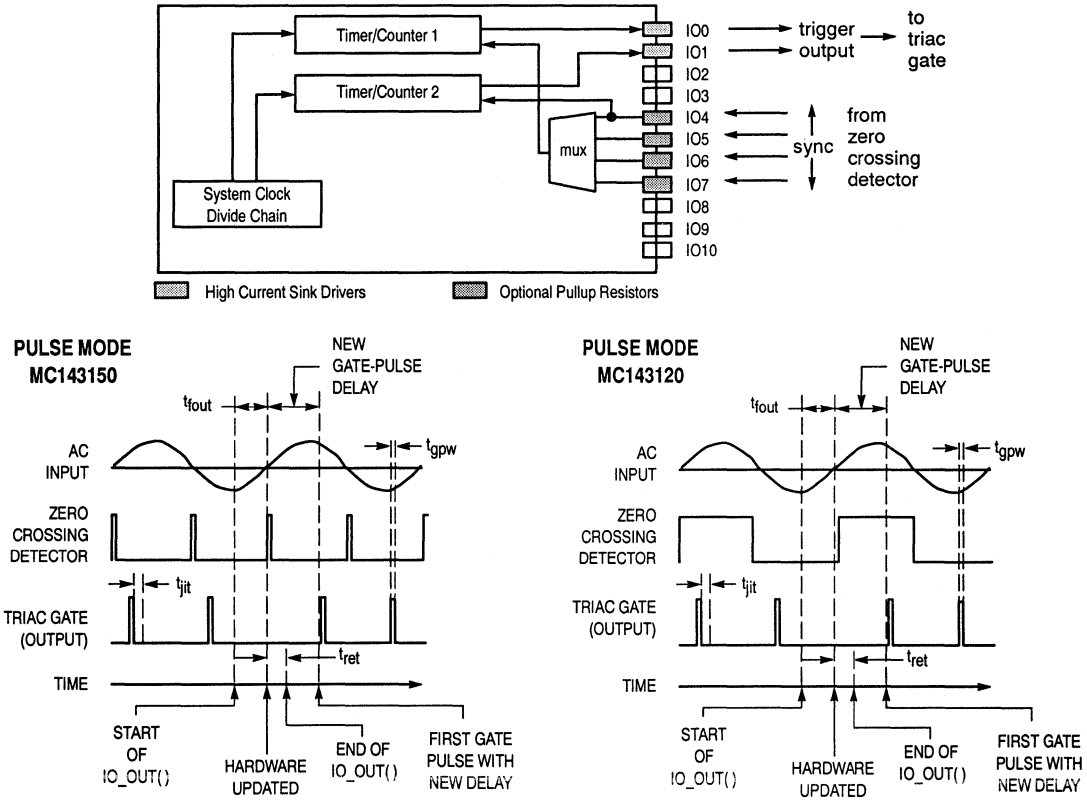
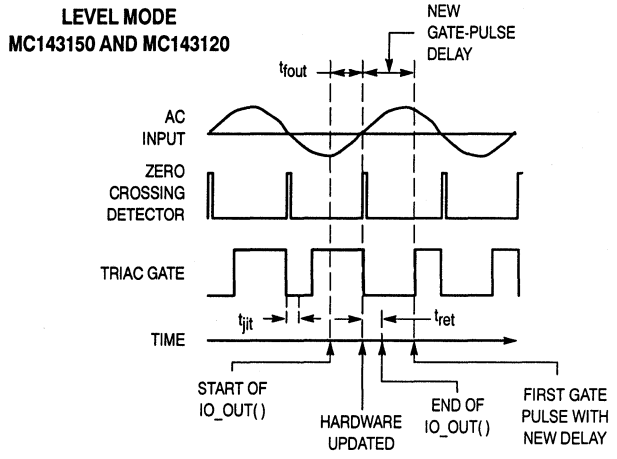


Figure 5-41. Triac Output Latency Values (Sheet 1 of 2)



Symbol	Description	Min	Typ @ 10 MHz	Max
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	—	25.6 µs	—
t_{jit}	Gate output jitter	0	—	26.5 µs

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

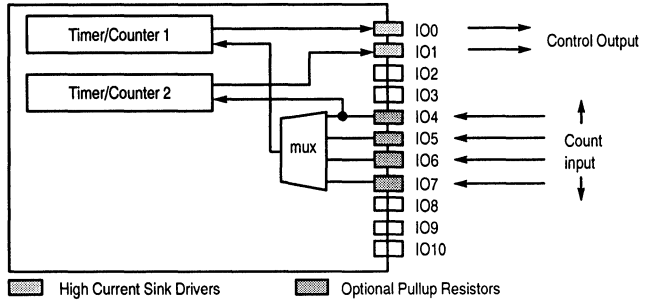
t_{fout} (min) refers to the delay from the initiation of the function call to the first sampling of the sync input. In the absence of an active sync clock edge, the input is repeatedly sampled for 10 ms (1/2 wave of a 50 Hz line cycle time), t_{fout} (max), during which the application processor is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

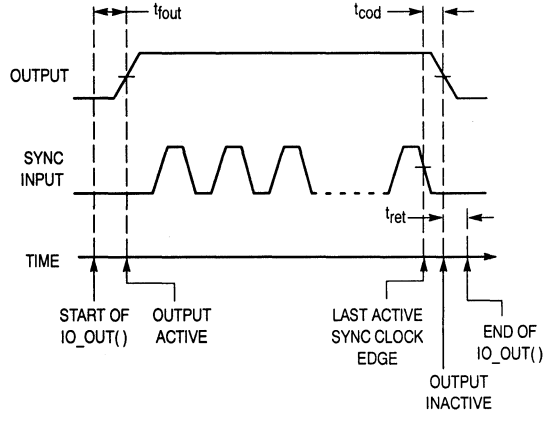
The output gate pulse is gated by an internal clock with a constant period of 25.6 µs (independent of the NEURON CHIP input clock). Since the input trigger signal (zero crossing) is asynchronous relative to this internal clock, there is a jitter, t_{jit}, associated with the output gate pulse.

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

Figure 5-41. Triac Output Latency Values (Sheet 2 of 2)

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 5-42.





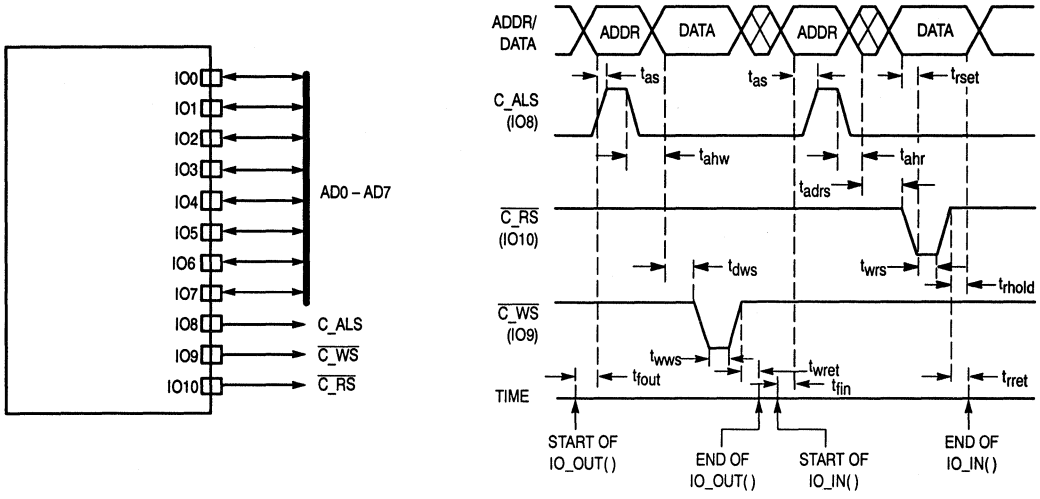
Symbol	Description	Typ @ 10 MHz
t_{fout}	Function call to output pulse	109 μs
t_{cod}	Last negative sync Clock edge to output inactive	min 550 ns max 750 ns
t_{ret}	Return from function	7 μs

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

Figure 5-42. Triggered Count Output Latency Values

5.7 MUXBUS I/O

This I/O object provides another means of performing parallel I/O data transfers between the NEURON CHIP and an attached peripheral device or processor (see Figure 5-43). Unlike the parallel I/O object, which makes use of a token-passing scheme for ensuring synchronization, the muxbus I/O enables the NEURON CHIP to essentially be in control of all read and write operations at all times. This relieves the burden of protocol handling from the attached device and results in an easier-to-use interface at the expense of data throughput capacity.



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

Figure 5-43. Muxbus I/O Object

5.8 NOTES

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

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

For on-time input, period input, dualslope, edgelog, and infrared, 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 5-4.

For one-shot output, frequency 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 5-4. 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 5-4. 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	13.1
1 (÷2) (2.5 MHz)	0.4	26.2
2 (÷4) (1.25 MHz)	0.8	52.4
3 (÷8) (625 kHz)	1.6	105
4 (÷16) (312.5 kHz)	3.2	210
5 (÷32) (156.25 kHz)	6.4	419
6 (÷64) (78.125 kHz)	12.8	839
7 (÷128) (39.06 kHz)	25.6	1,678

NOTE: This table is for 10 MHz input clock. Scale appropriately for other clock rates:
Resolution (μs) = $2^{(\text{Clock Select} + 1)}$ / Input Clock (MHz)
<Maximum Range> (ms) = 65.535 x Resolution (μs)

For pulse-width short output and pulse-count output, Table 5-5 gives the possible choices for pulse-train repetition frequencies, except that pulsecount cannot be used with clock select 0.

Table 5-5. 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:
Period (μs) = 512 x $2^{\text{Clock Select}}$ / Input Clock (MHz)
Frequency (Hz) = 1,000,000 / Period (μs)

For pulse-width long output, Table 5-6 gives the possible choices for pulse-train repetition frequencies.

Table 5-6. Timer/Counter Pulse Train Output

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

NOTE: This table is for 10 MHz input clock. Scale appropriately for other clock rates:
Period (ms) = 131.072 x $2^{\text{Clock Select}}$ / Input Clock (MHz)
Frequency (Hz) = 1,000 / Period (ms)

SECTION 6
NEURON CHIP ELECTRICAL AND MECHANICAL SPECIFICATIONS

6.1 INTRODUCTION

The MC143150 is available in two versions. The MC143150FU is designed for input clock operation up to and at 10 MHz over – 40 to + 85°C, with writes to internal EEPROM guaranteed down to – 20°C. The MC143150FU1 is a *lower cost device* designed for 5 MHz operation and below, over the full industrial temperature range of – 40 to + 85°C, including writes to EEPROM down to – 40°C. Memory interface timing is also different for the FU1 device. (See Sections 6.2.4, 6.2.5, and 6.2.6.)

When deciding what speed to operate the NEURON CHIP, the cost of the external memory will be an important consideration. There is a significant cost difference between a 200 ns memory (EPROM access time required at 5 MHz for the FU1 device) and a 90 ns memory (10 MHz access time required for the FU device) which is rated over the full industrial temperature range particularly in surface mount packages.

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

I/O response time will also double when operating at 5 MHz, and this must be considered. Proper prioritizing of "when" statements in the application program and use of timer counter objects can help to minimize latency. For nodes that require extremely fast I/O response time, if only a small number are needed in a network, using a 68HC11 or 68HC05 with their hardware interrupt capability and interfacing to the NEURON CHIP in one of the parallel modes is an option. The following electrical tables highlight the differences between the MC143150FU and FU1. Other characteristics which differ are noted.

6.2 ELECTRICAL SPECIFICATIONS

6.2.1 Absolute Maximum Ratings

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

NOTE: Writes to EEPROM are guaranteed down to – 20°C @ all frequencies for FU devices.

6.2.2 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

*Writes to EEPROM are guaranteed down to – 20°C @ all frequencies for FU devices.

6.2.3 Electrical Characteristics

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

Parameter	Symbol	Min	Typ	Max	Unit
Input Low Voltage	V_{IL}				V
IO0–IO10, D0–D7, A0–A15, CP2–CP4		—	—	0.8	
CP0, CP1 (Differential)		—	—	Programmable	
Input High Voltage	V_{IH}				V
IO0–IO10, D0–D7, A0–A15, CP2–CP4		2.0	—	—	
CP0, CP1 (Differential)		Programmable	—	—	
Low-Level Output Voltage	V_{OL}				V
Standard Outputs (I_{OL} = 1.4 mA) (Note 1)		—	—	0.4	
High Sink (IO0–IO3), $\overline{SERVICE}$, $\overline{RESET}$ (I_{OL}= 20 mA)		—	—	0.8	
High Sink (IO0–IO3), $\overline{SERVICE}$, $\overline{RESET}$ (I_{OL}= 10 mA)		—	—	0.4	
Maximum Drive (CP2, CP3) (I_{OL}= 40 mA)		—	—	1.0	
Maximum Drive (CP2, CP3) (I_{OL}= 15 mA)		—	—	0.4	
High-Level Output Voltage	V_{OH}				V
Standard Outputs (I_{OH} = –1.4 mA) (Note 1)		V_{DD} – 0.4 V	—	—	
High Sink (IO0–IO3), $\overline{SERVICE}$ (I_{OH} = – 1.4 mA)		V_{DD} – 0.4 V	—	—	
Maximum Drive (CP2, CP3) (I_{OH} = – 40 mA)		V_{DD} – 1.0 V	—	—	
Maximum Drive (CP2, CP3) (I_{OH} = – 15 mA)		V_{DD} – 0.4 V	—	—	
Hysteresis (Excluding CLK1, $\overline{RESET}$)	V_{hys}	175	—	—	mV
Input Current (Excluding pullups) (V_{SS} to V_{DD}) (Note 2)	I_{in}	– 10	—	10	µA
Pullup Source Current (V_{out} = 0 V, Output = High-Z)	I_{pu}	30	—	300	µA
NEURON 3150 CHIP					
Operating Mode Supply Current (Note 3) 10 MHz Clock		—	32	60	mA
5 MHz Clock		—	22	40	
2.5 MHz Clock		—	16	32	
1.25 MHz Clock		—	13	27	
0.625 MHz Clock		—	12	25	
Sleep Mode Supply Current (Note 3)		—	0.5	2	mA
NEURON 3120 CHIP					
Operating Mode Supply Current (Note 3) 10 MHz Clock		—	36	60	mA
5 MHz Clock		—	22	38	
2.5 MHz Clock		—	16	26	
1.25 MHz Clock		—	10	22	
0.625 MHz Clock		—	8	18	
Sleep Mode Supply Current (Note 3)		—	0.6	2	mA

NOTES:

1. 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.)
2. IO4–IO7 and $\overline{SERVICE}$ have configurable pullups. $\overline{RESET}$ has a permanent pullup.
3. Supply current measurement conditions: all outputs under no-load conditions, all inputs ≤ 0.2 V or ≥ (V_{DD} – 0.2 V), configurable pullups off, crystal oscillator clock input, differential receiver disabled. The differential receiver adds approximately 200 µA typical and 600 µA maximum when enabled. It is enabled on either of the following conditions:
   - NEURON CHIP in Operating mode **and** Comm Port in Differential mode.
   - NEURON CHIP in Sleep mode **and** Comm Port in Differential mode **and** Comm Port Wakeup not masked.

