

TOSHIBA

Neuron[®] Chip

TMPN3150/3120

TOSHIBA CORPORATION

Semiconductor Company

The information contained herein is subject to change without notice. 021023_D

TOSHIBA is continually working to improve the quality and reliability of its products. Nevertheless, semiconductor devices in general can malfunction or fail due to their inherent electrical sensitivity and vulnerability to physical stress.

It is the responsibility of the buyer, when utilizing TOSHIBA products, to comply with the standards of safety in making a safe design for the entire system, and to avoid situations in which a malfunction or failure of such TOSHIBA products could cause loss of human life, bodily injury or damage to property.

In developing your designs, please ensure that TOSHIBA products are used within specified operating ranges as set forth in the most recent TOSHIBA products specifications.

Also, please keep in mind the precautions and conditions set forth in the "Handling Guide for Semiconductor Devices," or "TOSHIBA Semiconductor Reliability Handbook" etc. 021023_A

The Toshiba products listed in this document are intended for usage in general electronics applications (computer, personal equipment, office equipment, measuring equipment, industrial robotics, domestic appliances, etc.).

These Toshiba products are neither intended nor warranted for usage in equipment that requires extraordinarily high quality and/or reliability or a malfunction or failure of which may cause loss of human life or bodily injury ("Unintended Usage"). Unintended Usage include atomic energy control instruments, airplane or spaceship instruments, transportation instruments, traffic signal instruments, combustion control instruments, medical instruments, all types of safety devices, etc. Unintended Usage of Toshiba products listed in this document shall be made at the customer's own risk. 021023_B

The products described in this document shall not be used or embedded to any downstream products of which manufacture, use and/or sale are prohibited under any applicable laws and regulations. 060106_Q

The information contained herein is presented only as a guide for the applications of our products. No responsibility is assumed by TOSHIBA for any infringements of patents or other rights of the third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of TOSHIBA or others. 021023_C

Neuron[®] Chips are manufactured and sold under license from Echelon. In order to purchase Neuron Chips, customers must first enter into an OEM License Agreement.

Echelon, LON, LONWORKS, 3120, 3150, Digital Home, iLON, LNS, LONMARK, LonBuilder, LonMaker, LonManager, LonPoint, LonTalk, LonUsers, LONWORLD, Neuron, NodeBuilder, Panoramix, ShortStack, the Echelon logo, and the LonUsers logo are trademarks of Echelon Corporation registered in the United States and other countries.

LONews, LonLink, LonResponse, LonScanner, LonSupport, Open Systems Alliance, OpenLDV, Pyxos, Powered by Echelon, Digital Home Powered by Echelon, LNS Powered by Echelon, LONWORKS Powered by Echelon, Networked Energy Services Powered by Echelon, NES Powered by Echelon, Panoramix Powered by Echelon, and Thinking Inside the Box are trademarks of Echelon Corporation.

The products described in this document may include products subject to the foreign exchange and foreign trade laws. 021023_F

Preface

This data book is intended for use with the following Toshiba products:
TMPN3120FE3MG and TMPN3120FE5MG are hereinafter called
TMPN3120××, TMPN3150B1AFG. Older versions of these parts are not
discussed in this book. For information on these first generation parts,
contact Toshiba Corporation.

March 2006

TOSHIBA CORPORATION
Semiconductor Company

Contents

[1]	Discontinued Products and Product List.....	13
	1. List of Discontinued Products.....	15
	2. Product Comparison	15
[2]	Handling Precautions	17
	1. Using Toshiba Semiconductors Safely.....	19
	2. Safety Precautions.....	20
	3. General Safety Precautions and Usage Considerations.....	22
	4. Precautions and Usage Considerations Specific to Microcontrollers	39
[3]	Introduction	41
	1. General Description	43
[4]	Technical Datasheets.....	45
	TMPN3150B1AFG	47
	TMPN3120FE3MG	54
	TMPN3120FE5MG	62
[5]	LONWORKS Architectural Overview	71
[6]	CPUs	75

[7]	Memory	83
1.	EEPROM	85
2.	RAM	86
3.	ROM	86
4.	External Memory.....	87
[8]	LonTalk Protocol.....	89
1.	Multiple Media Support	91
2.	Support for Multiple Communication Channels	92
3.	Communication Rates.....	92
4.	LonTalk Addressing Limits.....	93
5.	Message Services	94
6.	Authentication.....	95
7.	Priority	95
8.	Collision Avoidance	95
9.	Collision Detection	96
10.	Data Interpretation	96
11.	Network Management and Diagnostic Services.....	96
[9]	Network Communication Port.....	97
1.	Single-Ended Mode	101
2.	Differential Mode.....	106
3.	Special-Purpose Mode.....	107

[10] Programming Model.....	111
1. Timers.....	113
2. Network Variables.....	114
3. Network Variable Aliases	118
4. Explicit Messages	119
5. Scheduler	124
6. Additional Functions	126
7. Built-In Variables.....	132
8. TMPN3120xx Chip Firmware Extensions	132
 [11] Application I/O.....	 133
1. I/O Timing Issues.....	141
2. I/O Objects.....	145
3. Notes	207
 [12] Additional Functions	 209
1. Service Pin	211
2. Sleep/Wake-Up Circuitry.....	212
3. Watchdog Timer	213
4. Reset Circuit	214
5. Programmable Low Voltage Detection.....	218
6. Programmable Reset Time	219
7. Reset Processes and Timing	220
8. Clocking System.....	227
 [13] Package Types and Dimensions	 229
1. PCB Neuron Chip Pad Layout	233
2. Socket Information.....	234

[14] Electrical Characteristics	235
1. Communications Port Differential Receiver Electrical Characteristics.....	237
2. Memory Interface Timing Specifications	239
 Appendix A:	A-1
A.1 Fixed Read-Only Data Structure	A-9
A.2 The Domain Table	A-15
A.3 The Address Table	A-17
A.4 Network Variable Tables.....	A-25
A.5 The Standard Network Variable Type (SNVT) Structures	A-30
A.6 The Configuration Structure.....	A-37
 Appendix B:	B-1
B.1 Network Management Messages.....	B-8
B.2 Network Diagnostic Messages.....	B-30
B.3 Network Variable Messages	B-36

Usage Guidelines for Echelon Trademarks

Note: These guidelines do not cover usage of the Echelon Logo, the LNS[®] Powered by Echelon Logo, LONWORKS[®] Independent Developer Logo, the trademark LONMARK[®], the LONMARK Logo, the Open Systems Alliance[™] program logos, the Echelon Authorized Network Integrator Logo, or the Echelon LONWORKS Integrator Partner Logo.

1. Always Use Echelon Trademarks in an Approved Form.

Trademarks should be presented in the recommended styles shown below (all caps, upper and lower case, or a combination of large and small caps):

Registered Trademarks

3120[®]
3150[®]
BeAtHome[®]
Digital Home[®]
Echelon[®]
i.LON[®]
LNS[®]

LON[®]
LonBuilder[®]
LonMaker[®]
LonManager[®]
LONMARK[®]
LonPoint[®]
LonTalk[®]

LonUsers[®]
LONWORKS[®]
LONWORLD[®]
Neuron[®]
NodeBuilder[®]
Panoramix[®]
ShortStack[®]

Other Trademarks

LONews[™]
LonLink[™]
LonResponse[™]
LonScanner[™]
LonSupport[™]
Open Systems
Alliance[™]
OpenLDV[™]
Pyxos[™]

Powered by Echelon[™]
Digital Home Powered by Echelon[™]
LNS Powered by Echelon[™]
LONWORKS Powered by Echelon[™]
NES Powered by Echelon[™]
Networked Energy Services Powered by
Echelon[™]
Panoramix Powered by Echelon[™]
Thinking Inside the Box[™]

Changes or alterations in the names above are not generally allowed. Any variation must be cleared through Echelon's Advertising and Creative Services Director.

RIGHT
LONWORKS[®]
WRONG
Lonworks[®]

RIGHT
LonTalk[®]
WRONG
LONTALK[®]

2. Always use Echelon Trademarks in a Manner Clearly Indicating That They are Trademarks Owned by Echelon.

Trademarks should be properly marked to give notice that they are, in fact, trademarks of Echelon Corporation. Registered trademarks should always be used with the federal registration symbol “®”, while marks that have not been registered (including marks that are the subject of “pending” applications) should be used with the symbol “™ ” where the mark initially appears. Use of the registration symbol or “™ ” must appear with the first usage of the trademark in a document; subsequent occurrences do not require the symbols. When used in a printed document, the symbols should be half the point size of the word and then superscripted half the point size of the word.

A footnote reference to ownership of the trademarks must be used on all products, documentation, and advertisements in the following format:

“Echelon, LON, LONWORKS, 3120, 3150, Digital Home, i.LON, LNS, LONMARK, LonBuilder, LonMaker, LonManager, LonPoint, LonTalk, LonUsers, LONWORLD, Neuron, NodeBuilder, Panoramix, ShortStack, the Echelon logo, and the LonUsers logo are trademarks of Echelon Corporation registered in the United States and other countries. LONews, LonLink, LonResponse, LonScanner, LonSupport, Open Systems Alliance, OpenLDV, Pyxos, Powered by Echelon, Digital Home Powered by Echelon, LNS Powered by Echelon, LONWORKS Powered by Echelon, Networked Energy Services Powered by Echelon, NES Powered by Echelon, Panoramix Powered by Echelon, and Thinking Inside the Box are trademarks of Echelon Corporation.”

Where appropriate, a subset of this attribution may be used.

The list of registered and unregistered marks will be updated periodically as pending applications mature to registration, and as such, it is important to determine status of the marks before using either symbol.

Trademarks should not be joined with other terms (by a hyphen, for instance) nor used with unapproved logos, graphics, photos, slogans, numbers, design features or symbols. Trademarks should never be “made plural,” never be mixed with other trademarks, and a trademark’s spelling should never be altered.

3. Always use Echelon Trademarks as Adjectives, Never Nouns, and the Marks Should be Followed by the Appropriate Generic Terminology.

Trademarks are meant to signify the brand or source of the product and should not be used in a manner which suggests that the trademark is the name of the product. Companies that allow their trademarks to be used as nouns risk losing their rights in those trademarks. Classic examples of words that began as trademarks but were lost due to misuse include “aspirin,” “escalator” and “cellophane.” Because the public came to see these terms as the product names instead of the brand names, trademark rights were forfeited.

The most common mistake is to use the trademark as a noun instead of as an adjective followed by the generic term. Examples:

RIGHT

“LONWORKS[®] networks can control existing home systems.”

WRONG

“LONWORKS[®] can control existing home systems.”

RIGHT

“The LonTalk[®] protocol is built into every Neuron[®] Chip.”

WRONG

“LonTalk[®] is built into every Neuron[®].”

4. Do not Imply or Suggest that a Product Based on Echelon's Technology is an Echelon Product or that Echelon Sponsors the Product.

While Echelon understands and appreciates the customer’s need to accurately describe the technology incorporated into the product as Echelon technology, the product materials and advertising cannot wrongly imply that the customer’s product is an Echelon product, explicitly or implicitly. Care must be taken to clearly distinguish Echelon’s products (and trademarks) from the customer’s products and marks, as discussed more specifically in these guidelines. Some basic examples are:

RIGHT

“The ACME[™] light switch is for use in LONWORKS[®] networks.”

WRONG

“The LONWORKS[®] light switch is made by ACME[™].”

RIGHT

“The ACME[™] security system incorporates Echelon Corporation's LonTalk[®] protocol.”

WRONG

“The ACME[™] LonTalk[®] security system incorporates the Echelon protocol.”

5. Never use the Trademark “LON” as a Separate Word, Even if Used as an Adjective.

6. Never use an Echelon Trademark as Part of Your Name or Trademark.

Even though LON is a registered trademark of Echelon Corporation, it should not be used by a customer except as part of Echelon's family of marks, such as LonTalk, LonBuilder, and the like.

Echelon trademarks may not be used as part of another company's mark or company name. While Echelon's marks can be used in explanatory fashion (such as "for use in LONWORKS® networks"), the mark cannot be used as part of the company or product name. The only exception to this rule relates specifically to LON, where precise guidelines have been established. The mark LON may be used ONLY if it is incorporated into a composite mark in such a way that (1) the LON portion is not highlighted in any way, including by CAPITAL letters, graphics, or stylizations, and (2) that LON is not used as the beginning letters of the composite word mark, and (3) the LON portion may not be preceded by the letters I or E if the letter forms the beginning of the word.