6.2.4 External Memory Bus Timing — MC143150FU, V_{DD} ± 10%
(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	7	—	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	55	—	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	—	60	ns
t_{WH}	R/$\overline{W}$ Hold Time Write	5	—	ns

NOTES:
* All values are preliminary and subject to change.
1. $t_{cyc} = 2 \cdot 1/f$, where f is the input clock frequency.
2. See Figure 6-3. 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.

6.2.5 External Memory Bus Timing — MC143150FU, V_{DD} ± 5%
(V_{DD} = 4.75 to 5.25 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	—	45	ns
t_{AH}	Address Hold Time	8	—	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	50	—	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	—	60	ns
t_{WH}	R/$\overline{W}$ Hold Time Write	5	—	ns

NOTES:
* All values are preliminary and subject to change.
1. $t_{cyc} = 2 \cdot 1/f$, where f is the input clock frequency.
2. See Figure 6-3. 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.

6.2.6 External Memory Bus Timing — MC143150FU1, V_{DD} ± 10%
(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)	400	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	—	80	ns
t_{AH}	Address Hold Time	7	—	ns
t_{RD}	Delay, $\overline{E}$ High to R/$\overline{W}$ Valid Read	—	50	ns
t_{RH}	R/$\overline{W}$ Hold Time Read	5	—	ns
t_{DSR}	Read Data Setup Time	80	—	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	—	50	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	—	100	ns
t_{WH}	R/$\overline{W}$ Hold Time Write	5	—	ns

NOTES:
* All values are preliminary and subject to change.
1. $t_{cyc} = 2 \cdot 1/f$, where f is the input clock frequency.
2. See Figure 6-3. 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.

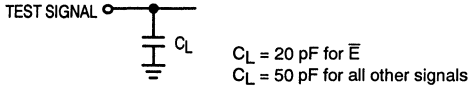
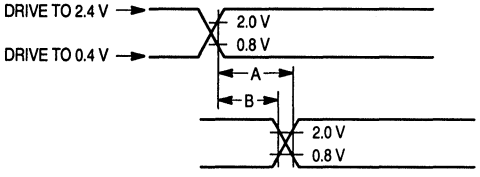
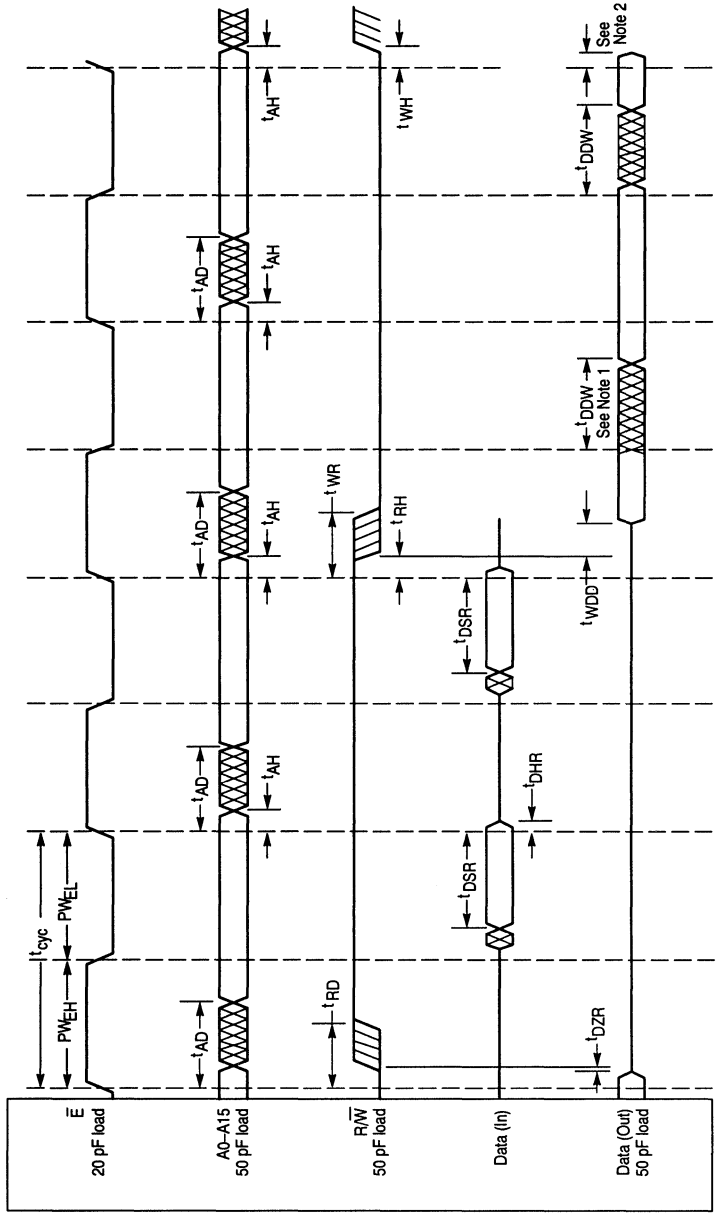


Figure 6-1. Signal Loading for Memory Interface Timing Specifications



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

Figure 6-2. Drive Levels and Test Points for Memory Interface Timing Specifications



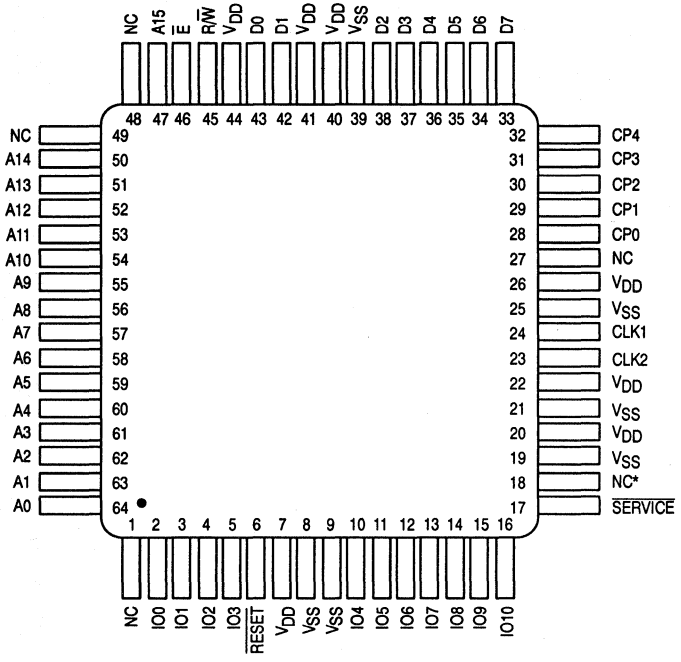
NOTES:
1. The NEURON CHIP drives the previously read data until after the falling edge of $\overline{E}$. Therefore, an external memory and the NEURON CHIP may both be driving the data lines to the same levels during this time without contention.
2. Since the data bus goes high-Z at the beginning of a read cycle, the effective data hold time is dependent on the current load on the bus. Typically this will be >>50 ns.

Figure 6-3. External Memory Interface Timing Diagram

6.3 MECHANICAL SPECIFICATIONS

6.3.1 MC143150 Pin Assignments

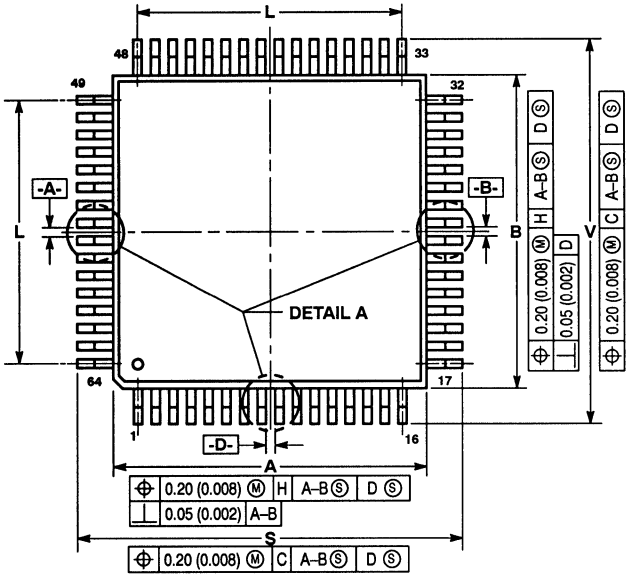
MC143150FU
64-LEAD QUAD FLAT-PACK

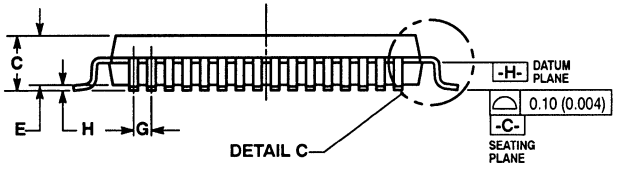


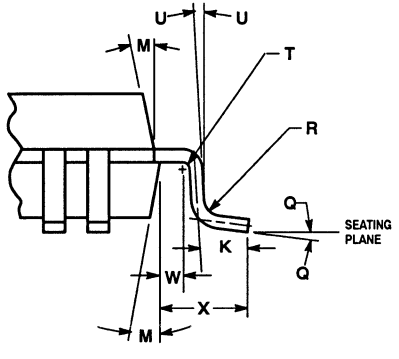
*Pin 18 must **NOT** be connected.

6.3.2 MC143150 Package Dimensions

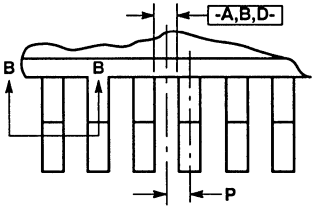
MC143150FU
PLASTIC
64-LEAD QUAD FLAT-PACK
CASE 840C-01



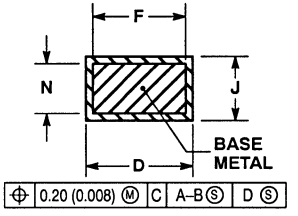




DETAIL C



DETAIL A



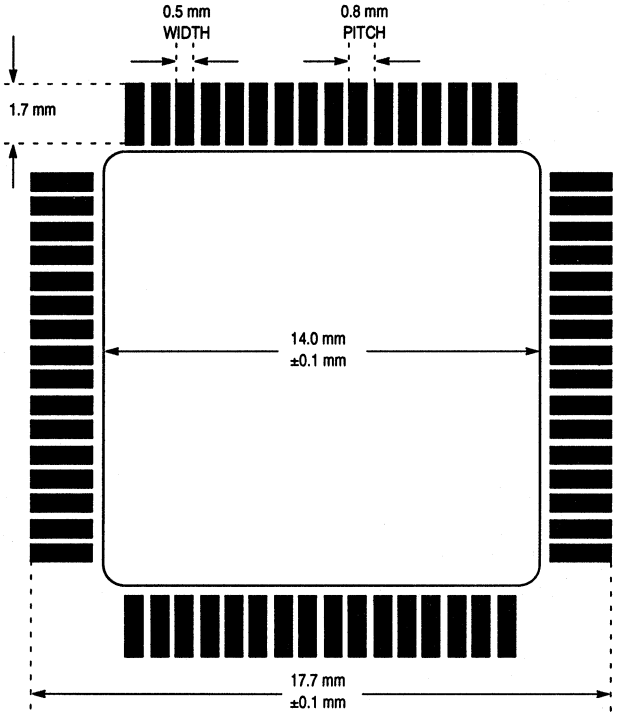
SECTION B–B

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

	MILLIMETERS		INCHES	
DIM	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
X	1.10	1.30	0.043	0.051

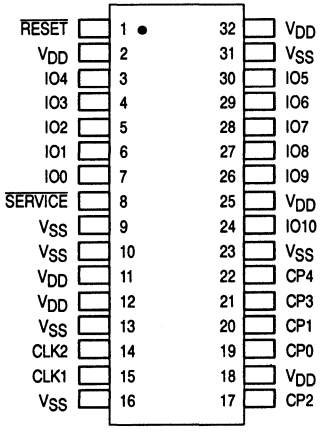
6.3.3 MC143150 Pad Layout

MC143150FU
64-LEAD QUAD FLAT PACK



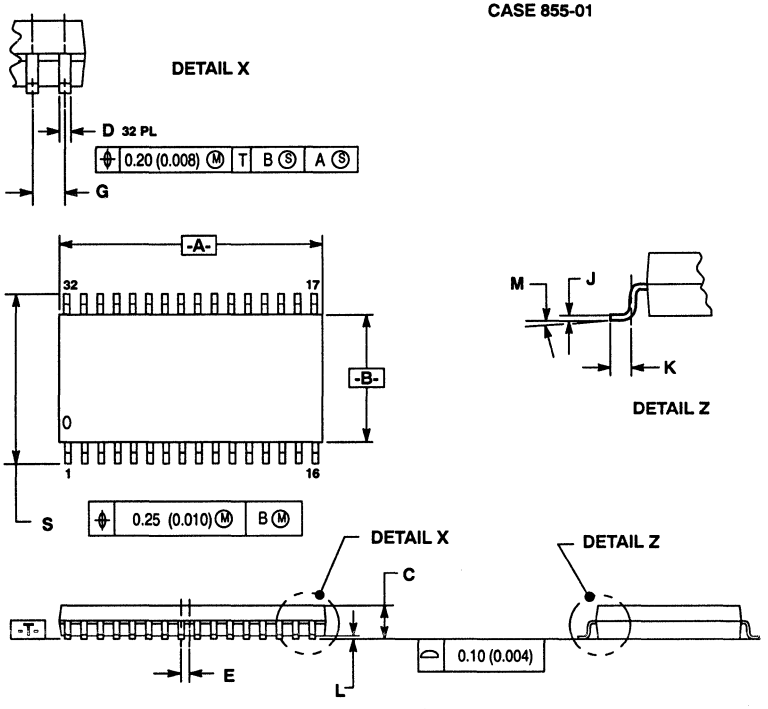
6.3.4 MC143120 Pin Assignment

MC143120DW
32-LEAD SOG



6.3.5 MC143120 Package Dimensions

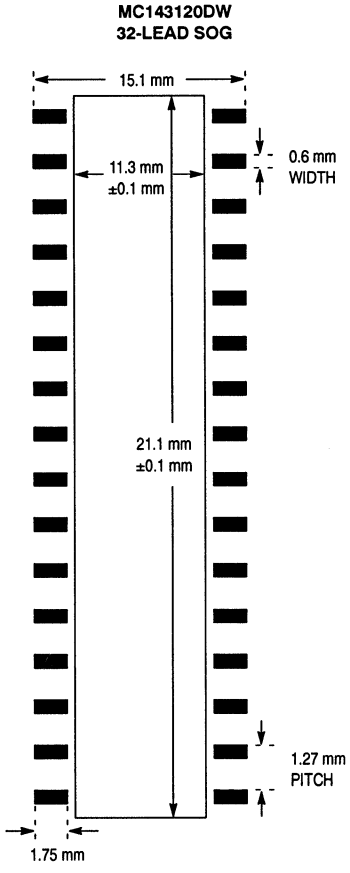
MC 143120DW
PLASTIC
32-LEAD SOG
CASE 855-01



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.

	MILLIMETERS		INCHES	
DIM	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	0.106	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

6.3.6 MC143120 Pad Layout



6.3.7 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 . This event is called *timer_expires*. The value of a timer variable is an unsigned long (0–65,535). Timers can be set to any value at any time by the application program.

7.2 NETWORK VARIABLES

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

Binding is the process of connecting network variables from different nodes together, and is typically performed by a network management device. The LONBUILDER Developer's Workbench and the LONMANAGER Applications Programming Interface (API) 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 NEURON CHIP's EEPROM, which may be updated.

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

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

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

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

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

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

A custom LONWORKS network interface can also be built using the LONBUILDER Microprocessor Interface Program (MIP). The MIP is firmware for the NEURON CHIP that transforms the NEURON CHIP into a high-performance communications processor for an attached host.

When using a LONWORKS network interface, network variables are moved to the host processor. Network variable configuration management may be performed entirely by the host (called *host selection*), or may

be split between the host and network interface (called *network interface selection*). When host selection is used, the host application can implement up to 4096 bound network variables versus the 62 bound network variables for NEURON CHIP-hosted nodes.

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

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

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

Completion status is binary. A network variable update either succeeded or it did not. Additional information about the source of failures can be derived by examining node statistics using the query status network management message.

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

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

7.3 EXPLICIT MESSAGES

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

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

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 to 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. 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
resp_out	Response to be sent to a previous request

The declarations of these structures are built in to the NEURON C compiler. They are reproduced here for reference.

```
typedef enum {
     ACKD,                          // acknowledged
     UNACKD_RPT,                    // unacknowledged repeated
     UNACKD,                        // unacknowledged
     REQUEST                        // request/response
} service_type;

struct {
     int        code;               // message code
     int        len;                // length of message data
     int        data [ ];           // message data
     boolean    authenticated;      // TRUE if message was successfully
                                    // authenticated
     service_type service;          // service type
     msg_in_addr addr;              // optional field if explicit
                                    // addressing used
     msg_in;

struct {
     boolean    priority_on;        // TRUE if message is to be sent with
                                    // priority
     msg_tag;   tag;                // destination message tag
     int        code;               // message code
     int        data [ ];           // message data
     boolean    authenticated;      // TRUE if message is to be
                                    // authenticated
     service_type service;          // service type
     msg_out_addr addr;             // optional field if explicit
                                    // addressing used
} msg_out;

struct {
     int        code;               // response message code
     int        len;                // length of response message data
     int        data [ ];           // message data
     resp_in_addr addr;             // optional field if explicit
                                    // addressing used
} resp_in;

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

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

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

typedef struct {
   unsigned domain      : 1; // domain table reference
   unsigned flex_domain : 1; // received on flexible domain
   unsigned format      : 6; // 0 = broadcast, 1 = group,
                             // 2 = subnet/node; 3 = Neuron ID
   struct {
     unsigned subnet;
     unsigned                : 1;
     unsigned node           : 7;
   } src_addr;                     //source address
   union {
     unsigned bcast_subnet;        // broadcast destination address
     unsigned group;               // group destination address
     struct {
                        unsigned subnet;
                        unsigned       : 1;
                        unsigned node  : 7;
     } snode;                 // subnet/node destination address
     struct {
                        unsigned subnet;
                        unsigned nid[ NEURON_ID_LEN ];
     } nrnid;                        // Neuron ID destination address
   } dest_addr;                      // destination address
} msg_in_addr;

typedef struct {
   unsigned domain      : 1; // domain table reference
   unsigned flex_domain : 1; // received on flexible domain
   struct {
           unsigned subnet;
           unsigned is_snode : 1; // 0 = group response,
                                  // 1 = subnet/node response
           unsigned node     : 7;
   } src_addr;                       // source address
union {

           struct {
                        unsigned subnet;
                        unsigned        :1;
                        unsigned node   :7;
     } snode;                        // Subnet/node destination address
           struct {
                        unsigned subnet;
                        unsigned        :1;
                        unsigned node   :7;
                        unsigned group;
```

```
                        unsigned        :2;
                        unsigned member :6;
      } group;                  // group destination address
    } dest_addr;                // destination address
} resp_in_addr;
```

Preemption Mode

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

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 code is executed. The user has the capability of specifying one or more *when* clauses as having priority, and the scheduler checks priority *when* clauses, 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 with 'wink' command option

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

nv_update_occurs
nv_update_fails
nv_update_succeeds

nv_update_completes
msg_arrives
msg_completes
msg_succeeds
msg_fails
resp_arrives

User-Specified Events:

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

7.5 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	Define 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_clone_domain	Write node's domain table with clone entry
update_domain	Write node's domain table
update_nv	Write node's network variable configuration table
update_config_data	Write node's configuration structure

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

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
retrieve_xcvr_status	Read transceiver status register
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
nv_in_addr	Source address of the last network variable update

7.7 MC143120 SYSTEM LIBRARY

On an MC143120, all application code is placed in on-chip EEPROM. In addition, when any of the following system capabilities are used, object code is brought in from a system library and placed in on-chip EEPROM.

Functions

access_address( )	
access_domain( )	
access_nv( )	
bcd2bin( )	
bin2bcd( )	
flush_wait( )	
go_unconfigured( )	
memcpy( )	(from msg_in.data and resp_in.data)
memcpy( )	(to resp_out.data)
memcpy( )	(length ≥ 256 bytes)
memset( )	(length ≥ 256 bytes)
muldiv( )	
muldivs( )	
nv_in_addr	
power_up( )	
retrieve_status( )	
reverse( )	
update_address( )	
update_clone_domain( )	
update_config_data( )	
update_domain( )	
update_nv( )	

I/O Objects

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

Miscellaneous

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

SECTION 8
LONTALK PROTOCOL

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

Table 8-1. LONTALK Protocol Layering

OSI Layer		Purpose	Services Provided	Processor
7	Application	Application compatibility	Standard network variable types	Application
6	Presentation	Data interpretation	Network variables, foreign frame transmission	Network
5	Session	Remote actions	Request/response, authentication, network management	Network
4	Transport	End-to-end reliability	Acknowledged and unacknowledged, unicast and multicast, authentication, common ordering, duplicate detection	Network
3	Network	Destination addressing	Addressing, routers	Network
2	Link	Media access and framing	Framing, data encoding, CRC error checking, predictive 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 and can contain up to 32,385 nodes maximum. A network may be composed of one or more channels. In order for packets to be transferred from one channel to another, a device called a router is used to connect the two channels. Routers typically consist of two NEURON CHIPS connected together via their I/O pins. Each of the NEURON CHIPS uses its own transceiver to communicate with its channel. The protocol supports such devices so that multiple media networks may be constructed, and network loading can be optimized by localizing traffic.

8.3 COMMUNICATIONS RATES

Channels may be configured for different bit rates to allow trade-offs of distance, throughput, and power consumption. The allowable bit rates are 0.6, 1.2, 2.4, 4.9, 9.8, 19.5, 39.1, 78.1, 156.3, 312.5, 625, and

1,250 kb/s. Channel throughput depends on bit rate, oscillator frequencies and accuracy, transceiver characteristics, average size of a packet, and the use of acknowledgments, priority and authentication. An average packet is in the range of 10 to 16 bytes long, depending on the length of the domain identifier, the addressing mode, and the size of the data field for a network variable update or an explicit message.

8.4 LONTALK ADDRESSING LIMITS

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

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

The third level of addressing is the node. There may be up to 127 nodes per subnet. Thus a maximum of 255x127 = 32,385 nodes may be in a single domain.

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

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

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

Nodes are addressed using one of five addressing formats:

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

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

8.5 MESSAGE SERVICES

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

ACKD, or end-to-end acknowledged service, where a message is sent to a node or group of nodes, and individual acknowledgments are expected from each receiver. If the acknowledgments are not all received, the sender times out and retries the transaction. The number of retries and 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 unacknowledged. They are:

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

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

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

8.6 AUTHENTICATION

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

When an authenticated message is sent, the receiver challenges the sender to provide authentication, using a different random challenge 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.

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

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

8.10 DATA INTERPRETATION

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

8.11 NETWORK MANAGEMENT AND DIAGNOSTIC SERVICES

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

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 MC143120, this software is in the on-chip 10K ROM. In the MC143150, this software is in an external ROM. For the MC143150, 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 MC143120, 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_clone_domain`, `update_config_data`, `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` and `resp_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 MC143150, the application image is typically programmed into an external ROM, or downloaded over the network to EEPROM memory. In the MC143120, 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.

ON-CHIP EEPROM LAYOUT

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

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

Fixed read-only data structure	0xF000
Configuration data structure	0xF024
Domain table	0xF03D
Address table	0xF03D + 15 (decimal) bytes per domain
Network variable configuration table	Address table + 15 bytes per address

The network variable fixed table, the SNVT descriptor table, and the various other parts of the application image are located by the LONBUILDER linker at addresses which depend on the memory map of the node.

For an MC143150-based node, it is possible to specify that the off-chip ROM is just 16K bytes (64 pages), which will force the whole application image into internal EEPROM.

RAM LAYOUT

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

System Base RAM. This memory contains stacks and system control variables. The amount of this fixed memory is a function of the NEURON CHIP model, i.e., 3120 and 3150. The values for this are 454 and 636 bytes respectively. The portion of this memory allocated to the application data and return stacks are 114 and 232 bytes respectively.

Application Timers. The number of application software timers times 4 bytes per timer.

Receive Transactions. The number of receive transactions times 13 bytes per transaction record.

Transmit Transactions. The number of transmit transactions times 18 bytes per transaction record. The number of transmit transactions is two if priority buffers are present, one otherwise.

Buffers. Buffers are allocated in the following order:

- Application input buffers
- Application output buffers
- Application output priority buffers
- Network input buffers
- Network output buffers
- Network output priority buffers

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

I/O Changes Array. The number of I/O changes events in the program times 3 bytes per event.

Application Data. The application data is allocated from the end of RAM downward towards the end of system RAM. Note that the firmware is not aware of where the application data resides. It relies on the linker to correctly partition the RAM. Thus, increasing system RAM requirements after linking has occurred must be done with caution. To aid such an effort, a node's interface (.XIF) file contains the amount of total RAM available for system usage.

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

A.1 FIXED READ-ONLY DATA STRUCTURE

This structure is physically located at the start of the 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    NV_processing_off    : 1;
      unsigned    two_domains          : 1;
      unsigned    explicit_addr        : 1;
      unsigned                         : 5;
      unsigned    address_count        : 4;
      unsigned                         : 4;
      unsigned                         : 4;
      unsigned    receive_trans_count  : 4;
      unsigned    app_buf_out_size     : 4;
      unsigned    app_buf_in_size      : 4;
      unsigned    net_buf_out_size     : 4;
      unsigned    net_buf_in_size      : 4;
      unsigned    net_buf_out_priority_count  : 4;
      unsigned    app_buf_out_priority_count  : 4;
      unsigned    app_buf_out_count    : 4;
      unsigned    app_buf_in_count     : 4;
      unsigned    net_buf_out_count    : 4;
      unsigned    net_buf_in_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.