Acceptable**Unacceptable**

**Avalon AvaLON DRESLON DresLon brinlon Lonbrin alonet alonet Relongate Elongate
llongate**

Also, customers may not personify trademarks or create characters that represent the trademarks. However, the trademarks may be used in taglines in accordance with the guidelines listed above.

7. Certain echelon Marks Are not Available for Use Except by WRITTEN LICENSE Agreement and Compliance with standards.

Echelon owns marks which may only be used by organizations that have executed written agreements and have complied with certain requirements. These marks include the Echelon Logo, the LNS Powered by Echelon Logo, LONWORKS Independent Developer Logo, the Open Systems Alliance program Logos, the trademark LONMARK, LONMARK Logo, the Echelon Authorized Network Integrator Logo, and Echelon LonWorks Integrator Partner Logo. This document does not give the right to use such trademarks. These marks may only be used by organizations that have written agreements. For more information please contact the responsible program manager at Echelon.

[1] Discontinued Products and Product List

[1] Discontinued Products and Product List

1. List of Discontinued Products

Discontinued	Recommended Alternative
TMPN3150AF TMPN3150BF TMPN3150B1F TMPN3150B1AF	TMPN3150B1AFG
TMPN3120AM TMPN3120B1M TMPN3120B1AM TMPN3120FE1M TMPN3120A20M TMPN3120E3M TMPN3120FE3M TMPN3120FE5M	TMPN3120FE3MG (or TMPM3120FE5MG)
TMPN3120A20U TMPN3120FE3MU	No alternative in TOSHIBA

2. Product Comparison

Product Name	EEPROM (in bytes)	RAM (in bytes)	ROM (in bytes)	External Memory I/F	Maximum Operating Frequency (MHz)	Package	Others
TMPN3150B1AFG	512	2 K	—	Available (Note1)	10	64QFP	Three 8-bit CPUs
TMPN3120FE3MG	2 K	2 K	16 K	Not available	20	32SOP	Two channels for 16-bit timer/ counter
TMPN3120FE5MG	3 K	4 K	16 K	Not available	20	32SOP	

Note1: Out of 58 Kbytes of external memory, 42 Kbytes are available for the user.

[2] Handling Precautions

[2] Handling Precautions

1. Using Toshiba Semiconductors Safely

TOSHIBA is continually working to improve the quality and reliability of its products. Nevertheless, semiconductor devices in general can malfunction or fail due to their inherent electrical sensitivity and vulnerability to physical stress. It is the responsibility of the buyer, when utilizing TOSHIBA products, to comply with the standards of safety in making a safe design for the entire system, and to avoid situations in which a malfunction or failure of such TOSHIBA products could cause loss of human life, bodily injury or damage to property. In developing your designs, please ensure that TOSHIBA products are used within specified operating ranges as set forth in the most recent TOSHIBA products specifications. Also, please keep in mind the precautions and conditions set forth in the “Handling Guide for Semiconductor Devices,” or “TOSHIBA Semiconductor Reliability Handbook” etc..




The TOSHIBA products listed in this document are intended for usage in general electronics applications (computer, personal equipment, office equipment, measuring equipment, industrial robotics, domestic appliances, etc.). These TOSHIBA products are neither intended nor warranted for usage in equipment that requires extraordinarily high quality and/or reliability or a malfunction or failure of which may cause loss of human life or bodily injury (“Unintended Usage”). Unintended Usage include atomic energy control instruments, airplane or spaceship instruments, transportation instruments, traffic signal instruments, combustion control instruments, medical instruments, all types of safety devices, etc.. Unintended Usage of TOSHIBA products listed in this document shall be made at the customer’s own risk.

2. Safety Precautions


This section lists important precautions which users of semiconductor devices (and anyone else) should observe in order to avoid injury and damage to property, and to ensure safe and correct use of devices.

Please be sure that you understand the meanings of the labels and the graphic symbol described below before you move on to the detailed descriptions of the precautions.

[Explanation of labels]

	Indicates an imminently hazardous situation which will result in death or serious injury if you do not follow instructions.
	Indicates a potentially hazardous situation which could result in death or serious injury if you do not follow instructions.
	Indicates a potentially hazardous situation which if not avoided, may result in minor injury or moderate injury.

[Explanation of graphic symbol]

Graphic symbol	Meaning
	Indicates that caution is required (laser beam is dangerous to eyes).

2.1 General Precautions regarding Semiconductor Devices

⚠ CAUTION

Do not use devices under conditions exceeding their absolute maximum ratings (e.g. current, voltage, power dissipation or temperature).

This may cause the device to break down, degrade its performance, or cause it to catch fire or explode resulting in injury.

Do not insert devices in the wrong orientation.

Make sure that the positive and negative terminals of power supplies are connected correctly. Otherwise the rated maximum current or power dissipation may be exceeded and the device may break down or undergo performance degradation, causing it to catch fire or explode and resulting in injury.

When power to a device is on, do not touch the device's heat sink.

Heat sinks become hot, so you may burn your hand.

Do not touch the tips of device leads.

Because some types of device have leads with pointed tips, you may prick your finger.

When conducting any kind of evaluation, inspection or testing, be sure to connect the testing equipment's electrodes or probes to the pins of the device under test before powering it on.

Otherwise, you may receive an electric shock causing injury.

Before grounding an item of measuring equipment or a soldering iron, check that there is no electrical leakage from it.

Electrical leakage may cause the device which you are testing or soldering to break down, or could give you an electric shock.

Always wear protective glasses when cutting the leads of a device with clippers or a similar tool.

If you do not, small bits of metal flying off the cut ends may damage your eyes.

3. General Safety Precautions and Usage Considerations

This section is designed to help you gain a better understanding of semiconductor devices, so as to ensure the safety, quality and reliability of the devices which you incorporate into your designs.

3.1 From Incoming to Shipping

3.1.1 Electrostatic discharge (ESD)

When handling individual devices (which are not yet mounted on a printed circuit board), be sure that the environment is protected against electrostatic electricity. Operators should wear anti-static clothing, and containers and other objects which come into direct contact with devices should be made of anti-static materials and should be grounded to earth via an 0.5- to 1.0-M Ω protective resistor.



Please follow the precautions described below; this is particularly important for devices which are marked “Be careful of static.”.

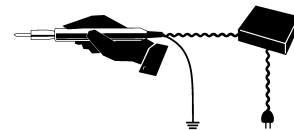
(1) Work environment

- When humidity in the working environment decreases, the human body and other insulators can easily become charged with static electricity due to friction. Maintain the recommended humidity of 40% to 60% in the work environment, while also taking into account the fact that moisture-proof-packed products may absorb moisture after unpacking.
- Be sure that all equipment, jigs and tools in the working area are grounded to earth.
- Place a conductive mat over the floor of the work area, or take other appropriate measures, so that the floor surface is protected against static electricity and is grounded to earth. The surface resistivity should be 10^4 to $10^8 \Omega/\text{sq}$ and the resistance between surface and ground, 7.5×10^5 to $10^8 \Omega$.
- Cover the workbench surface also with a conductive mat (with a surface resistivity of 10^4 to $10^8 \Omega/\text{sq}$, for a resistance between surface and ground of 7.5×10^5 to $10^8 \Omega$). The purpose of this is to disperse static electricity on the surface (through resistive components) and ground it to earth. Workbench surfaces must not be constructed of low-resistance metallic materials that allow rapid static discharge when a charged device touches them directly.
- Pay attention to the following points when using automatic equipment in your workplace:
 - (a) When picking up ICs with a vacuum unit, use a conductive rubber fitting on the end of the pick-up wand to protect against electrostatic charge.
 - (b) Minimize friction on IC package surfaces. If some rubbing is unavoidable due to the device's mechanical structure, minimize the friction plane or use material with a small friction coefficient and low electrical resistance. Also, consider the use of an ionizer.
 - (c) In sections which come into contact with device lead terminals, use a material which dissipates static electricity.
 - (d) Ensure that no statically charged bodies (such as work clothes or the human body) touch the devices.

- (e) Make sure that sections of the tape carrier which come into contact with installation devices or other electrical machinery are made of a low-resistance material.
- (f) Make sure that jigs and tools used in the assembly process do not touch devices.
- (g) In processes in which packages may retain an electrostatic charge, use an ionizer to neutralize the ions.
- Make sure that CRT displays in the working area are protected against static charge, for example by a VDT filter. As much as possible, avoid turning displays on and off. Doing so can cause electrostatic induction in devices.
- Keep track of charged potential in the working area by taking periodic measurements.
- Ensure that work chairs are protected by an anti-static textile cover and are grounded to the floor surface by a grounding chain. (Suggested resistance between the seat surface and grounding chain is 7.5×10^5 to $10^{12} \Omega$.)
- Install anti-static mats on storage shelf surfaces. (Suggested surface resistivity is 10^4 to $10^8 \Omega/\text{sq}$; suggested resistance between surface and ground is 7.5×10^5 to $10^8 \Omega$.)
- For transport and temporary storage of devices, use containers (boxes, jigs or bags) that are made of anti-static materials or materials which dissipate electrostatic charge.
- Make sure that cart surfaces which come into contact with device packaging are made of materials which will conduct static electricity, and verify that they are grounded to the floor surface via a grounding chain.
- In any location where the level of static electricity is to be closely controlled, the ground resistance level should be Class 3 or above. Use different ground wires for all items of equipment which may come into physical contact with devices.

(2) Operating environment

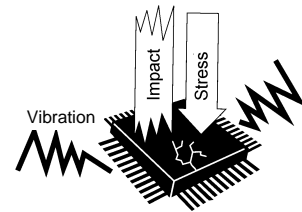
- Operators must wear anti-static clothing and conductive shoes (or a leg or heel strap).
- Operators must wear a wrist strap grounded to earth via a resistor of about $1 \text{ M}\Omega$.
- Soldering irons must be grounded from iron tip to earth, and must be used only at low voltages (6 V to 24 V).
- If the tweezers you use are likely to touch the device terminals, use anti-static tweezers and in particular avoid metallic tweezers. If a charged device touches a low-resistance tool, rapid discharge can occur. When using vacuum tweezers, attach a conductive chucking pat to the tip, and connect it to a dedicated ground used especially for anti-static purposes (suggested resistance value: 10^4 to $10^8 \Omega$).
- Do not place devices or their containers near sources of strong electrical fields (such as above a CRT).



- When storing printed circuit boards which have devices mounted on them, use a board container or bag that is protected against static charge. To avoid the occurrence of static charge or discharge due to friction, keep the boards separate from one other and do not stack them directly on top of one another.
- Ensure, if possible, that any articles (such as clipboards) which are brought to any location where the level of static electricity must be closely controlled are constructed of anti-static materials.
- In cases where the human body comes into direct contact with a device, be sure to wear anti-static finger covers or gloves (suggested resistance value: $10^8 \Omega$ or less).
- Equipment safety covers installed near devices should have resistance ratings of $10^9 \Omega$ or less.
- If a wrist strap cannot be used for some reason, and there is a possibility of imparting friction to devices, use an ionizer.

3.1.2 Vibration, impact and stress

Handle devices and packaging materials with care. To avoid damage to devices, do not toss or drop packages. Ensure that devices are not subjected to mechanical vibration or shock during transportation. Ceramic package devices and devices in canister-type packages which have empty space inside them are subject to damage from vibration and shock because the bonding wires are secured only at their ends.



Plastic molded devices, on the other hand, have a relatively high level of resistance to vibration and mechanical shock because their bonding wires are enveloped and fixed in resin. However, when any device or package type is installed in target equipment, it is to some extent susceptible to wiring disconnections and other damage from vibration, shock and stressed solder junctions. Therefore when devices are incorporated into the design of equipment which will be subject to vibration, the structural design of the equipment must be thought out carefully.

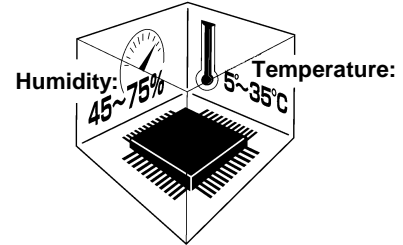
If a device is subjected to especially strong vibration, mechanical shock or stress, the package or the chip itself may crack. In products such as CCDs which incorporate window glass, this could cause surface flaws in the glass or cause the connection between the glass and the ceramic to separate.