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

These are two fields that specify the model of the NEURON CHIP. The encoding of the model number field is: MC143150 = 0 , MC143120 = 8.

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

This field is a pointer to the Network Variable fixed data table (see Section A.4). If there are no network variables on the node, or this node is running the Microprocessor Interface Program, then this pointer is not useful.

```
unsigned   read_write_protect: 1;      // offset 0x0A
```

This bit specifies that parts of the NEURON CHIP memory may not be read or written over the network with the Read Memory and Write Memory network management messages. The application program may set this bit with the NEURON C compiler directive `#pragma read_write_protect`. If this bit is set, only the Read-only Structure (Section A.1), the SNVT Structures (Section A.5), the Configuration Structure (Section A.6), and the application data area may be read, and only the Configuration Structure may be written. The write-protected data includes the `read_write_protect` bit itself, so that once set, the bit may not be reset over the network.

```
unsigned   nv_count          : 6;
```

This field specifies the number of network variables declared in the application program running on this node (0–62). Each element of a network variable array is counted separately. If this node is running the Microprocessor Interface Program with the network variable configuration table on the host, this field is zero.

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

This field is a pointer to the data structure that gives self-identification information for the network variables (see Section A.5). If the self-identification information is not present, this is a null (0) pointer. If the NEURON CHIP is executing the Microprocessor Interface Program, 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 nodes that conform to the LONMARK interoperability guidelines and conformance review. 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 8 bits are the manufacturer-assigned model number.

These fields are assigned as part of the LONMARK conformance review. If the program ID string is not specified by either of the compiler directives, then it contains the name of the NEURON C source file. The program ID string may be read over the network using the Query ID network management message. It is also part of the unsolicited Service Pin network management message.

```
unsigned   NV_processing_off :  1;     // offset 0x15
```

This bit specifies that network variable processing is being performed off-chip in a host-based node using the Microprocessor Interface Program (MIP).

```
unsigned    two_domains      : 1;
```

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

```
unsigned    explicit_addr    : 1;
```

This bit specifies that the node uses explicit message addressing in its application. If this bit is set, the application buffers contain an 11-byte explicit address field. This is set for any application using explicit addressing or the *nv_in_addr* structure.

```
unsigned    address_count    : 4;        // offset 0x16
```

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

```
unsigned    receive_trans_count : 4;    // offset 0x17
```

This field specifies the number of receive transactions, as defined by the NEURON C compiler directive `#pragma receive_trans_count`. The number of receive transactions is one more than the number specified in this field. Each receive transaction uses 13 bytes of RAM.

```
unsigned    app_buf_out_size: 4;        // offset 0x18
unsigned    app_buf_in_size : 4;
unsigned    net_buf_out_size: 4;        // offset 0x19
unsigned    net_buf_in_size : 4;
```

These fields specify the sizes of the application and network buffers, as defined by the corresponding NEURON C compiler directives. The fields are encoded as follows:

field value	buffer size
2	20
3	21
4	22
5	24
6	26
7	30
8	34
9	42
10	50
11	66
12	82
13	114
14	146
15	210
0	255

```
unsigned    net_buf_out_priority_count : 4;    // offset 0x1A
unsigned    app_buf_out_priority_count : 4;
unsigned    app_buf_out_count          : 4;    // offset 0x1B
unsigned    app_buf_in_count           : 4;
unsigned    net_buf_out_count          : 4;    // offset 0x1C
unsigned    net_buf_in_count           : 4;
```

These fields specify the number of application and network buffers, as defined by the corresponding NEURON C compiler directives (from 0 to 63 buffers). The fields are encoded as follows:

field value	buffer count
0	0
2	1
3	2
4	3
5	5
6	7
7	11
8	15
9	23
10	31
11	47
12	63
13	95
14	127
15	191

Note that if one of the priority output buffer counts is zero, then both of them must be zero.

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 );
void update_clone_domain( const domain_struct * domain, int index );
```

The application program may read or write any entry in this table using the access routines `access_domain ()`, `update_domain ()` and `update_clone_domain ()`. Normally, the function `update_domain ()` should be used to write into the node's domain table. The function `update_clone_domain ()` can be used for self-installation applications where it is not necessary to assign unique subnet and node IDs to each node. It will allow the node to receive messages from another node with the same subnet and node ID, but it will prevent the node from receiving any message addressed with subnet/node addressing mode in that domain. If `update_clone_domain ()` is used, the node can only receive messages addressed with NEURON ID, group or broadcast addressing modes. The domain table consists of up to two entries, each 15 bytes in length. The default number of entries is two, which may be overridden by the NEURON C compiler directive `#pragma one_domain`. The entries in this table may be written and read over the network with the Update Domain, Leave Domain, and Query Domain network management messages.

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. A broadcast address 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               : 1;
     unsigned   backlog     : 6;         // offset 0x01
     unsigned   rpt_timer   : 4;         // offset 0x02
     unsigned   retry       : 4;
     unsigned               : 4;         // offset 0x03
     unsigned   tx_timer    : 4;
     unsigned   subnet      : 8;         // offset 0x04
} bcast_struct;
```

A.3.6 Broadcast Address Field Descriptions

```
addr_type  type;
```

This field contains the value `BROADCAST (3)`.

```
unsigned   domain     : 1;
```

This field specifies the index into the domain table for this address (0 or 1).

```
unsigned   backlog    : 6;
```

This field specifies an estimate of the channel backlog that would be created by an acknowledged or request/response message broadcast using this address. It should be set to the expected number of acknowledgements or responses. For example, this might be the worst case number of nodes expected to respond to a QUERY_ID message on a channel. If this is unknown, this field can be set to zero in which case a backlog of fifteen is assumed.

```
unsigned   subnet     : 8;
```

This field specifies the subnet number (1–255) within the specified domain. The message is delivered to all nodes in this subnet. If the subnet number is 0, the message is delivered to all nodes in the domain. If Request/Response or Acknowledged service is used with a broadcast address, the transaction completes as soon as the first response or 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
     unsigned   rpt_timer; : 4;        // offset 0x02
     unsigned   retry      : 4;
     unsigned              : 4;        // offset 0x03
     unsigned   tx_timer;  : 4;
} turnaround_struct;
```

A.3.8 Turnaround Address Field Descriptions

```
addr_type  type;
```

Contains the value `UNBOUND (0)`.

```
unsigned   turnaround;
```

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

A.3.9 Declaration of NEURON ID Address Format

Note: This format is not used in the address table, but it may be used as a destination address for an explicitly addressed message.

```
typedef struct {
     addr_type  type;                  // offset 0x00
     unsigned   domain     : 1;        // offset 0x01
     unsigned              : 7;
     unsigned   rpt_timer  : 4;        // offset 0x02
     unsigned   retry      : 4;
```

```
        unsigned                : 4;        // offset 0x03
        unsigned    tx_timer    : 4;
        unsigned    subnet      : 8;        // offset 0x04
        unsigned    nid[ NEURON_ID_LEN ];   // offset 0x05
    } nrnid_struct;
```

A.3.10 NEURON ID Address Field Descriptions

```
addr_type  type;
```

This field contains the value `NEURON_ID (2)`.

```
unsigned   domain      : 1;
```

This field specifies the index into the domain table for the destination address (0 or 1). However, unique-ID addressed messages may be sent on any domain. Unconfigured nodes will receive messages addressed in unique-ID format on any domain. When this occurs, the message is said to be received on the flexible domain, and any response or acknowledgement will be sent back on the domain from which it was received, with source subnet and node identifiers of zero.

```
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 transaction retry timer is restarted when each attempt is made, and also when any acknowledgement or response (except for the last one) is received. For request/response service, the requesting node should take into account the delay necessary for the application to respond when setting the transaction timer. This is important, for example, for network management messages that write into EEPROM. The encoding of this field is specified in Table A-1.

See Section A.6.1 for a description of the non_group_timer defined in this table.

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 ROM, and is part of the application image written during node manufacture.

For a node using a LONWORKS Network Interface, the network variable fixed table, and the network variables themselves are located in the memory of the host microprocessor. The network variable configuration table may be in either the memory of the NEURON CHIP, or in the memory of the host microprocessor. In the latter case, the limit of 62 bound network variables per node increases to 4096, and the LONTALK protocol firmware on the NEURON CHIP does not process network variable update messages, but instead passes them to the host microprocessor.

Declarations from `ACCESS.H`

```
#define MAX_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. This bit is technically not configuration data; it resides in this table for efficiency.

```
unsigned   nv_selector_hi   : 6;
unsigned   nv_selector_lo   : 8;
```

These two fields form a 14-bit network variable selector in the range 0 –0x3FFF. 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 as follows:

- a fixed size SNVT structure
- a table containing a self-identification descriptor for each network variable
- a self-documentation string for the node
- self-documentation information for network variables

This information forms part of the application image written during node manufacture. If the NEURON C compiler directive `#pragma disable_snvt_si` is specified in the application program, none of this information is present. In an MC143150, these tables are normally located in read-only memory if space is available. However, if the NEURON C compiler directives `#pragma snvt_si_eecode` or `#pragma snvt_si_ramcode` are specified, the SNVT structures are compiled into EEPROM or non-volatile RAM respectively,where they may be modified by a network management tool.

Declarations from `ACCESS.H`

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

If the version number in the previous byte is zero, this byte is not present. If the version number is one, this byte is the most significant byte of the number of network variables declared in this node.

```
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 it is a network variable array element. Each array element has its own extension record containing the name of the array, a left square bracket, one 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.

```
unsigned   nc : 1;
```

If this bit is set, the extension record contains a 16-bit count of the number of network variables of this type (version 1 format only). This is used to define network variable arrays. If this bit is clear, one variable of this type is defined by this record.

A.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`. The Media Access Control processor reads the channel parameters (offsets 0x08 through 0x17) from the configuration structure when the node is reset and initializes the transceiver. Therefore, after changing any of the channel parameters, the node should be reset for the changes to take effect.

A.6.1 Configuration Structure Field Descriptions

```
unsigned long channel_id;
```

This field specifies the ID of the channel that this node is assigned. In the 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 ];
```

This 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, it is defined in the node specification screen.

```
unsigned    comm_clock          : 5;
```

For direct mode transceivers, this field specifies the ratio between the NEURON CHIP input clock oscillator frequency and the transceiver bit rate. For special purpose mode transceivers, it specifies the rate of the bit clock between the NEURON CHIP and the transceiver. In the development environment, the transceiver bit rate is specified in the channel screen. Table A-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

		input_clock				
comm_clock	ratio	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	2.4
6	512:1	19.5	9.8	4.9	2.4	1.2
7	1,024:1	9.8	4.9	2.4	1.2	0.6

```
unsigned      input_clock          : 3;
```

This field specifies the NEURON CHIP input clock (oscillator frequency). In the 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, it 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 ];
```

The following five fields specify the raw transceiver parameters used by the media Access Control processor. In the development environment, these parameters are specified as the Raw Data in the channel screen. The values in these fields are the repetition counts for software delay loops in the firmware that control the timing of the media access algorithm. In the following descriptions, the timing formula are given in terms of 'x', the value in the control field, which may be 0 .. 255 unless otherwise specified. A NEURON CHIP processor cycle is 0.6 μs at 10 MHz input clock, 1.2 μs at 5 MHz and so on. See Figure 3-10 for a description of the timing of packet transmission. The fields are the following:

`preamble_length` This field determines the length of the preamble for direct mode. Time = (209 .. 223) + 32 x cycles (x = 1 .. 253). For special purpose mode transceivers, the value should be zero.

`packet_cycle`	This field determines the packet cycle duration for counting down the backlog. Time = 1675 x cycles for direct mode, 1794 x cycles for special purpose mode.
`beta2_control`	This field determines the beta2 slot width. Time = 40 + 20 x cycles.
`xmit_interpacket`	This field determines the interpacket padding after transmitting. Time = 41 x cycles for $x < 128$, 145 $(x - 128)$ for $x \geq 128$.
`recv_interpacket`	This field determines the interpacket padding after transmitting. Time = 41 x cycles for $x < 128$, 145 $(x - 128)$ for $x \geq 128$.

```
unsigned    node_priority;
```

This field specifies the priority slot used by the node when sending priority messages on the channel (1–255). It should not be greater than the number of priority slots on the channel. If the node has no priority slot allocated, this is 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 tool. However, network management messages received when the node is unconfigured cannot be authenticated. Before setting a node's state to configured, a network management tool should ensure that the network management authentication bit is set to the desired state. In the LONBUILDER environment, network management authentication is defined in the node specification screen.

```
unsigned    preemption_timeout  : 3;
```

This field specifies the maximum time the node will wait for a free buffer in preemption mode. A preemption mode timeout logs an error and resets the chip. 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 3-8) 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 3-7) for details of the available hysteresis filter settings.

```
unsigned    cd_to_end_packet    : 6;
```

This field controls how close to the end of a packet the collision detect signal is checked in a transmitting NEURON CHIP. It is set as a function of the bit rate and the node's clock rate.

```
unsigned    cd_tail             : 1;
```

This bit specifies that collisions are to be detected at the end of the transmitted packet following the code violation.

```
unsigned    cd_preamble         : 1;
```

This bit specifies that collisions are to be detected during the preamble at the beginning of the transmitted packet. When specified, the packet is terminated at the end of the preamble if a collision is detected during the preamble.

APPENDIX B
NETWORK MANAGEMENT AND DIAGNOSTIC SERVICES

In addition to application message services, the LONTALK protocol provides network management services for installation and configuration of nodes, downloading of software, and diagnosis of the network. Message codes used by the LONTALK protocol are as follows:

Message Type	Hexadecimal Message Codes
Application Messages	0x00 – 0x3E
Foreign Messages	0x40 – 0x4E
Network Diagnostics Messages	0x50 – 0x5F
Network Management Messages	0x60 – 0x73
Router Configuration Messages	0x74 – 0x7E
Service Pin Message	0x7F
Network Variable Messages	0x80 – 0xFF

Response codes used by the LONTALK protocol are as follows:

Response Type	Hexadecimal Message Codes
Application Responses	0x00 – 0x3E
Response if node is off-line	0x3F
Foreign Responses	0x40 – 0x4E
Response if node is off-line	0x4F
Network Diagnostic Success	0x31 – 0x3F
Network Diagnostic Failure	0x11 – 0x1F
Network Management Success	0x21 – 0x33
Network Management Failure	0x01 – 0x13
Router Configuration Success	0x34 – 0x3E
Router Configuration Failure	0x14 – 0x1E
Network Variable Poll Responses	0x80 – 0xFF

Notice that response codes are not unique. They must be interpreted based on the original request.

Section 7.3 describes the use of application messages. A foreign message is a packet of another protocol embedded in the LONTALK protocol packet. The application program allocates space for foreign message and response codes.

Network Diagnostic Messages	Request Code	Success Response	Failed Response
Query Status	0x51	0x31	0x11
Proxy Command	0x52	0x32	0x12
Clear Status	0x53	0x33	0x13
Query XCVR Status	0x54	0x34	0x14

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.

For NEURON C programs, these message structures are defined in the include file `NETMGMT.H`.

In the following descriptions of the network management messages, the data structures named `NM_xxx_request` specify the data field of the outgoing message (following the code field). Similarly, the data structures named `NM_xxx_response` specify the data field of the corresponding response. Network management messages are sent like any other explicit message, either from a host microprocessor or from a NEURON CHIP. The following example shows how a NEURON C program running on a NEURON CHIP could use the Read Memory network management message (see 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.

Sending Network Management or Diagnostic Messages

Most NM/ND commands are delivered using the request-response service type. A few are limited to acknowledged service type. Throughout the messages descriptions that follow, request-response is assumed unless otherwise mentioned.

When the node is configured for network management authentication, most NM/ND transactions must be authenticated in order to take effect. However, if a node is not in the configured state, the network management authentication bit is ignored. Before setting a node's state to configured, the network manager should

ensure that the network management authentication bit is set to the desired state. Network management messages that do not require authentication to be executed are so noted.

The transmit transaction timer value of the client node must be extended to handle the lengthy delays involved with any command that alters EEPROM. When NEURON ID addressing is used, the node that receives the NM or ND message automatically extends the non-group receive transaction timer to about 8 seconds. This allows the non-group receive transaction timer to be tuned for normal application traffic without concerns for lengthy network management transactions.

Addressing

NEURON ID addressed messages are received regardless of the domain in which they are sent. Unconfigured nodes will also accept any subnet or domain-wide broadcast regardless of the domain. In either of these cases, acknowledgements and responses are returned on the domain in which the message was received with a source subnet/node pair of 0/0. Messages received in a domain in which the node is not a member (either because the node is unconfigured or is simply not in the domain) are termed as being received on a flexible domain. Some commands are not permitted under these circumstances and are noted below.

An advantage of using NEURON ID addressing for network management commands is that if a node were to accidentally go unconfigured (e.g., due to a checksum error resulting from a power cycle while changing configuration), the network manager would not lose its ability to communicate with the node.

However, a disadvantage of using NEURON ID addressing is that the extended receive transaction timeout could lead to subsequent false detection of duplicates. Therefore, it is recommended that, if possible, subnet/node addressing be used for network management activity. NEURON ID addressing should be used only for communicating with nodes that are not in the configured state.

Configuration Changes

The paradigm for making configuration changes is as follows:

1. Alter the node state or condition (optional).
2. Perform t he change or changes. Most messages automatically update the configuration checksum. The exception is memory writes to the configuration data structure.
3. Update the configuration checksum if necessary.
4. Return to step 2 if more needs to be done.
5. Restore the node state or condition if changed in step 1.
6. Reset the node if communication parameter changes were made. Communication parameters are copied from on-chip EEPROM to RAM at node reset, so that the media access control processor can access them even during EEPROM writes.

Step 1 typically involves taking a node off-line (with step 5 doing an on-line). This is to ensure that the application is not accessing configuration addressing information as it is changing. It may be acceptable to eliminate this step in some circumstances.

In addition to going off-line, the actual node state may be changed to unconfigured or hard-off-line (with step 5 restoring it to configured). This has the advantage that, were the node to reset during the update, it would come up with the application not running. A disadvantage of making a state change is that the node state is considered to be part of the application. If it is corrupted (e.g., due to a power cycle while the state is being changed), the node will come up applicationless. If the network manager does not have the application available to reload, this can be catastrophic for the node. It is also important to note that the

node should not leave the applicationless state as a result of reconfiguration alone. If the node is initially applicationless, setting it to configured will probably cause it to crash.

Application Downloading

The paradigm for downloading an application is as follows:

1. Take the node off-line.
2. Alter the node state to applicationless.
3. If node went bypass off-line (see B.1.6) (in step 1), reset the node.
4. Perform a sequence of write memory commands to load the application. Writes should be limited to 11 bytes and the checksum should not be computed after each write. The order and contents of the writes should be determined by the contents of a .NXE load file generated by the LONBUILDER export utility, with two exceptions. If the first record has a length of one byte, then it should not be written until after all the other records have been written. This is because it contains the read/write protect bit which could prevent further downloading if it is set. Also, if a record has a data count of zero, then the node should be reset at this point.
5. If no record with a data count of zero was encountered in step 4, reset the node in order to cause the RAM to be partitioned according to the new application's requirements.
6. Compute the application checksum.
7. Enter the unconfigured state.

Before the node may be put into the configured state, the network manager should make sure that the domain table, address table, and network variable configuration table have known states. Note that after loading an application followed by loading of the configuration, a node comes up in the off-line condition. To get the application running, you should initiate an on-line request.

Bypass off-line is defined as the application checking for "offline" events directly and invoking "offline_confirm" to effect the off-line condition. Normally, going off-line is handled by the scheduler.

Note that node resets can take quite a while. The slower a node's input clock, the longer the reset. The more off-chip EEPROM or RAM, the longer the reset. Durations up to 18 seconds are possible in the worst case. See Section 3.6.5 for details of the reset process.

Node Resets or Power Cycles: If a node resets while a network management command is in progress, the reset will likely manifest itself as either a communication problem or a transaction failure. When EEPROM writes are involved, there is a significant probability that a location being modified at the time of the reset will become corrupted (most likely with the erase pattern of 0xFF).

Read/Write Protect Violations: If a node is read/write protected, attempts to write to the application code area are denied. The client can verify that a write memory attempt failed for this reason by reading the `read_write_protect` field of the `read_only_data` structure.

B.1 NETWORK MANAGEMENT MESSAGES

These messages are described in six main groups: node identification messages, domain table messages, address table messages, network variable-related messages, memory-related messages, and special-purpose messages.

B.1.1 Node Identification Messages

Query ID (Request/Response Only)

This message requests selected nodes to respond with a message containing their 48-bit unique ID and program ID. This message is normally broadcast during network installation to find specific nodes in the domain. It can be used to find unconfigured nodes or explicitly selected nodes, or nodes with specified memory contents at a specified address. This address may be specified 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
        absolute            = 0,
        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 nodes that are both selected and unconfigured. A node may be selected with the Respond to Query message. Optional fields may be present in the message which specify an address mode, a byte count and an array of bytes which must match a part of the destination node's memory. For interoperability, the length of the matched region can be up to 11 bytes long. The matched region may be in the read-only structure, the configuration structure, the SNVT structure or the application data area. The address mode specifies whether the address of the matched memory is an absolute address in the memory space of the NEURON CHIP, an address relative to the read-only data structure (see Section A.1), or an address relative to the configuration data structure (see Section A.6). If the memory matching feature is not required, the optional fields must not be present in the message; the length of the data part of the message must be one. Setting the count field to zero does not disable the memory matching feature. The response message contains the 6-byte NEURON ID and the 8-byte program ID. Authentication is never used with this message.

Respond to Query

This message explicitly selects or deselects a node to respond to a Query ID message. It can be used to determine network topology. Resetting the node clears the selection.

Message declaration:

```
typedef enum {
    disable= 0,
     enable= 1,
} NM_respond_to_query_request;
```

The request message consists of a single byte specifying whether the node should be selected or deselected.

Service Pin Message (Unsolicited)

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

Message declaration:

```
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 checksum. This assigns a domain, subnet and node identifier to the node, as well as an authentication key for that domain. For security reasons, the authentication key should not be transmitted on an open network where it is possible for others to tap in the network and learn the authenticaiton key value. The node does not enter the configured state until a Set Node Mode message is sent to change its state to configured. The Update Domain message is honored even if the node is read/write protected. For a description of the domain table, see A.2. If the Update Domain message is received by a node on the domain that is being updated, the response is returned in the new domain. A node that receives this message can take up to 330 ms to execute the function.

Message declaration:

```
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 on an open or public network. This message is honored even if the node is read/write protected. For a description of the domain table, see A.2. Senders of this message should be especially careful that the message is not received more than once, since its function is incremental. A node that receives this message can take up to 150 ms to execute this function.

Message declaration:

```
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. A node that receives this message can take up to 130 ms to execute the function.

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)

A NEURON CHIP can be in one of four states. These states are maintained in EEPROM and are as follows:

Node State	State Code	Service LED
Applicationless and Unconfigured	3	On
Unconfigured (but with an Application)	2	Flashing
Configured, Hard Off-Line	6	Off
Configured, On-Line	4	Off
Configured, Soft Off-Line	12	Off

Applicationless and Unconfigured (3): No application is loaded yet, the application is in the process of being loaded,or the application is deemed corrupted due to application checksum error or signature inconsistency. The application does not run in this state. The service LED is steadily on in this state.

Unconfigured (2): The application is loaded but the configuration either is not loaded, is being reloaded, or is deemed corrupted due to configuration checksum error. A program can make itself

unconfigured by calling the `go_unconfigured ( )` function. The service LED flashes at a one second rate in this state.

Configured, Hard Off-Line (6): The application is loaded but not running. The configuration is considered valid in this state; the network management authentication bit is honored. The service LED is off in this state.

Configured (4): Normal Node State. The application is running and the configuration is considered valid. This is the only state in which messages addressed to the application are received. In all other states, they are discarded. The service LED is off in this state.

The configured state has an additional modifier, which is the on-line/off-line condition. This condition is **not** maintained in EEPROM. The states and on-line/off-line condition are controlled via different mechanisms. However, they are reported together in the Query Status network management message. The configured/off-line condition is also known as soft off-line.

A node that is in the soft off-line state will go on-line when it is reset. The hard off-line state is preserved across a reset. When the application is of either type off-line, the scheduler is disabled. When soft off-line, polling a network variable will return null data, but incoming network variable updates will be processed normally, except that the `nv_update_occurs` events will be lost. In all states other than configured, no response is returned to network variable polls and incoming network variable updates are discarded.

If 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 must not be delivered with request/response service. There will be no response to a *Reset* message, since the node is reset immediately. Changing the state of a node recomputes the configuration checksum, which takes some time, so verification of state changes should always be made with the Query Status message.

Message declaration:

```
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 has two formats. If the message is sent with no data, it causes the receiving node to execute the *wink* clause in the application program (if any). This may be used to identify nodes visually or audibly if it

is more convenient than grounding the service pin. Wink messages may not be delivered with request/response service.

The second format of this message is used only with host-based nodes, that is, nodes implemented with a LONWORKS network interface on a host processor. This format is needed when installing application nodes with multiple network interfaces attached. When installing a host-based node, depressing a service pin results in a single service pin message being generated. A network management toll can send this command to interrogate the node about any additional network interfaces that might be attached.

The first byte of the host-node format specifies a subcommand, which may be either a wink message or a request to send identifying information from the host. The `wink` subcommand may not be delivered with request/response service. The `send_id_info` subcommand should be delivered with request/response service, and the following byte in the message specifies a network interface number to support hosts that may have multiple network interfaces.

For host-based nodes, all forms of this message are passed to the host application, where they should be handled appropriately, either by executing a wink function or responding with the NEURON ID and program ID for the specified network interface. The first byte of the response should be zero if the specified network interface is functional, non-zero otherwise.

Message declaration from `NETMGMT.H`:

```
typedef struct {                    // used for host-based nodes only
    enum {
        WINK            = 0,
        SEND_ID_INFO    = 1,
    } subcommand;
    unsigned network_interface; // present if subcommand = SEND_ID_INFO
} NM_wink_request;

typedef struct {                    // response to SEND_ID_INFO from a host
                                       node
    boolean interface_down;
    unsigned neuron_id[ NEURON_ID_LEN ];
    unsigned id_string[ ID_STR_LEN ];
} NM_wink_response;
```

B.2 NETWORK DIAGNOSTIC MESSAGES

Query Status (Request/Response Only)

This message retrieves the network error statistics accumulators, the cause of the last reset, the state of the node, and the last 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. This 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. This may be due to an application program being too slow to process incoming packets, insufficient application buffers, or excess traffic on the channel. If the incoming message is too large for the application buffer, an error is logged, but the lost message count is not incremented.

Missed messages – The number of times that an incoming packet was discarded because there was no network buffer available. This may be due to excess traffic on the channel, insufficient network buffers, or the network buffers not being large enough to accept all packets on the channel, whether or not addressed to this node.

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

Power-up reset	`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 reach of the network management node. The response is also relayed back to the original sender.