Furthermore, it is known that stress applied to a semiconductor device through the package changes the resistance characteristics of the chip because of piezoelectric effects. In analog circuit design attention must be paid to the problem of package stress as well as to the dangers of vibration and shock as described above.

3.2 Storage

3.2.1 General storage

- Avoid storage locations where devices will be exposed to moisture or direct sunlight.
- Follow the instructions printed on the device cartons regarding transportation and storage.
- The storage area temperature should be kept within a temperature range of 5°C to 35°C, and relative humidity should be maintained at between 45% and 75%.
- Do not store devices in the presence of harmful (especially corrosive) gases, or in dusty conditions.
- Use storage areas where there is minimal temperature fluctuation. Rapid temperature changes can cause moisture to form on stored devices, resulting in lead oxidation or corrosion. As a result, the solderability of the leads will be degraded.
- When repacking devices, use anti-static containers.
- Do not allow external forces or loads to be applied to devices while they are in storage.
- If devices have been stored for more than two years, their electrical characteristics should be tested and their leads should be tested for ease of soldering before they are used.



3.2.2 Moisture-proof packing

Moisture-proof packing should be handled with care. The handling procedure specified for each packing type should be followed scrupulously. If the proper procedures are not followed, the quality and reliability of devices may be degraded. This section describes general precautions for handling moisture-proof packing. Since the details may differ from device to device, refer also to the relevant individual datasheets or databook.



(1) General precautions

Follow the instructions printed on the device cartons regarding transportation and storage.

- Do not drop or toss device packing. The laminated aluminum material in it can be rendered ineffective by rough handling.
- The storage area temperature should be kept within a temperature range of 5°C to 30°C, and relative humidity should be maintained at 90% (max). Use devices within 12 months of the date marked on the package seal.

- If the 12-month storage period has expired, or if the 30% humidity indicator shown in Figure 1 is pink when the packing is opened, it may be advisable, depending on the device and packing type, to back the devices at high temperature to remove any moisture. Please refer to the table below. After the pack has been opened, use the devices in a 5°C to 30°C, 60% RH environment and within the effective usage period listed on the moisture-proof package. If the effective usage period has expired, or if the packing has been stored in a high-humidity environment, back the devices at high temperature.

Packing	Moisture removal
Tray	If the packing bears the "Heatproof" marking or indicates the maximum temperature which it can withstand, bake at 125°C for 20 hours. (Some devices require a different procedure.)
Tube	Transfer devices to trays bearing the "Heatproof" marking or indicating the temperature which they can withstand, or to aluminum tubes before baking at 125°C for 20 hours.
Tape	Devices packed on tape cannot be baked and must be used within the effective usage period after unpacking, as specified on the packing.

- When baking devices, protect the devices from static electricity.
- Moisture indicators can detect the approximate humidity level at a standard temperature of 25°C. 6-point indicators and 3-point indicators are currently in use, but eventually all indicators will be 3-point indicators.

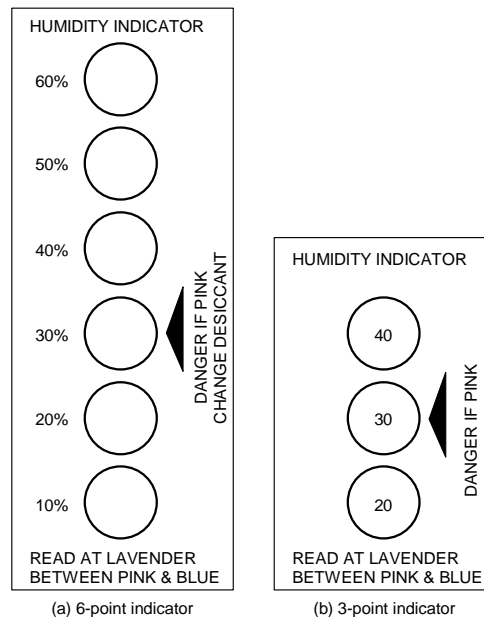


Figure 1 Humidity indicator

3.3 Design

Care must be exercised in the design of electronic equipment to achieve the desired reliability. It is important not only to adhere to specifications concerning absolute maximum ratings and recommended operating conditions, it is also important to consider the overall environment in which equipment will be used, including factors such as the ambient temperature, transient noise and voltage and current surges, as well as mounting conditions which affect device reliability. This section describes some general precautions which you should observe when designing circuits and when mounting devices on printed circuit boards.

For more detailed information about each product family, refer to the relevant individual technical datasheets available from Toshiba.

3.3.1 Absolute maximum ratings

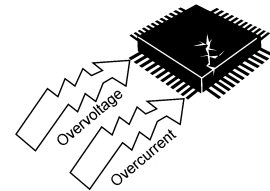
▲ CAUTION

Do not use devices under conditions in which their absolute maximum ratings (e.g. current, voltage, power dissipation or temperature) will be exceeded. A device may break down or its performance may be degraded, causing it to catch fire or explode resulting in injury to the user.

The absolute maximum ratings are rated values which must not be exceeded during operation, even for an instant. Although absolute maximum ratings differ from product to product, they essentially concern the voltage and current at each pin, the allowable power dissipation, and the junction and storage temperatures.

If the voltage or current on any pin exceeds the absolute maximum rating, the device's internal circuitry can become degraded. In the worst case, heat generated in internal circuitry can fuse wiring or cause the semiconductor chip to break down.

If storage or operating temperatures exceed rated values, the package seal can deteriorate or the wires can become disconnected due to the differences between the thermal expansion coefficients of the materials from which the device is constructed.



3.3.2 Recommended operating conditions

The recommended operating conditions for each device are those necessary to guarantee that the device will operate as specified in the datasheet.

If greater reliability is required, derate the device's absolute maximum ratings for voltage, current, power and temperature before using it.

3.3.3 Derating

When incorporating a device into your design, reduce its rated absolute maximum voltage, current, power dissipation and operating temperature in order to ensure high reliability. Since derating differs from application to application, refer to the technical datasheets available for the various devices used in your design.

3.3.4 Unused pins

If unused pins are left open, some devices can exhibit input instability problems, resulting in malfunctions such as abrupt increase in current flow. Similarly, if the unused output pins on a device are connected to the power supply pin, the ground pin or to other output pins, the IC may malfunction or break down.

Since the details regarding the handling of unused pins differ from device to device and from pin to pin, please follow the instructions given in the relevant individual datasheets or databook.

CMOS logic IC inputs, for example, have extremely high impedance. If an input pin is left open, it can easily pick up extraneous noise and become unstable. In this case, if the input voltage level reaches an intermediate level, it is possible that both the P-channel and N-channel transistors will be turned on, allowing unwanted supply current to flow. Therefore, ensure that the unused input pins of a device are connected to the power supply (Vcc) pin or ground (GND) pin of the same device. For details of what to do with the pins of heat sinks, refer to the relevant technical datasheet and databook.

3.3.5 Latch-up

Latch-up is an abnormal condition inherent in CMOS devices, in which Vcc gets shorted to ground. This happens when a parasitic PN-PN junction (thyristor structure) internal to the CMOS chip is turned on, causing a large current of the order of several hundred mA or more to flow between Vcc and GND, eventually causing the device to break down.

Latch-up occurs when the input or output voltage exceeds the rated value, causing a large current to flow in the internal chip, or when the voltage on the Vcc (Vdd) pin exceeds its rated value, forcing the internal chip into a breakdown condition. Once the chip falls into the latch-up state, even though the excess voltage may have been applied only for an instant, the large current continues to flow between Vcc (Vdd) and GND (Vss). This causes the device to heat up and, in extreme cases, to emit gas fumes as well. To avoid this problem, observe the following precautions:

- (1) Do not allow voltage levels on the input and output pins either to rise above Vcc (Vdd) or to fall below GND (Vss). Also, follow any prescribed power-on sequence, so that power is applied gradually or in steps rather than abruptly.
- (2) Do not allow any abnormal noise signals to be applied to the device.
- (3) Set the voltage levels of unused input pins to Vcc (Vdd) or GND (Vss).
- (4) Do not connect output pins to one another.

3.3.6 Input/Output protection

Wired-AND configurations, in which outputs are connected together, cannot be used, since this short-circuits the outputs. Outputs should, of course, never be connected to Vcc (Vdd) or GND (Vss).

Furthermore, ICs with tri-state outputs can undergo performance degradation if a shorted output current is allowed to flow for an extended period of time. Therefore, when designing circuits, make sure that tri-state outputs will not be enabled simultaneously.

3.3.7 Load capacitance

Some devices display increased delay times if the load capacitance is large. Also, large charging and discharging currents will flow in the device, causing noise. Furthermore, since outputs are shorted for a relatively long time, wiring can become fused.

Consult the technical information for the device being used to determine the recommended load capacitance.

3.3.8 Thermal design

The failure rate of semiconductor devices is greatly increased as operating temperatures increase. As shown in Figure 2, the internal thermal stress on a device is the sum of the ambient temperature and the temperature rise due to power dissipation in the device. Therefore, to achieve optimum reliability, observe the following precautions concerning thermal design:

- (1) Keep the ambient temperature (T_a) as low as possible.
- (2) If the device's dynamic power dissipation is relatively large, select the most appropriate circuit board material, and consider the use of heat sinks or of forced air cooling. Such measures will help lower the thermal resistance of the package.
- (3) Derate the device's absolute maximum ratings to minimize thermal stress from power dissipation.

$$\theta_{ja} = \theta_{jc} + \theta_{ca}$$

$$\theta_{ja} = (T_j - T_a) / P$$

$$\theta_{jc} = (T_j - T_c) / P$$

$$\theta_{ca} = (T_c - T_a) / P$$

in which θ_{ja} = thermal resistance between junction and surrounding air ($^{\circ}\text{C}/\text{W}$)

θ_{jc} = thermal resistance between junction and package surface, or internal thermal resistance ($^{\circ}\text{C}/\text{W}$)

θ_{ca} = thermal resistance between package surface and surrounding air, or external thermal resistance ($^{\circ}\text{C}/\text{W}$)

T_j = junction temperature or chip temperature ($^{\circ}\text{C}$)

T_c = package surface temperature or case temperature ($^{\circ}\text{C}$)

T_a = ambient temperature ($^{\circ}\text{C}$)

P = power dissipation (W)

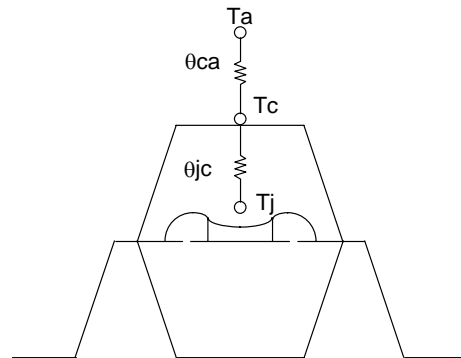


Figure 2 Thermal resistance of package

3.3.9 Interfacing

When connecting inputs and outputs between devices, make sure input voltage (V_{IL}/V_{IH}) and output voltage (V_{OL}/V_{OH}) levels are matched. Otherwise, the devices may malfunction. When connecting devices operating at different supply voltages, such as in a dual-power-supply system, be aware that erroneous power-on and power-off sequences can result in device breakdown. For details of how to interface particular devices, consult the relevant technical datasheets and databooks. If you have any questions or doubts about interfacing, contact your nearest Toshiba office or distributor.

3.3.10 Decoupling

Spike currents generated during switching can cause Vcc (Vdd) and GND (Vss) voltage levels to fluctuate, causing ringing in the output waveform or a delay in response speed. (The power supply and GND wiring impedance is normally 50 Ω to 100 Ω .) For this reason, the impedance of power supply lines with respect to high frequencies must be kept low. This can be accomplished by using thick and short wiring for the Vcc (Vdd) and GND (Vss) lines and by installing decoupling capacitors (of approximately 0.01 μF to 1 μF capacitance) as high-frequency filters between Vcc (Vdd) and GND (Vss) at strategic locations on the printed circuit board.