Message declaration:

```
typedef struct {
    enum {
        query_unconfigured    = 0,
        status_request        = 1,
        xcvr_status           = 2,
    } sub_command;
    unsigned type;                          // addr_type or (0x80 |group_size)
    unsigned                  : 1;
    unsigned member_or_node   : 7;
    unsigned rpt_timer        : 4;
    unsigned retry            : 4;
    unsigned tx_timer         : 4;
    unsigned group_or_subnet;
    unsigned neuron_id[ 6 ];              // for type = 2
} ND_proxy_request;
```

The request message contains a byte specifying which message is to be delivered by proxy, followed by a destination address in the format of a `msg_out_addr` declared in `\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 that have the same structure as network variable messages.

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

Normally, network variables are updated across the network using implicit addressing. When the application program on the source node updates the value of a bound output (non-polled) network variable, the NEURON CHIP firmware automatically builds an outgoing network variable update message using the information from the network variable configuration table and the address table. The information in these two tables is created as a result of the binding process. In certain applications, it is desirable to explicitly address a network variable update, rather than have the address implicitly taken from the network variable configuration and address tables of the source node. For example, if a single node wishes to send network variable updates to more than 15 different destination addresses (single nodes or groups), then it can use explicit addressing to overcome the limit of 15 address table entries on a node.

In this case, the source node can create an explicit message that is functionally identical to a network variable update message. (See Figure B-1.) The code field of this message contains the most significant 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.

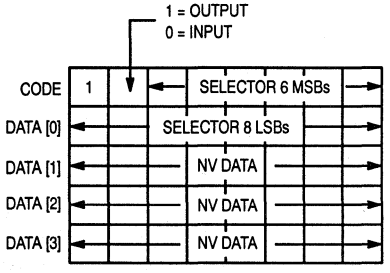


Figure B-1. Network Variable Message Structure

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 NEURON C `nv_in_addr` built-in variable can be used (see Section 7.2). Nodes using a LONWORKS network interface can optionally receive the network variable update on the host processor.

When a network variable update is addressed using group (multicast) addressing with acknowledged service, all members of the group acknowledge the update message. Those members of the group that have input network variables with a matching selector will update those variables as a result of receiving the message, and generate an `nv_update_occurs` application event. Those members of the group that have an output network variable with a matching selector, or no network variable with a matching selector, will not generate any application event, even though they acknowledge the update message. If all the acknowledgements are successfully received by the sending node, an `nv_update_succeeds` event is generated. If one or more acknowledgements are not received after the configured number of retries, an `nv_update_fails` event is generated.

Network Variable Poll (Request/Response Only)

Normally, network variables are polled across the network using implicit addressing. When the application program on the source node issues a `poll ( )` request to a bound input network variable, the NEURON CHIP firmware automatically builds an outgoing network variable request message using the information from the network variable configuration table and the address table. The information in these two tables is created as a result of the binding process. In certain applications, it is desirable to explicitly address a network variable poll, rather than have the address implicitly taken from the network variable configuration and address tables of the source node. For example, if a single node wishes to poll network variables on more than 15 different destination nodes (single nodes or groups), then it can use explicit addressing to overcome the limit of 15 address table entries on a node. It can use a single input network variable to receive an unlimited number of responses to polls of any given data type.

In this case, the source node can create an explicit message that is functionally identical to a network variable poll message. The response to the poll will be processed by the network processor, and 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 Section B.1.4. Alternatively, a node using a LONWORKS network interface can handle the response to the poll explicitly on the host processor if the MIP is configured with network variable processing off.

The code field of the Network Variable Poll request message contains the most significant 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.

When a network variable poll is addressed using group (multicast) addressing with acknowledged service, all members of the group acknowledge the poll request message. Those members of the group that have output network variables with a matching selector will respond with a message containing the value of the variable. These responses will generate `nv_update_occurs` events on the polling node. Those members of the group that have an input network variable with a matching selector, or no network variable with a matching selector, or are off-line, will generate a response containing no data. The generation of these responses requires the participation of the application processor in the polled node, and occurs at the end of the currently executing critical section. This should be taken into account when designing the application code for a node whose network variables may be polled, and when configuring the transaction timer for the poll message. If all the responses are successfully received by the polling node, an `nv_update_succeeds` event is generated. If one or more responses is not received after the configured number of retries, an `nv_update_fails` event is generated.

APPENDIX C
EXTERNAL MEMORY INTERFACING

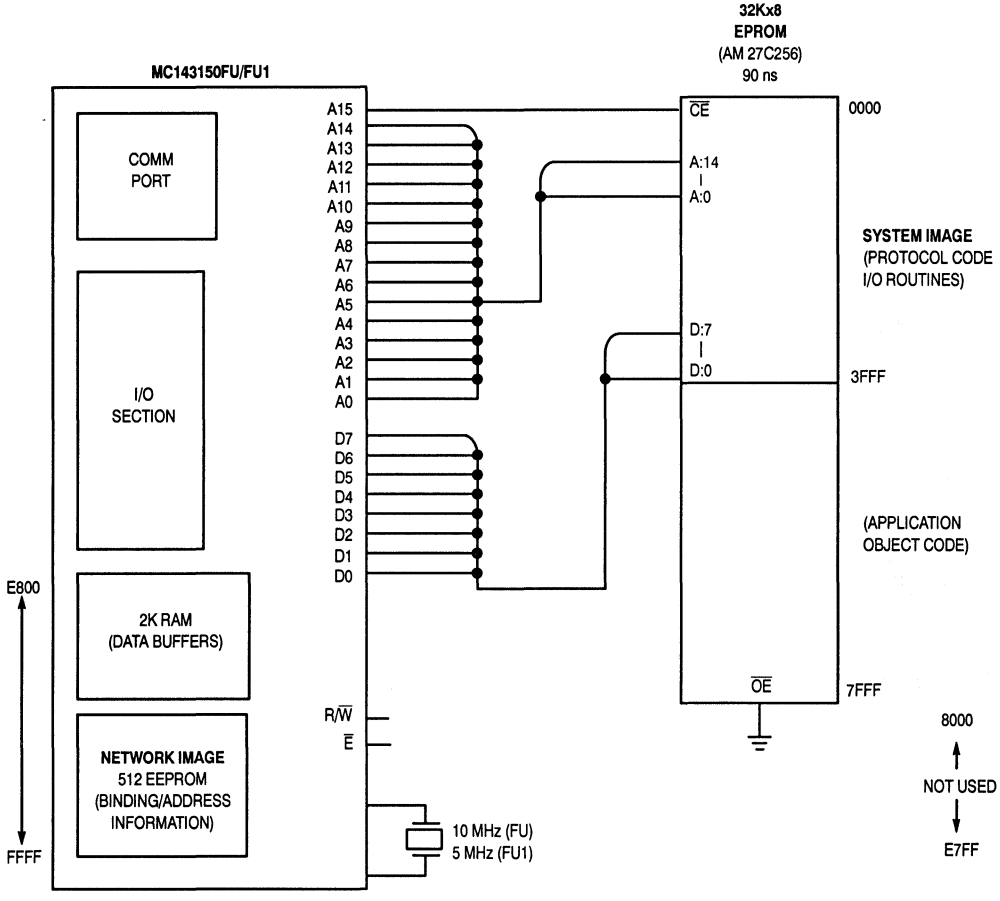


Figure C-1. General EPROM Memory Interface

For systems in which the supply current is limited to less than 300 mA or the reset line may be held low for extended periods of time the followng circuit is recommended. The delay from output enable ($\overline{OE}$) going low to when data is valid for the memory must be ≤ 45 ns.

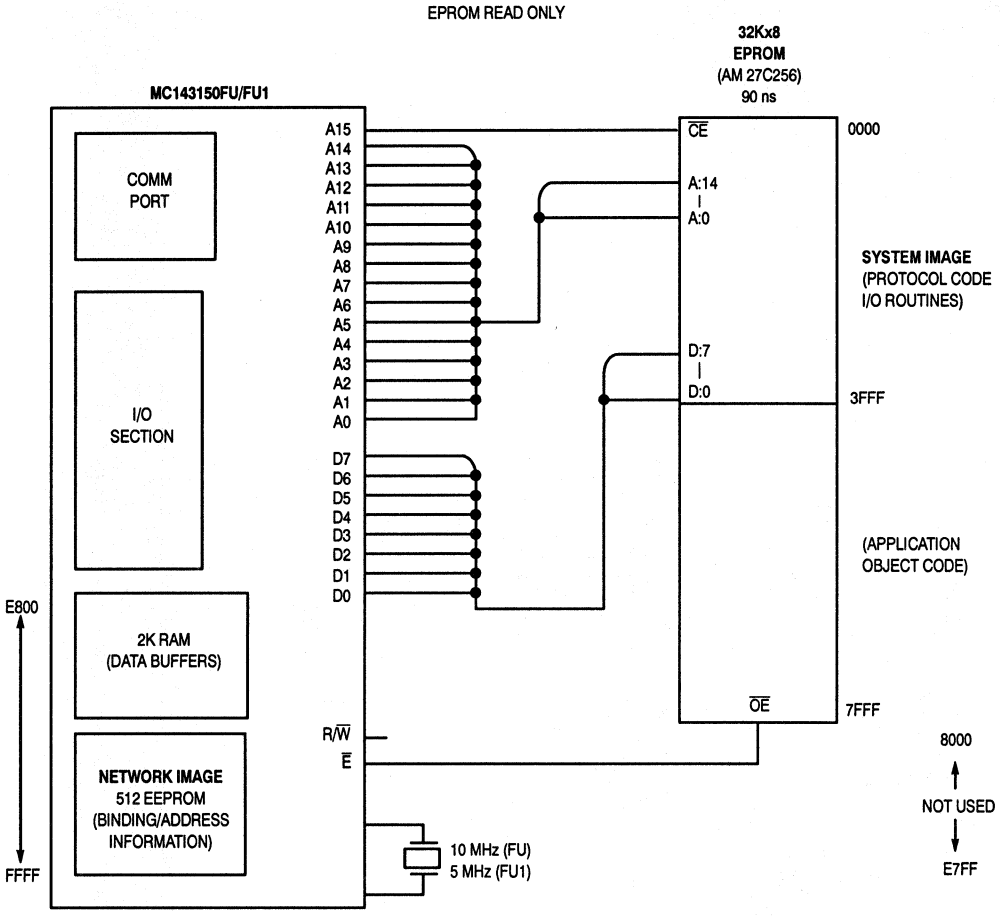


Figure C-2. Low Supply Current EPROM Memory Interface

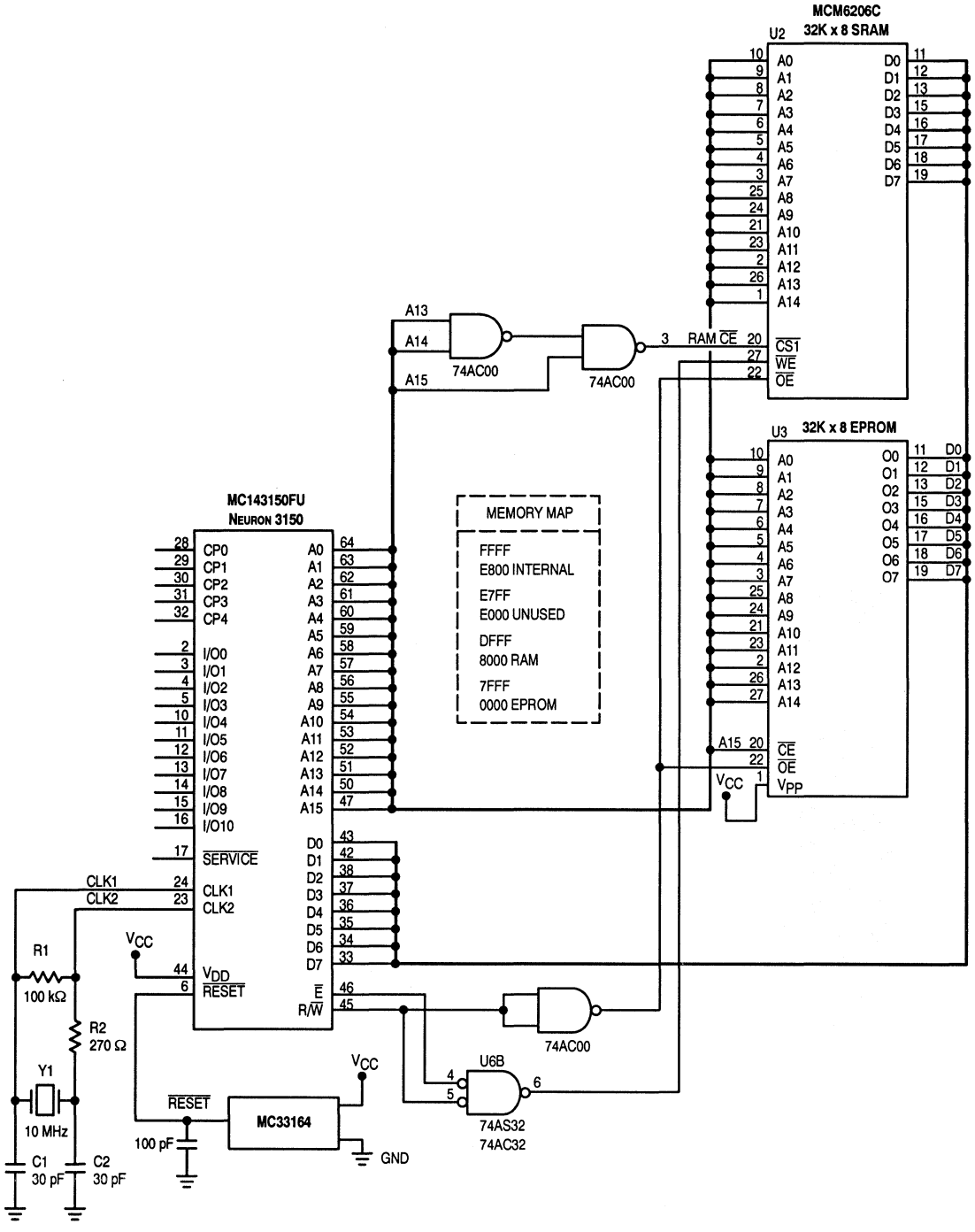


Figure C-3. General NEURON 3150 CHIP External Memory Interface with 32K-Byte EPROM and 32K-Byte RAM

For systems in which the supply current is limited to less than 300 mA or the reset line may be held low for extended periods of time the followng circuit is recommended. The delay from output enable ($\overline{OE}$) going low to when data is valid for the memory must be ≤ 45 ns.