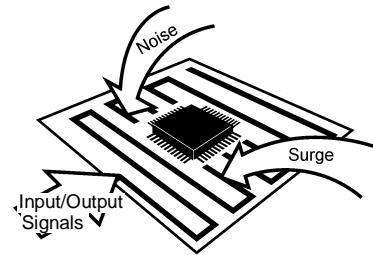
For low-frequency filtering, it is a good idea to install a 10- to 100- μF capacitor on the printed circuit board (one capacitor will suffice). If the capacitance is excessively large, however, (e.g. several thousand μF) latch-up can be a problem. Be sure to choose an appropriate capacitance value.

An important point about wiring is that, in the case of high-speed logic ICs, noise is caused mainly by reflection and crosstalk, or by the power supply impedance. Reflections cause increased signal delay, ringing, overshoot and undershoot, thereby reducing the device's safety margins with respect to noise. To prevent reflections, reduce the wiring length by increasing the device mounting density so as to lower the inductance (L) and capacitance (C) in the wiring. Extreme care must be taken, however, when taking this corrective measure, since it tends to cause crosstalk between the wires. In practice, there must be a trade-off between these two factors.

3.3.11 External noise

Printed circuit boards with long I/O or signal pattern lines are vulnerable to induced noise or surges from outside sources. Consequently, malfunctions or breakdowns can result from overcurrent or overvoltage, depending on the types of device used. To protect against noise, lower the impedance of the pattern line or insert a noise-canceling circuit. Protective measures must also be taken against surges.

For details of the appropriate protective measures for a particular device, consult the relevant databook.



3.3.12 Electromagnetic interference

Widespread use of electrical and electronic equipment in recent years has brought with it radio and TV reception problems due to electromagnetic interference. To use the radio spectrum effectively and to maintain radio communications quality, each country has formulated regulations limiting the amount of electromagnetic interference which can be generated by individual products.

Electromagnetic interference includes conduction noise propagated through power supply and telephone lines, and noise from direct electromagnetic waves radiated by equipment. Different measurement methods and corrective measures are used to assess and counteract each specific type of noise.

Difficulties in controlling electromagnetic interference derive from the fact that there is no method available which allows designers to calculate, at the design stage, the strength of the electromagnetic waves which will emanate from each component in a piece of equipment. For this reason, it is only after the prototype equipment has been completed that the designer can take measurements using a dedicated instrument to determine the strength of electromagnetic interference waves. Yet it is possible during system design to incorporate some measures for the prevention of electromagnetic interference, which can facilitate taking corrective measures once the design has been completed. These include installing shields and noise filters, and increasing the thickness of the power supply wiring patterns on the printed circuit board. One effective

method, for example, is to devise several shielding options during design, and then select the most suitable shielding method based on the results of measurements taken after the prototype has been completed.

3.3.13 Peripheral circuits

In most cases semiconductor devices are used with peripheral circuits and components. The input and output signal voltages and currents in these circuits must be chosen to match the semiconductor device's specifications. The following factors must be taken into account.

- (1) Inappropriate voltages or currents applied to a device's input pins may cause it to operate erratically. Some devices contain pull-up or pull-down resistors. When designing your system, remember to take the effect of this on the voltage and current levels into account.
- (2) The output pins on a device have a predetermined external circuit drive capability. If this drive capability is greater than that required, either incorporate a compensating circuit into your design or carefully select suitable components for use in external circuits.

3.3.14 Safety standards

Each country has safety standards which must be observed. These safety standards include requirements for quality assurance systems and design of device insulation. Such requirements must be fully taken into account to ensure that your design conforms to the applicable safety standards.

3.3.15 Other precautions

- (1) When designing a system, be sure to incorporate fail-safe and other appropriate measures according to the intended purpose of your system. Also, be sure to debug your system under actual board-mounted conditions.
- (2) If a plastic-package device is placed in a strong electric field, surface leakage may occur due to the charge-up phenomenon, resulting in device malfunction. In such cases take appropriate measures to prevent this problem, for example by protecting the package surface with a conductive shield.
- (3) With some microcomputers and MOS memory devices, caution is required when powering on or resetting the device. To ensure that your design does not violate device specifications, consult the relevant databook for each constituent device.
- (4) Ensure that no conductive material or object (such as a metal pin) can drop onto and short the leads of a device mounted on a printed circuit board.

3.4 Inspection, Testing and Evaluation

3.4.1 Grounding



CAUTION Ground all measuring instruments, jigs, tools and soldering irons to earth. Electrical leakage may cause a device to break down or may result in electric shock.

3.4.2 Inspection Sequence

⚠ CAUTION

- ① Do not insert devices in the wrong orientation. Make sure that the positive and negative electrodes of the power supply are correctly connected. Otherwise, the rated maximum current or maximum power dissipation may be exceeded and the device may break down or undergo performance degradation, causing it to catch fire or explode, resulting in injury to the user.
 - ② When conducting any kind of evaluation, inspection or testing using AC power with a peak voltage of 42.4 V or DC power exceeding 60 V, be sure to connect the electrodes or probes of the testing equipment to the device under test before powering it on. Connecting the electrodes or probes of testing equipment to a device while it is powered on may result in electric shock, causing injury.
- (1) Apply voltage to the test jig only after inserting the device securely into it. When applying or removing power, observe the relevant precautions, if any.
 - (2) Make sure that the voltage applied to the device is off before removing the device from the test jig. Otherwise, the device may undergo performance degradation or be destroyed.
 - (3) Make sure that no surge voltages from the measuring equipment are applied to the device.
 - (4) The chips housed in tape carrier packages (TCPs) are bare chips and are therefore exposed. During inspection take care not to crack the chip or cause any flaws in it. Electrical contact may also cause a chip to become faulty. Therefore make sure that nothing comes into electrical contact with the chip.

3.5 Mounting

There are essentially two main types of semiconductor device package: lead insertion and surface mount. During mounting on printed circuit boards, devices can become contaminated by flux or damaged by thermal stress from the soldering process. With surface-mount devices in particular, the most significant problem is thermal stress from solder reflow, when the entire package is subjected to heat. This section describes a recommended temperature profile for each mounting method, as well as general precautions which you should take when mounting devices on printed circuit boards. Note, however, that even for devices with the same package type, the appropriate mounting method varies according to the size of the chip and the size and shape of the lead frame. Therefore, please consult the relevant technical datasheet and databook.

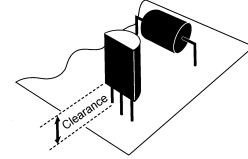
3.5.1 Lead forming

⚠ CAUTION

- ① Always wear protective glasses when cutting the leads of a device with clippers or a similar tool. If you do not, small bits of metal flying off the cut ends may damage your eyes.
- ② Do not touch the tips of device leads. Because some types of device have leads with pointed tips, you may prick your finger.

Semiconductor devices must undergo a process in which the leads are cut and formed before the devices can be mounted on a printed circuit board. If undue stress is applied to the interior of a device during this process, mechanical breakdown or performance degradation can result. This is attributable primarily to differences between the stress on the device's external leads and the stress on the internal leads. If the relative difference is great enough, the device's internal leads, adhesive properties or sealant can be damaged. Observe these precautions during the lead-forming process (this does not apply to surface-mount devices):

- (1) Lead insertion hole intervals on the printed circuit board should match the lead pitch of the device precisely.
- (2) If lead insertion hole intervals on the printed circuit board do not precisely match the lead pitch of the device, do not attempt to forcibly insert devices by pressing on them or by pulling on their leads.
- (3) For the minimum clearance specification between a device and a printed circuit board, refer to the relevant device's datasheet and databook. If necessary, achieve the required clearance by forming the device's leads appropriately. Do not use the spacers which are used to raise devices above the surface of the printed circuit board during soldering to achieve clearance. These spacers normally continue to expand due to heat, even after the solder has begun to solidify; this applies severe stress to the device.
- (4) Observe the following precautions when forming the leads of a device prior to mounting.
 - Use a tool or jig to secure the lead at its base (where the lead meets the device package) while bending so as to avoid mechanical stress to the device. Also avoid bending or stretching device leads repeatedly.
 - Be careful not to damage the lead during lead forming.
 - Follow any other precautions described in the individual datasheets and databooks for each device and package type.



3.5.2 Socket mounting

- (1) When socket mounting devices on a printed circuit board, use sockets which match the inserted device's package.
- (2) Use sockets whose contacts have the appropriate contact pressure. If the contact pressure is insufficient, the socket may not make a perfect contact when the device is repeatedly inserted and removed; if the pressure is excessively high, the device leads may be bent or damaged when they are inserted into or removed from the socket.
- (3) When soldering sockets to the printed circuit board, use sockets whose construction prevents flux from penetrating into the contacts or which allows flux to be completely cleaned off.
- (4) Make sure the coating agent applied to the printed circuit board for moisture-proofing purposes does not stick to the socket contacts.
- (5) If the device leads are severely bent by a socket as it is inserted or removed and you wish to repair the leads so as to continue using the device, make sure that this lead correction is only performed once. Do not use devices whose leads have been corrected more than once.
- (6) If the printed circuit board with the devices mounted on it will be subjected to vibration from external sources, use sockets which have a strong contact pressure so as to prevent the sockets and devices from vibrating relative to one another.

3.5.3 Soldering temperature profile

The soldering temperature and heating time vary from device to device. Therefore, when specifying the mounting conditions, refer to the individual datasheets and databooks for the devices used.

(1) Using a soldering iron

Complete soldering within ten seconds for lead temperatures of up to 260°C, or within three seconds for lead temperatures of up to 350°C.

(2) Using medium infrared ray reflow

- Heating top and bottom with long or medium infrared rays is recommended (see Figure 3).

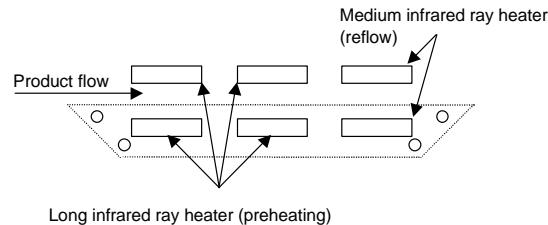


Figure 3 Heating top and bottom with long or medium infrared rays

- Complete the infrared ray reflow process within 30 seconds at a package surface temperature of between 210°C and 240°C.
- Refer to Figure 4 for an example of a good temperature profile for infrared or hot air reflow.

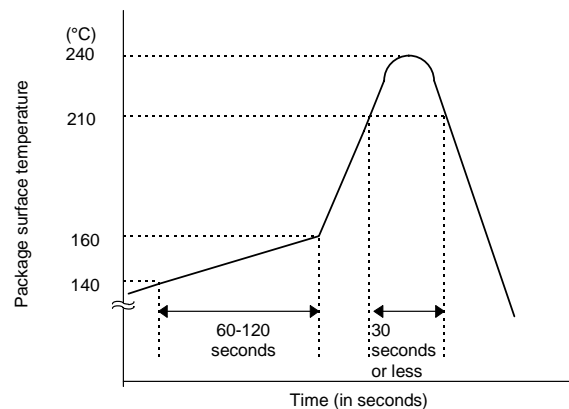


Figure 4 Sample temperature profile for infrared or hot air reflow

(3) Using hot air reflow

- Complete hot air reflow within 30 seconds at a package surface temperature of between 210°C and 240°C.
- For an example of a recommended temperature profile, refer to Figure 4 above.

(4) Using solder flow

- Apply preheating for 60 to 120 seconds at a temperature of 150°C.
- For lead insertion-type packages, complete solder flow within 10 seconds with the temperature at the stopper (or, if there is no stopper, at a location more than 1.5 mm from the body) which does not exceed 260°C.

- For surface-mount packages, complete soldering within 5 seconds at a temperature of 250°C or less in order to prevent thermal stress in the device.
- Figure 5 shows an example of a recommended temperature profile for surface-mount packages using solder flow.