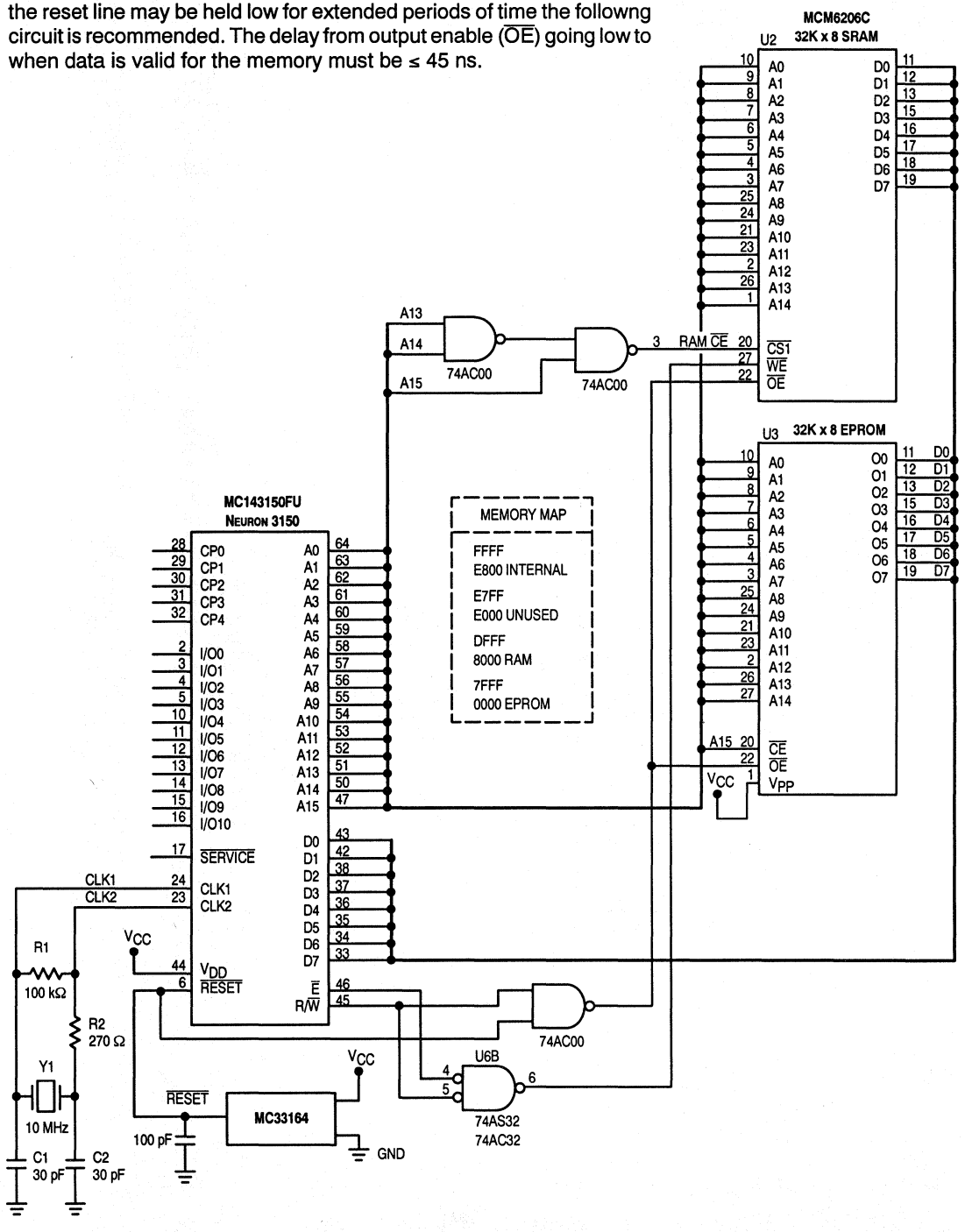


Figure C-4. Low Supply Current NEURON 3150 CHIP External Memory Interface with 32K-Byte EPROM and 32K-Byte RAM

APPENDIX D
DESIGN AND HANDLING GUIDELINES

D.1 APPLICATION CONSIDERATIONS

D.1.1 Termination of Unused Pins

Because the NEURON CHIP is a CMOS device, unused input pins *including undeclared/unconnected I/O pins configured as inputs or three-state* must be terminated to assure proper operation and reliability. Figure D-1 shows a CMOS inverter representative of circuitry found on CMOS input pins. When the input is logic zero, the P-channel transistor is on (conducts), and the N-channel transistor is off. When the input is a logic 1, the P-channel transistor is off, and the N-channel transistor is on. These transistors are linear devices with relatively broad switch points. As the input transitions through the mid-supply region, there is a duration of time when both transistors are conducting. With fast rise time digital signals at the input, this duration is very short. Once the inverter is out of the linear region (it has switched high or low) there is very little current flow. This effect is the reason that the overall current drain of a CMOS device is directly proportional to the switching speed. Almost all the current consumption is by transistors passing through the linear region and charging and discharging of internal capacitances. If a pin is configured as an input or three-state, then the input can oscillate due to supply noise or float to the mid-supply region, resulting in higher current consumption. Current design techniques have made latchup due to a floating input unlikely, but it is good design practice to terminate unused I/O pins that are not configured (high-impedance) or are configured as inputs. On the NEURON CHIP the only pin other than the I/O pins that could be configured as an unterminated input is the Service pin if the optional pullup is disabled. If the optional pullup devices are disabled on IO4–IO7, then termination is necessary for those pins.

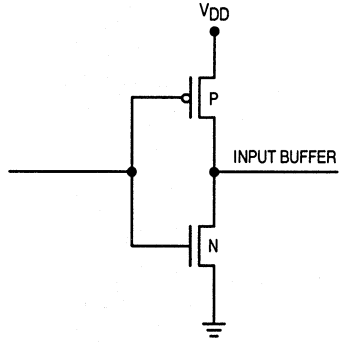


Figure D-1. CMOS Inverter

The best method to terminate unused I/O pins is with an individual pullup or pulldown resistor for each unused pin. Unused input pins can be connected to each other and then to a common termination point. This cost/space effective method has the disadvantage of not allowing individual pin configuration later and

the possibility of contention in the event pins are declared as outputs. Individual unused I/O pins may be connected directly to V_{SS} or V_{DD}, but this is not recommended in case of software error and the possibility of output declaration to an opposing state. Unused pins may be declared as outputs, but this consumes application code space which is particularly valuable in 3120 applications. A pin capable of being configured as an output should never be connected to another such pin or directly to V_{SS} or V_{DD}.

D.1.2 Avoidance of Damaging Conditions

All integrated circuit devices can be damaged or destroyed by exceeding specified voltage and environmental limits. These limits are conservative to ensure reliable operation within the conditions specified.

Most potentially destructive ac waveforms fall into one of two categories. One type is high voltage (10 – 25 kV) low energy spikes usually under 100 ns in duration due to ESD discharge. ESD modeling has shown that the human body can generate and discharge electrostatic voltages of up to 12 kV. The second type is lower voltage higher energy transients that can last for several hundred microseconds or more and can be caused by capacitive coupling of lightning or inductive load sources. Different protection devices must be implemented depending on what is anticipated in the operating environment. Failure modes can be quantified and protective precautions taken to avoid product malfunction. This may be PC board layout related or may involve the use of external protection devices. All pins on the NEURON CHIP have internal diode protection that will protect ESD type transients up to 4 kV. External protection is required in products subject to human contact or where interfaces to other equipment may be encountered.

Many factors including ambient temperature and semiconductor lot-to-lot processing variations will influence the effect of illegal conditions on the NEURON CHIP. The V_{SS} pins are internally connected to the substrate of the silicon die and are the reference point for all voltages. The NEURON CHIP is guaranteed to function for a V_{DD} connected to the positive supply pin(s) equal to 5 V ±10%. In limited temperature range environments the device may operate over a wider V_{DD} with timing, drive, and other specifications not met. it can operate with V_{DD} up to 7 V without damage, but timing and drive levels will differ from those specified. There may also be some adverse effects on gate oxides from long-term exposures to V_{DD} equal to or greater than 7 V. Some batteries will output 7 V, gradually decaying to 5 V or less with time and usage.

Zap and latchup refer to two damage mechanisms resident in CMOS ICs. Zap refers to damage caused by very-high-voltage, static-electricity exposure. This damage usually appears as breakdown of the relatively thin oxide layers that causes leakage or shorts. Often secondary damage occurs after an initial zap failure causes a short.

Latchup refers to a usually catastrophic condition caused by turning on an unintentional, bipolar, silicon-controlled rectifier (SCR). A latchup is formed by N and P regions in the layout of the integrated circuit, which act as the collector, base and emitter of unintentional parasitic transistors. Bulk resistance of silicon in the wells and substrate acts as resistors in the SCR circuit. Application of voltages to pins above V_{DD} + 0.3 V or below V_{SS} – 0.3 V in conjunction with enough current to develop voltage drops across the parasitic resistors can cause the SCR to turn on. Once on, the SCR can be turned off only by removal of all power and applied voltages. The low on impedance of the SCR circuit can overheat and destroy the IC.

Figure D-2 shows the MOS circuitry for a digital input-only pin. The gates of the input buffer are very high impedance for all voltages that would ever be applied to the pin. Protection is implemented with a P-channel transistor acting as a diode to V_{DD} and an N-channel transistor acting as a diode to V_{SS}. Allowing a pin to float or be driven to a mid-supply level can result in both the N- and P-channel devices in the input buffer simultaneously being partially on, which causes excess current and noise on the V_{DD}/V_{SS} power supply. If a digital input is driven above V_{DD}, the pseudo-diode will conduct, protecting the input. As the current is increased to high levels (100 mA), damage can result. Figure D-3 shows the CMOS circuitry for a digital input/output-only pin.

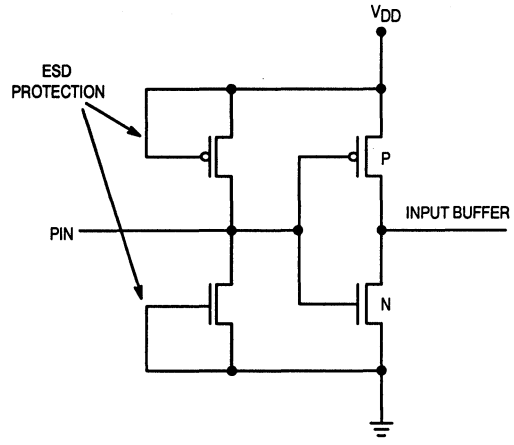


Figure D-2. Digital Input

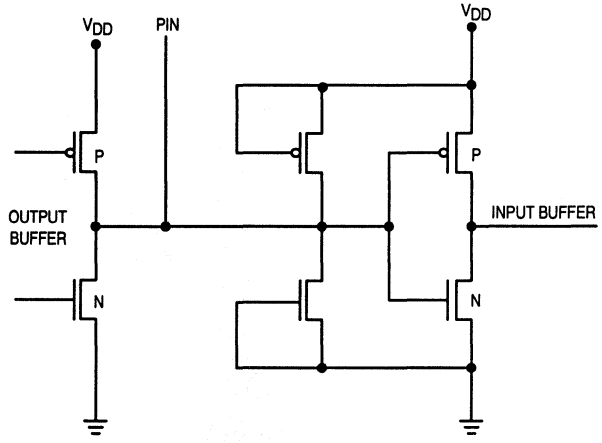


Figure D-3. Digital I/O

D.1.3 Power Supply, Ground, and Noise Considerations

This device may be used in digital equipment applications which require plugging the PC board into a rack with power applied. This is referred to as "hot-rack insertion". In these applications, care should be taken to limit the voltage on any pin from going positive relative to the V_{DD} pins or negative relative to the V_{SS} pins. One method to accomplish this is to extend the ground and power contacts of the PCB connector so that power is applied prior to any other pins having voltage applied. The device has input protection on all pins and may source or sink a limited amount of current without damage. See Section 6.2.1, Absolute Maximum Ratings, for more information concerning the current into or out of the device pins. Current limiting may be accomplished by series resistors between the signal pins and the connector contacts.