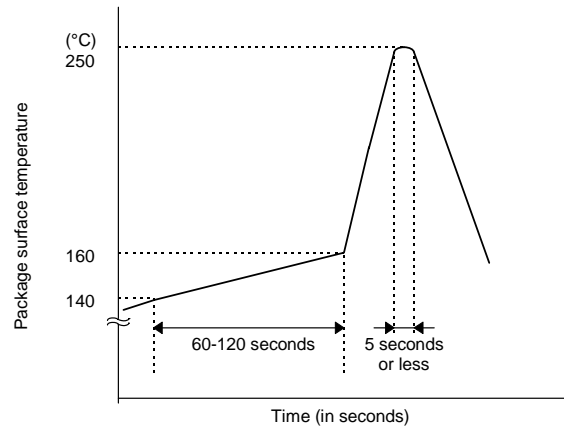


Figure 5 Sample temperature profile for solder flow

3.5.4 Flux cleaning and ultrasonic cleaning

- (1) When cleaning circuit boards to remove flux, make sure that no residual reactive ions such as Na or Cl remain. Note that organic solvents react with water to generate hydrogen chloride and other corrosive gases which can degrade device performance.
- (2) Washing devices with water will not cause any problems. However, make sure that no reactive ions such as sodium and chlorine are left as a residue. Also, be sure to dry devices sufficiently after washing.
- (3) Do not rub device markings with a brush or with your hand during cleaning or while the devices are still wet from the cleaning agent. Doing so can rub off the markings.
- (4) The dip cleaning, shower cleaning and steam cleaning processes all involve the chemical action of a solvent. Use only recommended solvents for these cleaning methods. When immersing devices in a solvent or steam bath, make sure that the temperature of the liquid is 50°C or below, and that the circuit board is removed from the bath within one minute.
- (5) Ultrasonic cleaning should not be used with hermetically-sealed ceramic packages such as a leadless chip carrier (LCC), pin grid array (PGA) or charge-coupled device (CCD), because the bonding wires can become disconnected due to resonance during the cleaning process. Even if a device package allows ultrasonic cleaning, limit the duration of ultrasonic cleaning to as short a time as possible, since long hours of ultrasonic cleaning degrade the adhesion between the mold resin and the frame material. The following ultrasonic cleaning conditions are recommended:

Frequency: 27 kHz ~ 29 kHz

Ultrasonic output power: 300 W or less (0.25 W/cm² or less)

Cleaning time: 30 seconds or less

Suspend the circuit board in the solvent bath during ultrasonic cleaning in such a way that the ultrasonic vibrator does not come into direct contact with the circuit board or the device.

3.5.5 No cleaning

If analog devices or high-speed devices are used without being cleaned, flux residues may cause minute amounts of leakage between pins. Similarly, dew condensation, which occurs in environments containing residual chlorine when power to the device is on, may cause between-lead leakage or migration. Therefore, Toshiba recommends that these devices be cleaned. However, if the flux used contains only a small amount of halogen (0.05W% or less), the devices may be used without cleaning without any problems.

3.5.6 Mounting chips

Devices delivered in chip form tend to degrade or break under external forces much more easily than plastic-packaged devices. Therefore, caution is required when handling this type of device.

- (1) Mount devices in a properly prepared environment so that chip surfaces will not be exposed to polluted ambient air or other polluted substances.
- (2) When handling chips, be careful not to expose them to static electricity.
In particular, measures must be taken to prevent static damage during the mounting of chips. With this in mind, Toshiba recommend mounting all peripheral parts first and then mounting chips last (after all other components have been mounted).
- (3) Make sure that PCBs (or any other kind of circuit board) on which chips are being mounted do not have any chemical residues on them (such as the chemicals which were used for etching the PCBs).
- (4) When mounting chips on a board, use the method of assembly that is most suitable for maintaining the appropriate electrical, thermal and mechanical properties of the semiconductor devices used.

* For details of devices in chip form, refer to the relevant device's individual datasheets.

3.5.7 Circuit board coating

When devices are to be used in equipment requiring a high degree of reliability or in extreme environments (where moisture, corrosive gas or dust is present), circuit boards may be coated for protection. However, before doing so, you must carefully consider the possible stress and contamination effects that may result and then choose the coating resin which results in the minimum level of stress to the device.

3.5.8 Repeated device mounting and usage

Do not remount or re-use devices which fall into the categories listed below; these devices may cause significant problems relating to performance and reliability.

- (1) Devices which have been removed from the board after soldering
- (2) Devices which have been inserted in the wrong orientation or which have had reverse current applied
- (3) Devices which have undergone lead forming more than once

3.6 Protecting Devices in the Field

3.6.1 Temperature

Semiconductor devices are generally more sensitive to temperature than are other electronic components. The various electrical characteristics of a semiconductor device are dependent on the ambient temperature at which the device is used. It is therefore necessary to understand the temperature characteristics of a device and to incorporate device derating into circuit design. Note also that if a device is used above its maximum temperature rating, device deterioration is more rapid and it will reach the end of its usable life sooner than expected.

3.6.2 Humidity

Resin-molded devices are sometimes improperly sealed. When these devices are used for an extended period of time in a high-humidity environment, moisture can penetrate into the device and cause chip degradation or malfunction. Furthermore, when devices are mounted on a regular printed circuit board, the impedance between wiring components can decrease under high-humidity conditions. In systems which require a high signal-source impedance, circuit board leakage or leakage between device lead pins can cause malfunctions. The application of a moisture-proof treatment to the device surface should be considered in this case. On the other hand, operation under low-humidity conditions can damage a device due to the occurrence of electrostatic discharge. Unless damp-proofing measures have been specifically taken, use devices only in environments with appropriate ambient moisture levels (i.e. within a relative humidity range of 40% to 60%).

3.6.3 Corrosive gases

Corrosive gases can cause chemical reactions in devices, degrading device characteristics. For example, sulphur-bearing corrosive gases emanating from rubber placed near a device (accompanied by condensation under high-humidity conditions) can corrode a device's leads. The resulting chemical reaction between leads forms foreign particles which can cause electrical leakage.

3.6.4 Radioactive and cosmic rays

Most industrial and consumer semiconductor devices are not designed with protection against radioactive and cosmic rays. Devices used in aerospace equipment or in radioactive environments must therefore be shielded.

3.6.5 Strong electrical and magnetic fields

Devices exposed to strong magnetic fields can undergo a polarization phenomenon in their plastic material, or within the chip, which gives rise to abnormal symptoms such as impedance changes or increased leakage current. Failures have been reported in LSIs mounted near malfunctioning deflection yokes in TV sets. In such cases the device's installation location must be changed or the device must be shielded against the electrical or magnetic field. Shielding against magnetism is especially necessary for devices used in an alternating magnetic field because of the electromotive forces generated in this type of environment.

3.6.6 Interference from light (ultraviolet rays, sunlight, fluorescent lamps and incandescent lamps)

Light striking a semiconductor device generates electromotive force due to photoelectric effects. In some cases the device can malfunction. This is especially true for devices in which the internal chip is exposed. When designing circuits, make sure that devices are protected against incident light from external sources. This problem is not limited to optical semiconductors and EPROMs.

All types of device can be affected by light.

3.6.7 Dust and oil

Just like corrosive gases, dust and oil can cause chemical reactions in devices, which will adversely affect a device's electrical characteristics. To avoid this problem, do not use devices in dusty or oily environments. This is especially important for optical devices because dust and oil can affect a device's optical characteristics as well as its physical integrity and the electrical performance factors mentioned above.

3.6.8 Fire

Semiconductor devices are combustible; they can emit smoke and catch fire if heated sufficiently. When this happens, some devices may generate poisonous gases. Devices should therefore never be used in close proximity to an open flame or a heat-generating body, or near flammable or combustible materials.

3.7 Disposal of devices and packing materials

When discarding unused devices and packing materials, follow all procedures specified by local regulations in order to protect the environment against contamination.

4. Precautions and Usage Considerations Specific to Microcontrollers

4.1 Design

- (1) Using resonators which are not specifically recommended for use

Resonators recommended for use with Toshiba products in microcontroller oscillator applications are listed in Toshiba databooks along with information about oscillation conditions. If you use a resonator not included in this list, please consult Toshiba or the resonator manufacturer concerning the suitability of the device for your application.

- (2) Undefined functions

In some microcontrollers certain instruction code values do not constitute valid processor instructions. Also, it is possible that the values of bits in registers will become undefined. Take care in your applications not to use invalid instructions or to let register bit values become undefined.

[3] Introduction

[3] Introduction

1. General Description

Using the Neuron Chip configures a low-cost control network. The following types of network communications can be used.

- Twisted-pair cable
- Infrared rays
- RF
- Power line
- Coaxial cable

Any of the above can be used via the standard protocol.

The Neuron Chip is equipped with all the necessary functions to process information obtained via network communications, to make decisions, and to generate and transfer output. Figure 1.1 shows the LONWORKS system configuration. The example is a network using the three types of communications: twisted pair, radio frequency, and power line. Figure 1.2 is a node configuration example.

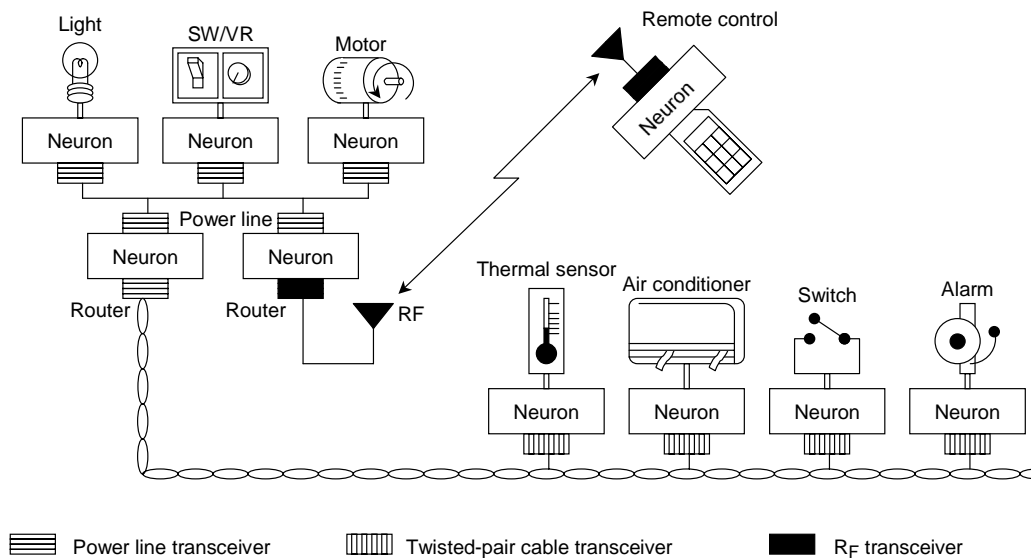


Figure 1.1 Example of LONWORKS System Configuration

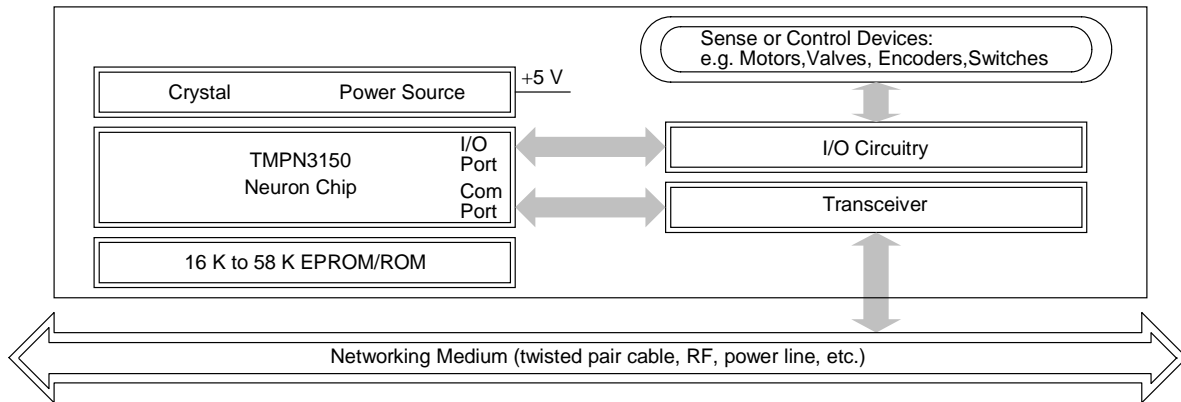


Figure 1.2 The TMPN3150 in a Typical Node Block Diagram Configuration

TMPN3120×× does not require external memory because the LonTalk protocol and a function I/O program are built into a 10-Kbyte (or 16-Kbyte) ROM .

The Neuron Chip comes in two versions: TMPN3150 and TMPN3120××. Both versions contain three 8-bit CPUs.

The first CPU performs the following processes:

- Measuring input parameters
- Timing events
- Making logical decisions
- Runs user applications including output drive

The second CPU performs the following processes:

- Executes LonTalk protocol
- Encodes and decodes messages to be sent over the network

The third CPU performs the following processes:

- Controls network communications ports for send/receive packets

To support these three CPUs, the Neuron Chip incorporates the following types of memory.

- EEPROM
- RAM
- Others

TMPN3120×× incorporates ROM for firmware

TMPN3150 provides an external memory port.

For sending and receiving information, either five pin communications port or eleven pin I/O port can be used. The I/O port supports 34 operating modes, which can be set beforehand by program ([11], Application I/O). Using the I/O port function, a control network which can implement test and control applications at low cost can be configured.

[**5**] LONWORKS Architectural Overview

[5] LONWORKS Architectural Overview

LONWORKS tools and components form a complete platform for implementing intelligent distributed sense and control applications based on LONWORKS technology. These applications consist of intelligent nodes that interact with their environment and communicate with one another over a variety of communications media using a common, message-based network protocol.

LONWORKS technology includes all of the elements required to design, deploy and support intelligent distributed control systems. Specifically, LONWORKS technology includes the following tools and components:

- TMPN3150/3120×× Neuron Chips and associated firmware including support for the LonTalk protocol
- LONWORKS transceivers providing the physical connection between the Neuron Chip and the communications medium
- LONWORKS I/O modules integrating a Neuron Chip and transceiver with external memory and a crystal
- LONWORKS routers for communications between devices on different communication channels of a LONWORKS network
- LONWORKS network interfaces for attaching a non-Neuron processor to a LONWORKS network
- LNS network services tools for installation, configuration, diagnostics, maintenance, control, and monitoring of LONWORKS networks
- Development tools for application development with Neuron Chips

The minimum control unit within the LONWORKS control network is node or LONWORKS device, which consists of the following:

- Neuron Chip
- Sensor and control unit
- Transceiver
- Power supply

Figure 1.1 is a node configuration.

Echelon manufactures the following LONWORKS products which support the Neuron Chips.

- Transceiver
- Router
- LonMaker and LNS products
- etc.
- Control module
- Network interface
- Development tool

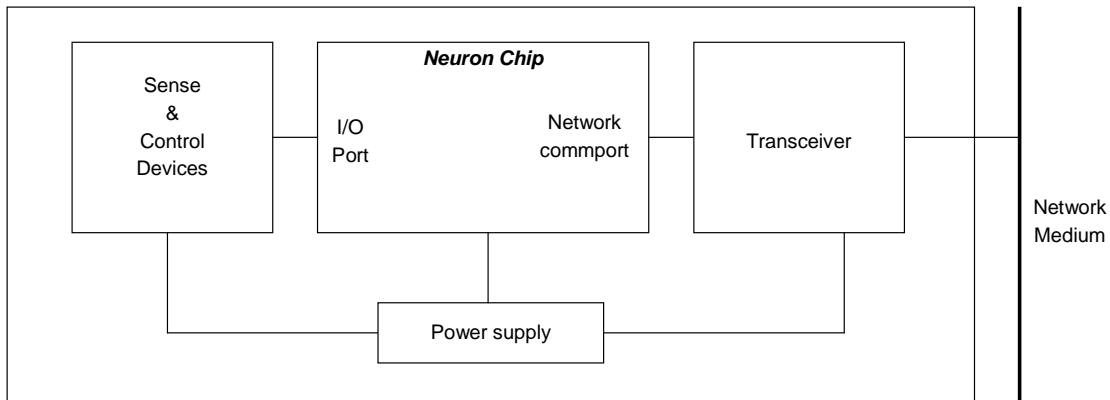


Figure 1.1 Block Diagram of a Typical LONWORKS Node

Applications

Listed below are the main applications of the LONWORKS.

- Measuring
- Machine automation
- Process control
- Diagnosis equipment
- Environment monitor/control
- Power supply/control
- Home electronic appliances
- Discrete control
- Lighting control
- Building automation
- Security system
- Robot
- Home automation

[6] CPUs

[6] CPUs

The TMPN3150 and TMPN3120×× contain three identical 8-bit central processing units (CPUs) which are dedicated to the following functions (see Figure 1.1).

CPU-1 is the Media Access Control (MAC) CPU that handles layers one and two of the seven-layer LonTalk protocol stack. CPU-1 processing includes driving the communications sub-system hardware as well as executing the media access algorithm. CPU-1 communicates with CPU-2 using network buffers located in shared memory (see Figure 1.1).

CPU-2 is the Network CPU which implements layers three through six of the LonTalk protocol stack. It handles network variable processing, addressing, transaction processing, authentication, background diagnostics, software timers and network management. CPU-2 uses network buffers to communicate with CPU-1, and application buffers to communicate with CPU-3 (see Figure 1.1). The buffers are also located in shared memory. Access to them is mediated with hardware semaphores to resolve contention when updating shared data.

CPU-3 is the Application CPU. It runs code written by the user, together with the operating system services called by applications code. The programming language used by the application programmer is Neuron C, a derivative of the ANSI C language optimized and enhanced for LONWORKS distributed control applications. The major enhancements are the following:

- A built-in multi-tasking scheduler that allows the applications programmer to express logically parallel event-driven tasks in a natural way, and to control priority-based execution of these tasks.
- A declarative syntax for input/output objects directly mapping into the input/output capabilities of the TMPN3150/3120×× (see section 8 for details).
- A declarative syntax for network variables, which are Neuron C language objects whose values are automatically propagated over the network whenever new values are assigned to them.
- A declarative syntax for millisecond and second timer objects which activate user tasks on expiration.
- A library of callable functions which can perform event checking, manage input/output activities, send and receive messages across the network, and control miscellaneous functions of the TMPN3150/3120××.

The support for all these capabilities is part of the TMPN3150/3120×× firmware and does not need to be programmed by the user.

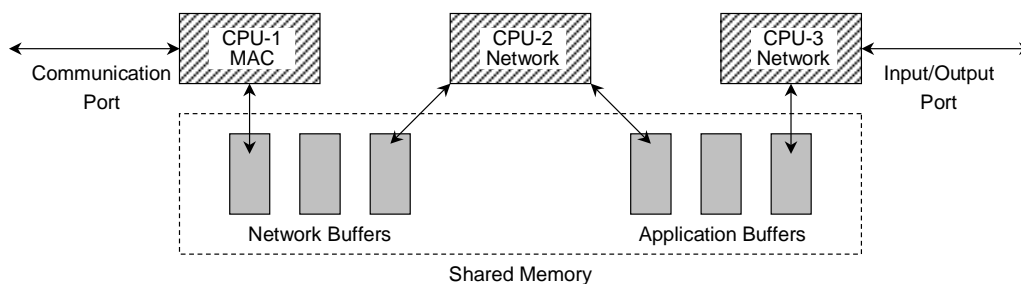


Figure 1.1 TMPN3150/3120×× Processor Organization

Each of the three identical CPUs has its own register set (Table 1.1), but all three CPUs share data and address ALUs and memory access circuitry (Figure 1.2). Each CPU minor cycle consists of *three system clock cycles*, or phases. In turn, each system clock cycle is comprised of two input clock cycles. The minor cycles of the three CPUs are offset from one another by one system clock cycle, so that each CPU can access memory and ALUs once during each minor cycle. Figure 1.2 shows the active elements for each CPU 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.

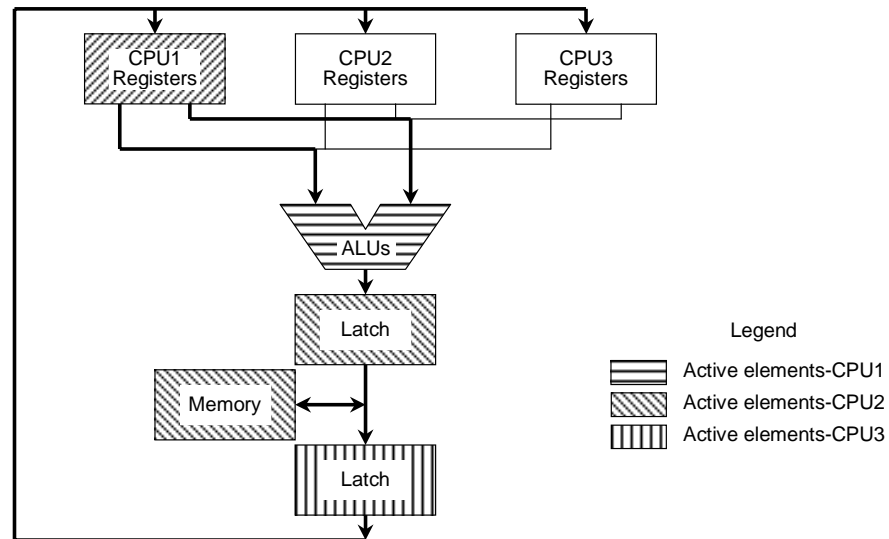


Figure 1.2 TMPN3150/3120xx Processor Organization

Table 1.1 CPU Register Set

Mnemonic	Bits	Contents
FLAGS	8	CPU flags
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

Figure 1.3 shows the layout of a base page, which may be up to 256 bytes long. Each of the three processors uses a different base page, whose address is given by the contents of the BP register of that processor. The top of the data stack is in the 8-bit TOS register, and the next element in the data stack is within the base page, at the location of 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.

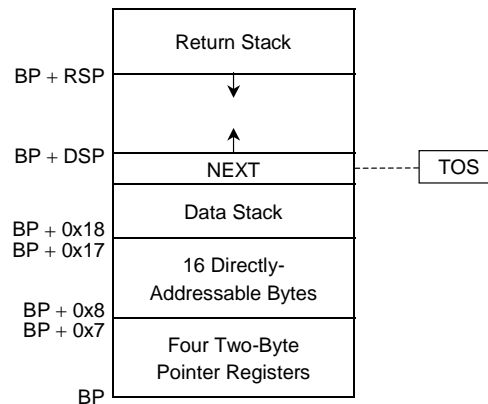


Figure 1.3 Base Page Memory Layout

The CPUs employ a stack architecture that results in very efficient memory utilization. Each CPU has two stacks. The data stack contains byte-wise operands. The return stack contains the return addresses from CALL instructions and may also be used for temporary data storage.

A CPU minor cycle is three system clock cycles, or six input clock cycles. Most instructions take between one and seven CPU minor cycles to execute. At a input clock frequency of 10 MHz, instructions execute in 0.6 to 4.2 micro seconds. The formula for instruction time is:

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

Table 1.2, Table 1.3 and Table 1.4 list the CPU instructions and their cycle counts. This is provided for purposes of calculating times for response to events on the I/O pins. ALL PROGRAMMING OF THE NEURON CHIP IS DONE WITH NEURON C USING A DEVELOPMENT TOOL.

Table 1.2 Program Control Instructions

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 1.3 Memory/Stack Instructions

Mnemonic	# cycles	Description
DROP NEXT	2	Drop next element from data stack
DROP [RSP]	2	Drop top of return stack
DROP TOS	3	Drop top of stack
PUSH TOS	3	Push top of stack
ALLOC #literal	3	Move data stack pointer by <i>literal</i> value
PUSH cpureg	4	Any CPU register except top of stack
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	Push data on top of return stack
PUSHS/PUSH #literal	4	Push 3 or 8-bit <i>literal</i> 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 page pointer register
PUSH/POP [DSP] [D]	5	Contents of DSP minus displacement (1-8)
DROP R NEXT	5	Drop next element from data stack and return
PUSHD #literal	6	Push 16-bit <i>literal</i> value
PUSHD/POPD [PTR]	6	16-bit indirect through base page pointer (0-3)
PUSH/POP [PTR] [TOS]	6	Indirect through base page pointer plus TOS
DEALLOC #literal	6	Drop <i>literal</i> bytes from stack and return
PUSH/POP [PTR] [D]	7	Indirect through base page pointer plus displacement.
PUSH/POP absolute	7	Push memory data to stack
IN/OUT	7 + 4n	Transfer <i>n</i> bytes to I/O

Table 1.4 ALU Instructions

Mnemonic	# cycles	Operation
INC/DEC/NOT	2	Increment/decrement/negate TOS
ROLC/RORC	2	Rotate left/right TOS through carry
SHL/SHR	2	Logical left/right shift TOS
SHLA/SHRA	2	Arithmetic left/right shift TOS
ADD/AND/OR/XOR #literal	3	Operate with <i>literal</i> on TOS
ADD/AND/OR/XOR	4	Operate with next element on TOS
ADC/SBC/SUB/XCH	4	
INC [PTR]	6	Increment base page pointer (0-3)
(ADD/AND/OR/XOR)_R	7	Operate with next element on TOS and return

[**7**] Memory

[7] Memory

The various components of TMPN3150/3120×× memory are described below.