The most important considerations for PCB layout deal with noise. This includes noise on the power supply, noise generated by the digital circuitry on the device, and coupling digital signals into the analog signals. The best PCB layout methods to prevent noise induced problems are:

1) Keep digital signals as far away from analog signals as possible.
2) Use short, low inductance traces for the analog circuitry to reduce inductive, capacitive, and radio frequency noise sensitivities.
3) Use short, low inductance traces for digital circuitry to reduce inductive, capacitive, and radio frequency radiated noise.
4) Bypass capacitors should be connected between the V_{DD} and V_{SS} pairs with minimal trace length. These capacitors help supply the instantaneous currents of the digital circuitry in addition to decoupling the noise that may be generated by other sections of the device or other circuitry on the power supply.
5) Use short, wide, low inductance traces to connect all of the V_{SS} ground pins together and, with one trace, connect all of the V_{SS} ground pins to the power supply ground. Depending on the application, a double sided PCB with a V_{SS} ground plane under the device connecting all of the digital and analog V_{SS} pins together would be a good grounding method. A multi-layer PCB with a ground plane connecting all of the digital and analog V_{SS} pins together would be the optimal ground configuration. These methods will result in the lowest resistance and the lowest inductance in the ground circuit. This is important to reduce voltage spikes in the ground circuit resulting from the high speed digital current spikes. Suppressing these voltage spikes on the integrated circuit is the reason for multiple V_{SS} ground leads.
6) Use short, wide, low inductance traces to connect all of the V_{DD} power supply pins together and, with one trace, connect all of the V_{DD} power supply pins to the 5 volt power supply. Depending on the application, a double sided PCB with V_{DD} bypass capacitors to the V_{SS} ground plane under the device may complete the low impedance coupling for the power supply. For a multi-layer PCB with a power plane, connecting all of the digital and analog V_{DD} pins to the power plane would be the optimal power distribution method. The integrated circuit layout and packaging considerations for the 5 volt V_{DD} power circuit are essentially the same as for the ground circuit.
7) Motorola recommends that a four layer board be used. It is possible to use a two layer board but special care must be taken.

D.2 BOARD SOLDERING CONSIDERATIONS

All production orders of NEURON CHIPS **will be shipped dry pack**. Dry pack is a process which slowly bakes moisture from the SMT package and then seals it into a dry pack bag to shield the unit from moisture in the atmosphere. The exterior of the bag will be marked with a label that indicates the devices are moisture sensitive and marked with the date the bag was sealed (there is a one year shelf life for these devices). There is a limited amount of time to use surface mount devices once they are removed from the dry pack. It is recommended that before surface mounting, packages should not be out of the dry pack longer than 48 hours at standard test floor conditions of 45% RH and 25°C. If the units have not been shipped dry pack or have been unpacked for too long, then units must be baked at 125°C for 24 hours prior to board soldering. If this is not done, some percentage of units will exhibit destructive failures or latent failures after the soldering process.

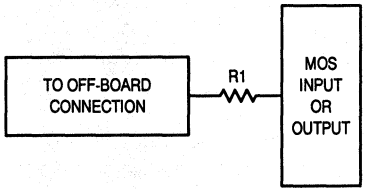
D.3 HANDLING PRECAUTIONS

All CMOS devices have an insulated gate that is subject to voltage breakdown. The high-impedance gates on the devices are protected by on-chip networks. However, these on-chip networks do not make the IC immune to electrostatic damage (ESD). Laboratory tests show that devices may fail after one very high voltage discharge. They may also fail due to the cumulative effect of several discharges of lower potential.

Static-damaged devices behave in various ways, depending on the severity of the damage. The most severely damaged are the easiest to detect because the input or output has been completely destroyed and is either shorted to V_{DD}, shorted to V_{SS}, or open-circuited. The effect is that the device is no longer functional. Less severe cases are more difficult to detect because they appear as intermittent failures or degraded performance. Static damage can often increase leakage currents.

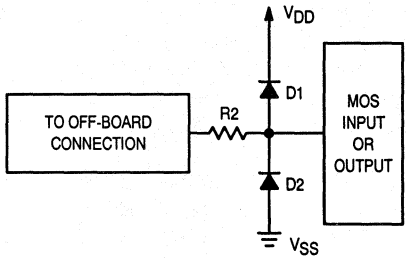
CMOS devices are not immune to large static voltage discharges that can be generated while handling. For example, static voltages generated by a person walking across a waxed floor have been measured in the 4–15 kV range (depending on humidity, surface conditions, etc.). Therefore, the following precautions should be observed.

1. Do not exceed the Maximum Ratings specified by the data sheet.
2. All unused device inputs should be connected to V_{DD} or V_{SS}.
3. All low-impedance equipment (pulse generators, etc.) should be connected to CMOS inputs only after the device is powered up. Similarly, this type of equipment should be disconnected before power is turned off.
4. A circuit board containing CMOS or devices is merely an extension of the device and the same handling precautions apply. Contacting connectors wired directly to devices can cause damage. Plastic wrapping should be avoided. When external connections to a PC board address pins of CMOS integrated circuits, a resistor should be used in series with the inputs or outputs. The limiting factor for the series resistor is the added delay caused by the time constant formed by the series resistor and input capacitance. This resistor will help limit accidental damage if the PC board is removed and brought into contact with static generating materials. For convenience, equations for added propagation delay and rise time effects due to series resistance size are given in Figure D-4.
5. All CMOS devices should be stored or transported in materials that are antistatic. Devices must not be inserted into conventional plastic "snow", styrofoam or plastic trays, but should be left in their original container until ready for use.
6. All CMOS devices should be placed on a grounded bench surface and operators should ground themselves prior to handling devices, since a worker can be statically charged with respect to the bench surface. Wrist straps in contact with skin are strongly recommended. See Figure D-5.
7. Nylon or other static generating materials should not come in contact with CMOS circuits.
8. If automatic handling is being used, high levels of static electricity may be generated by the movement of devices, belts, or boards. Reduce static build-up by using ionized air blowers or room humidifiers. All parts of machines which come into contact with the top, bottom, and sides of IC packages must be grounded metal or other conductive material.
9. Cold chambers using CO_2 for cooling should be equipped with baffles, and devices must be contained on or in conductive material.
10. When lead-straightening or hand-soldering is necessary, provide ground straps for the apparatus used and be sure that soldering ties are grounded.
11. The following steps should be observed during wave solder operations.
    a. The solder pot and conductive conveyor system of the wave soldering machine must be grounded to an earth ground.
    b. The loading and unloading work benches should have conductive tops which are grounded to an earth ground.
    c. Operators must comply with precautions previously explained.
    d. Completed assemblies should be placed in antistatic containers prior to being moved to subsequent stations.



Advantage: Requires minimal board area.

Disadvantage: R1 > R2 for the same level of protection, therefore rise and fall times, propagation delays, and output drives are severely affected.



Advantage: R2 < R1 for the same level of protection. Impact on ac and dc characteristics is minimized.

Disadvantage: More board area, higher initial cost.

NOTE: These networks are useful for protecting the following:
A. digital inputs and outputs
B. analog inputs and outputs
C. 3-state outputs
D. bidirectional (I/O) ports

Equation 1 – Propagation Delay vs Series Resistance

$$R \approx \frac{t}{C \cdot k}$$

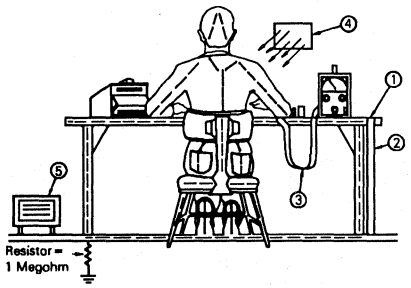
where:
R = the maximum allowable series resistance in ohms
t = the maximum tolerable propagation delay in seconds
C = the board capacitance plus the driven device's input capacitance in farads
k = 0.33 for the TTL input levels (switch point = 1.3 V)

Equation 2 – Rise Time vs Series Resistance

$$R \approx \frac{t}{C \cdot k}$$

where:
R = the maximum allowable series resistance in ohms
t = the maximum rise time per data sheet in seconds
C = the board capacitance plus the driven device's input capacitance in farads
k = 2.3 for other devices

Figure D-4. Networks for Minimizing ESD and Reducing CMOS Latch Up Susceptibility



NOTES:
1. 1/16 inch conductive sheet stock covering bench top work area.
2. Ground strap.
3. Wrist strap in contact with skin.
4. Static neutralizer. (Ionized air blower directed at work.) Primarily for use in areas where direct grounding is impractical.
5. Room humidifier. Primarily for use in areas where the relative humidity is less than 45%. Caution: building heating and cooling systems usually dry the air causing the relative humidity inside of buildings to be less than outside humidity.

Figure D-5. Typical Manufacturing Work Station

12. The following steps should be observed during board cleaning operation.
    a. Vapor degreasers and baskets must be grounded to an earth ground. Operators must likewise be grounded.
    b. Brush or spray cleaning should not be used.
    c. Assemblies should be placed into the vapor degreaser immediately upon removal from the antistatic container.
    d. Cleaned assemblies should be placed in antistatic containers immediately after removal from the cleaning basket.
    e. High velocity air movement or application of solvents and coatings should be employed only when module circuits are grounded and a static eliminator is directed at the module.
13. The use of static detection meters for line surveillance is highly recommended.
14. Equipment specifications should alert users to the presence of CMOS devices and require familiarization with this specification prior to performing any kind of maintenance or replacement of devices or modules.
15. Do not insert or remove CMOS devices from test sockets with power applied. Check all power supplies to be used for testing devices to be certain there are no voltage transients present.
16. Double check the equipment setup for proper polarity of voltage before conducting parametric or functional testing.
17. Do not recycle shipping rails. Continuous use causes deterioration of their antistatic coating.

RECOMMENDED READING

"Total Control of the Static in Your Business"

Available by writing to:

Static Control Systems Div.
Box ELB-3, 225-4S
3M Center
St. Paul, MN 55144

Or calling:

1-800-328-1368
1-612-733-9420 (in Minnesota)

CMOS LATCH UP

Latch up will not be a problem for most designs, but the designer should be aware of it, what causes it, and how to prevent it.

Figure D-6 shows the layout of a typical CMOS inverter and Figure D-7 shows the parasitic bipolar devices that are formed. The circuit formed by the parasitic transistors and resistors is the basic configuration of a silicon controlled rectifier, or SCR. In the latch-up condition, transistors Q1 and Q2 are turned on, each providing the base current necessary for the other to remain in saturation, thereby latching the devices on. Unlike a conventional SCR, where the device is turned on by applying a voltage to the base of the NPN transistor, the parasitic SCR is turned on by applying a voltage to the emitter of either transistor. The two emitters that trigger the SCR are the same point, the CMOS output. Therefore, to latch up the CMOS device, the output voltage must be greater than V_{DD} + 0.5 Vdc or less than – 0.5 Vdc and have sufficient current to trigger the SCR. The latch-up mechanism is similar for the inputs.

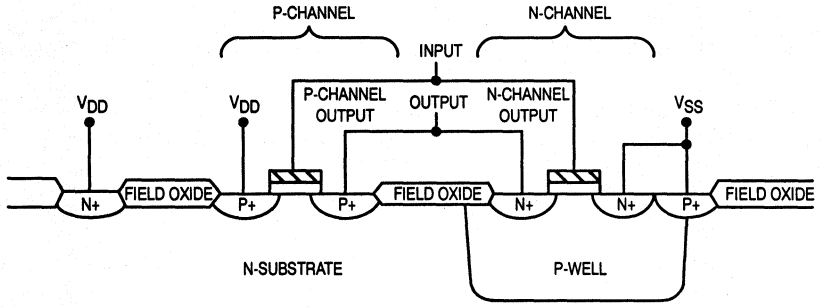


Figure D-6. CMOS Wafer Cross Section

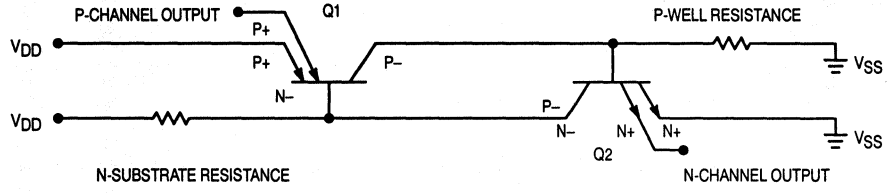


Figure D-7. Latch Up Circuit Schematic

Once a CMOS device is latched up, if the supply current is not limited, the device will be destroyed. Ways to prevent such occurrences are listed below.

1. Ensure that inputs and outputs are limited to the maximum rated values, as follows:
   $-0.5 \le V_{in} \le V_{DD} + 0.5$ Vdc referenced to V_{SS}
   $-0.5 \le V_{out} \le V_{DD} + 0.5$ Vdc referenced to V_{SS}
   $|I_{in}| \le 10$ mA
   $|I_{out}| \le 10$ mA when transients or dc levels exceed the supply voltages.
2. If voltage transients of sufficient energy to latch up the device are expected on the outputs, external protection diodes can be used to clamp the voltage. Another method of protection is to use a series resistor to limit the expected worst case current to the Maximum Ratings values. See Figure D-4.
3. If voltage transients are expected on the inputs, protection diodes may be used to clamp the voltage or a series resistor may be used to limit the current to a level less than the maximum rating of $I_{in} = 10$ mA. See Figure D-4.
4. Sequence power supplies so that the inputs or outputs of CMOS devices are not powered up first (e.g., recessed edge connectors may be used in plug-in board applications and/or series resistors).
5. Power supply lines should be free of excessive noise. Care in board layout and filtering should be used.
6. Limit the available power supply current to the devices that are subject to latch-up conditions. This can be accomplished with the power supply filtering network or with a current-limiting regulator.

Literature Distribution Centers:
USA: Motorola Literature Distribution; P.O. Box 20912; Phoenix, Arizona 85036.
EUROPE: Motorola Ltd.; European Literature Centre; 88 Tanners Drive, Blakelands, Milton Keynes, MK14 5BP, England.
JAPAN: Nippon Motorola Ltd.; 4-32-1, Nishi-Gotanda, Shinagawa-ku, Tokyo 141, Japan.
ASIA PACIFIC: Motorola Semiconductors H.K. Ltd.; Silicon Harbour Center, No. 2 Dai King Street, Tai Po Industrial Estate, Tai Po, N.T., Hong Kong.

MC143150/D

1ATX30337-1 PRINTED IN USA 7/93 IMPERIAL LITHO 92797 12,000 MOS D-A YFAAAA