1. EEPROM

All versions of the Neuron Chip have internal EEPROM containing:

- Network configuration and addressing information
- Unique 48-bit Neuron Chip identification code (the neuron ID)
- User-written application code and read-mostly data

User data in EEPROM can be written under program control. The Neuron Chip uses an on-board charge pump to generate the required programming voltage. The charge pump operation is transparent to the user. It takes 20 ms to erase and write one byte. EEPROM may be written 10,000 times with no data loss.

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

For both the Neuron 3120×× Chips and the Neuron 3150 Chip, the EEPROM stores installation-specific information such as network addresses and communication parameters. For the Neuron 3120×× Chips, EEPROM memory also stores the application program generated by the development toll. The application code for the Neuron 3150 Chip may be stored either on-chip in the EEPROM memory, or off-chip in external memory, or both.

Each Neuron Chip has a unique 48-bit identifier, the Neuron ID, that is permanently written into the EEPROM during manufacture.

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. There is a built-in low voltage detector which holds the chip in reset mode below a certain voltage. This helps prevent EEPROM data corruption, although additional external protection may be appropriate. See the section on electrical characteristics for low voltage detector parameters.

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

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

Model	EEPROM Use (bytes)	Segment Size (bytes)
TMPN3150××	417 code or data	2
TMPN3120FE3MG	1961 code or data	8
TMPN3120FE5MG	2985 code or data	16

The segment size is the smallest unit of EEPROM that is allocated for user data, and therefore provides a measure of the amount of fragmentation possible. For example, if there is a single one-byte user variable used in a Neuron 3120FE3MG Chip node, then eight EEPROM bytes are allocated for variable space.

If the supply voltage for the Neuron Chip is not within the specified range, data written to EEPROM or data stored in EEPROM cannot be guaranteed. Make sure that supply voltage is within the specified range.

A protection function (digital) is incorporated to prevent data in EEPROM from being destroyed at a power cut. When supply voltage (VDD) drops below the recommended operating range, it is important to set the ~RESET pin to Low level to prevent data from being destroyed. To further secure the reset status, Toshiba recommends use of an external reset IC. For details, see Chapter 12, Reset Circuit Diagram.

2. RAM

The TMPN3120×× and TMPN3150 contain a internal RAM. The RAM is used to store:

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

The RAM state is retained as long as power is applied to the chip, even in “sleep” mode. However, when the node is reset, the RAM is cleared.

3. ROM

All TMPN3120×× chips include a on chip ROM. This memory stores the Neuron Chip firmware, including.

- LonTalk protocol code
- Event-driven task scheduler
- Application function libraries

4. External Memory

The TMPN3150 does not include any on-chip ROM. Instead, it allows addressing of up to 59,392 bytes of external memory. External memory is used for:

- Application program and data (up to 43,008 bytes)
- Neuron Chip firmware and reserved space (16,348 bytes)

The 43,008 bytes of space available for application program and data may also contain additional network buffers and application buffers.

The external memory space can be occupied by combinations of RAM, ROM, PROM, EPROM, EEPROM or flash memory in 256-byte increments. The memory maps are shown in Figure 4.1 to Figure 4.3.

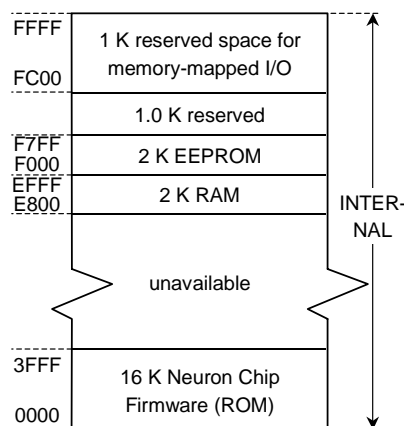


Figure 4.1 TMPN3120FE3MG

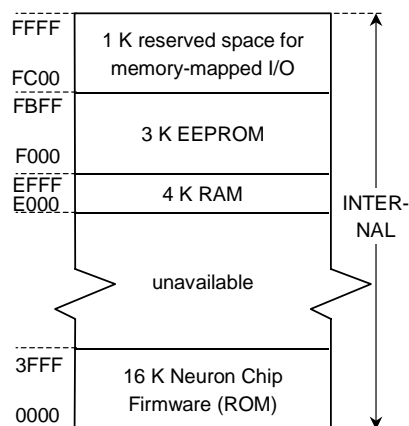


Figure 4.2 TMPN3120FE5MG

The external memory bus has eight bidirectional data lines, sixteen address lines and two control outputs driven by the TMPN3150.

See Section 2, *Memory Interface Timing Specifications*, for detailed timing specifications.

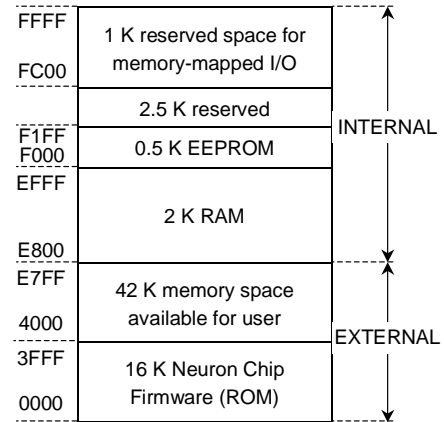


Figure 4.3 TMPN3150 Memory Map

Table 4.1 TMPN3150 External Memory Interface Pins

Pin Designation	Direction	Function
A0 to A15	Output	Address pins
D0 to D7	Input/Output	Data pins
\sim E	Output	Enable clock
R/ \sim W	Output	Read/ \sim Write Select

The Enable Clock (\sim E) runs at the system clock rate, which is one half the input clock rate. The Enable Clock pin is low whenever data are being transferred between the TMPN3150 and external memory. All memory, both internal and external, may be accessed by any of the three CPUs at an appropriate phase of the instruction cycle. Since the instruction cycles of the three CPUs are offset by one system clock cycle with respect to each other, the memory bus is in use by only one CPU at a time.

Figure 4.4 illustrates an example of the TMPN3150 connected to a read-only memory. In this example, **note that when A15 is low, the TMPN3150 accesses external memory at addresses 0000 to 7FFF**. Pin \sim E enables the external memory drivers as appropriate. Pin R/ \sim W is not used. For memory interface timing, see the electrical specifications section of this document.

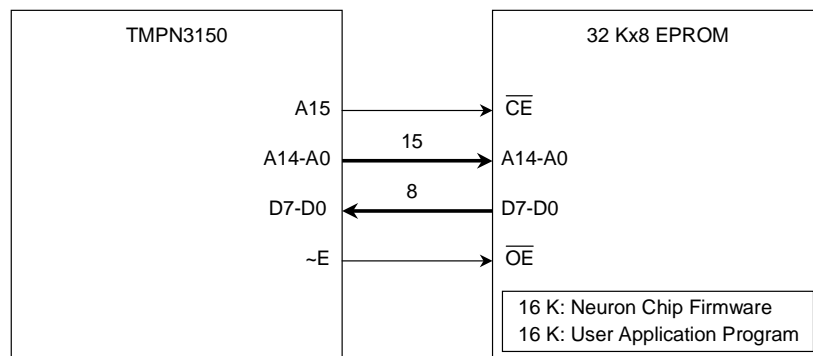


Figure 4.4 Connecting the TMPN3150 to an EPROM

[8] LonTalk Protocol

[8] LonTalk Protocol

The Neuron Chip implements a complete networking protocol using all three on-chip CPUs. This networking protocol follows the International Standards Organization Open System Interconnect (ISO OSI) reference model for network protocols, which support flexible addressing and multiple communications channels on a single network. This protocol is defined in ANSI approved standard EIA/CEA-709.1-A-1999. Application programs running on LONWORKS nodes use this protocol to communicate with applications running on other LONWORKS nodes elsewhere on the same network. See Sections 2 and 3 for details of how the protocol is used by application-level objects called network variables and message tags. The processors in the Neuron Chip are used to execute the protocol software along with the application program. Table 1.1 shows the mapping of LonTalk services onto the 7-layer OSI reference model, and the assignment of the layers to the three CPUs.

Table 1.1 LonTalk Protocol Layering

	OSI Layer	Purpose	Services Provided	CPU
7	Application	Application compatibility	LONMARK objects, configuration properties, Standard Network Variable Types (SNVTs), file transfer	APP
6	Presentation	Data interpretation	Network variables, application messages, foreign frame transmission, network interface	NET
5	Session	Remote actions	Request/response, authentication, network services	NET
4	Transport	End-to-end reliability	Acknowledged and unacknowledged messages, common ordering, duplicate detection	NET
3	NetWork	Destination addressing	Unicast and multicast addressing, routing information	NET
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 the following:

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 cable, powerline, radio-frequency, infrared rays, coaxial cable and fiber optic cable.

2. Support for Multiple Communication Channels

A Channel is a physical transport medium for packets. A network may be composed of one or more channels. In order for packets to be transferred from one channel to another, a device called a router or repeater is used to connect the two channels. A router includes two transceivers to communicate between the two channels. The LonTalk protocol supports such a device so that multi-media networks may be constructed, and network loading can be optimized by localizing traffic.

3. Communication 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 kbits/s.* Channel throughput depends on the bit rate, oscillator frequencies and accuracy, transceiver characteristics, the 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 size of the data field for a network variable update or an explicit message. The maximum packet size is 255 bytes including data, addressing, and protocol overhead.

*: 2,500 kbits/s at 20 MHz only for Single-Ended mode.

4. LonTalk Addressing Limits

The top level of the addressing hierarchy is a domain. For example, if different network applications are implemented on a shared communications medium such as RF, different domain identifiers can be used to keep the applications completely separate. The domain identifier is selectable to be 0, 1, 3 or 6 bytes long. 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 each side of it, and forwards packets accordingly.

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

Nodes may also be grouped. Groups of nodes may span multiple subnets within a domain. Groups may span different transmission media as well as different channels. Up to 256 groups may be specified within a domain, and up to 64 nodes may be in a group for acknowledged or request/response 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 a piece of information using a single message on the network.

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

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

Nodes are addressed using one of five addressing formats:

<i>Address Information Specified</i>	<i>Nodes Addressed</i>
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
Domain, Subnet, Neuron ID	Specific physical node

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

5. Message Services

The LonTalk protocol offers four basic types of message service. The first two service types are acknowledged end-to-end, and include the following:

Acknowledged (ACKD), 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 CPU without intervention of the application. Transaction IDs keep track of messages and acknowledgments so that an application does not receive duplicate messages.

Request/Response (REQUEST), 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 remaining two service types are unacknowledged. They are the following:

Repeated (UNACKD_RPT), 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 multicasting to large groups of nodes, a situation in which the traffic generated by all the responses would otherwise overload the network.

Unacknowledged (UNACKD), where a message is sent once to a node or group of nodes, and no response is expected. This is typically used when the highest attainable 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 occasional 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 receive transaction database in each node.

6. Authentication

The LonTalk protocol supports authenticated messages, which allow the receivers of a message to determine whether the sender is authorized to send that message. This is used to prevent unauthorized access to, or control of, nodes and their applications. Authentication is implemented by distributing 48-bit keys, one per domain, 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. This key is distinct from the node's Neuron ID.

When an authenticated message is sent, the receiver challenges the sender to provide authentication, using a different random number as a challenge every time. The sender then responds by transforming the challenge, using the authentication key along with the data in the original message. 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 the key, even if the challenge and the reply are both known. The use of authentication is configurable individually for each network variable connection. In addition, network management transactions may optionally be authenticated.

7. Priority

The LonTalk protocol offers an optional priority mechanism to improve the response time for 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 dedicated bandwidth is available at the end of each packet for priority access without any contention for the channel.

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.

9. Collision Detection

With communications transceivers that support hardware collision detection, the LonTalk protocol supports detection of collisions and automatic retransmission. This allows a node to retransmit the packet much sooner than if one were to rely solely on upper layer timeouts. Packet retransmission is subject to normal media access delays. The remaining details of collision detection operation vary as a function of the communication port mode.

In direct mode (i.e., differential or single-ended), collisions may be detected as early as 25% into the preamble up through the end of the packet (the preamble is defined in section 1). Detection of a collision does not normally terminate packet transmission. However, packet transmission can optionally 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., if 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 packet successfully transmitted.

10. Data Interpretation

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

11. Network Management and Diagnostic Services

Network management and diagnostic services are provided by two special classes of messages that are processed by every LONWORKS node. These messages are defined in Appendix B.

[9] Network Communication Port

[9] Network Communication Port

There are three modes of operation for the Neuron Chip's 5-pin communications port. They are Single-Ended, Differential and Special-Purpose. Table 1.1 and Figure 1.1 summarizes the pin assignments for each mode.

Table 1.1 Network Communication Port Pin Assignments

Pin	Single-Ended Mode	Differential Mode	Special-Purpose Mode
CP ₀	Data input	+Data input	R _X input
CP ₁	Data output	-Data input	T _X output
CP ₂	Transmit Enable output	+Data output	Bit clock output
CP ₃	~Sleep (~power-down) output	-Data output	~Sleep output or Wake-up input
CP ₄	~Collision Detect input	~Collision Detect input	Frame clock output

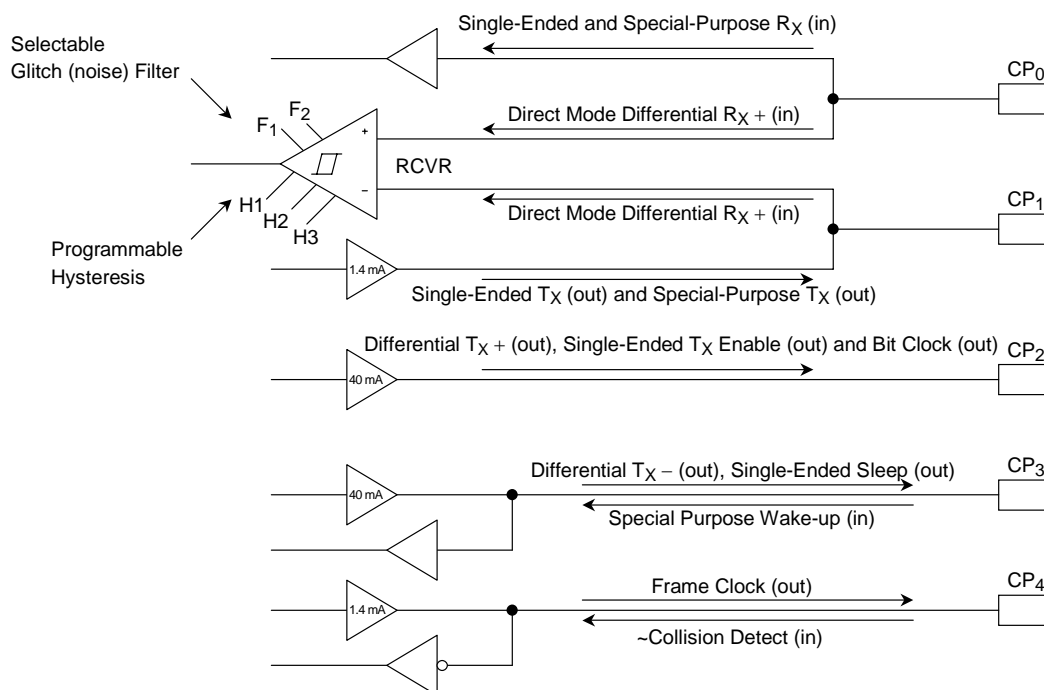


Figure 1.1 Internal Transceiver Block Diagram

Single-Ended and Differential modes use Differential Manchester encoding which is a widely used and reliable format for transmitting data over various media. Differential Manchester coding is polarity-insensitive. Thus, reversal of polarity in the communication link will not affect data reception. The available network bit rates for single-ended and differential modes are given in Table 1.2, as a function of the Neuron Chip's input clock rate.

Table 1.2 Single-Ended and Differential Network Data Rates

Network Bit Rate (kbps)	Minimum Input Clock Frequency (MHz)	Maximum Input Clock Frequency (MHz)
(2500*)	(20.0*)	(20.0*)
1250	10.0	10.0 (20*)
625	5.0	10.0 (20*)
312.5	2.5	10.0 (20*)
156.3	1.25	10.0 (20*)
78.1	0.625	10.0 (20*)
39.1	0.625	10.0 (20*)
19.5	0.625	10.0 (20*)
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

*: These values apply in Single-Ended Mode only to products for which a 20 MHz input clock is available.

1. Single-Ended Mode

The Single-Ended mode is most commonly used with external active transceivers interfacing to media such as RF, Infrared (IR), fiber optics, and coaxial cable. Using an external EIA-485 transceiver IC, the Neuron Chip also communicates over twisted-pair cable.

Figure 1.2 shows the communications port configuration for the Single-Ended mode of operation. Data communication occurs via the single-Ended (with respect to VSS) input and output buffers on pins CP0 and CP1. Single-Ended mode contains an active low sleep output (CP3) which can be used by the transceiver to power down active circuitry when the Neuron Chip goes to sleep.

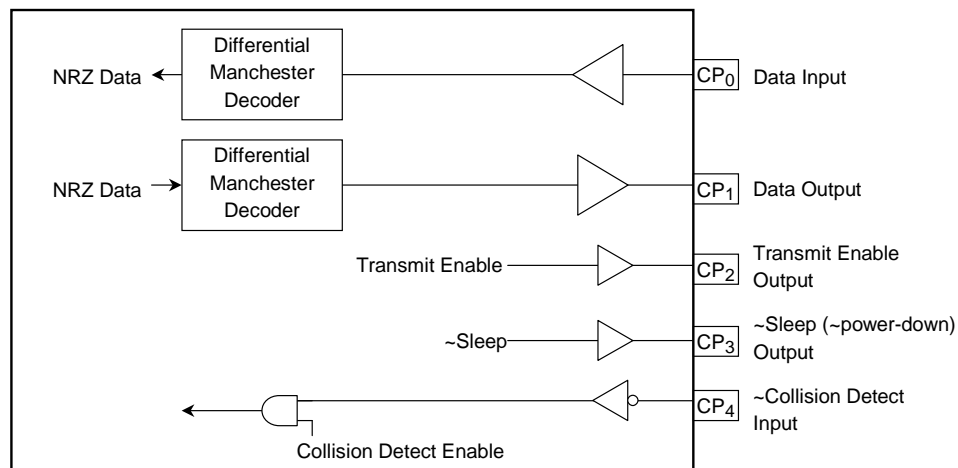


Figure 1.2 Network Communication Port Configuration in Single-Ended Mode

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, referred to as the clock transition. Zero/one data are indicated by the presence or absence of a second transition (the *data transition*) halfway between *clock transitions*. A mid-cell transition indicates a 0. Lack of a mid-cell transition indicates a 1. Figure 1.3 below shows a typical packet where T is the bit period, equal to $1/(\text{bit rate})$. **Note that clock transitions occur at the beginning of a bit period, and, therefore, the last valid bit in the packet does not have a trailing clock edge.**

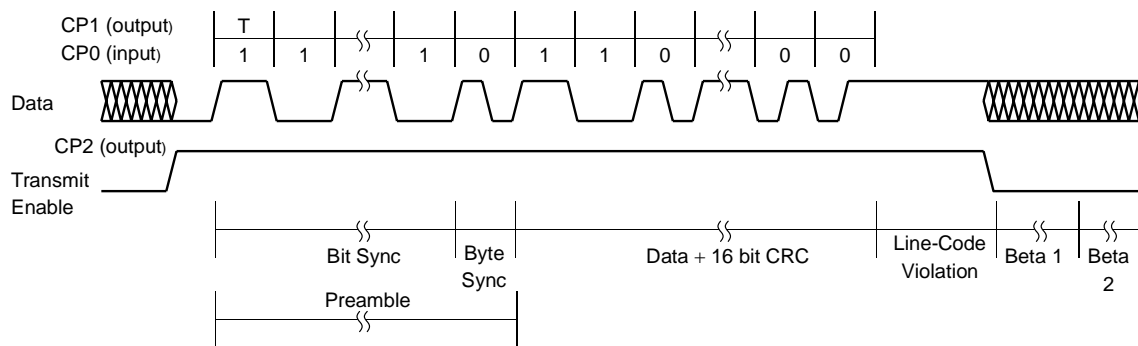


Figure 1.3 Single-Ended Mode Data Format

Before beginning to transmit the packet, the transmitting 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 and Data Output are actively driven at all times in Single-Ended mode.

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 bit-sync field and a byte-sync field. The bit-sync field is a series of Differential Manchester 1's; its duration is user selectable and is at least four bits long. The byte-sync field is a single bit Differential Manchester 0 that marks the end of the preamble, and the beginning of the first byte of the packet.

The Neuron Chip terminates the packet by forcing a Differential Manchester line-code violation; i.e., it holds the data output transitionless long enough for the receiver to recognize an invalid code that signals the end of transmission. The data output can be either high or low for the duration of the line-code violation, depending on the state of the data output after transmitting the last bit. The line-code violation begins after the end of the last CRC bit and lasts for at least 2 bit times. **Note that the last bit does not have a trailing clock edge, so the data output actually remains transitionless for at least 2-1/2 bit times.** The Transmit Enable pin is held active until the end of the line-code violation, and is then released.

As one option, the Neuron Chip can accept 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 occurred or is occurring and that the message must be resent. The node then attempts to re-access the channel.

If the node does not use collision detection, then the only way it can determine that a message has not been received is to request an acknowledgement. When acknowledged service is used, the 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 not routers in the transmission path). If the retry timer times out, the node attempts to re-access the channel.

The idle period between packets includes 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 during which the network can appear to be busy, for example due to ringing on the line
- Minimum interpacket gap—depending on transceiver timings
- Receive end delay—time lag between recognition of the end of the packet by the transmitter and receiver Neuron Chips, respectively. For example, this could be due to buffering in the transceivers, and is typically found in Special-Purpose mode transceivers.

Both priority 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 in $n \times 16$ where “n” has a range of 1 to 63).

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 (P) is fixed for a given channel and can be from 0 to 127. (see Figure 1.4)

The *Beta 2* time is a function of 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 the number of initial preamble bits lost.
- For Special-Purpose mode transceivers, framing delays between the Neuron Chip and the transceiver

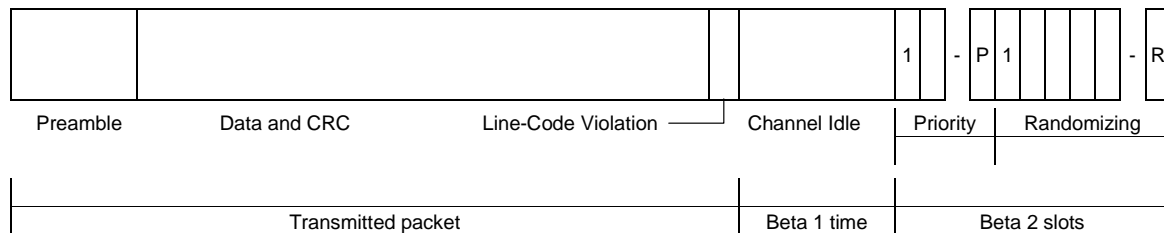


Figure 1.4 Packet Timing

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 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 1.3 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 and the packet will contain errors. The receiving Neuron Chip detects these errors and reports a bad packet. Timing instability of the transitions, known as jitter, may be caused by changes in the communications medium, or instability in the transmitting or receiving nodes' input clocks. The jitter tolerance windows are expressed as fraction of the bit period, T , in Table 1.3.

Table 1.3 Receiver Jitter Tolerance and Line-Code Violation Windows

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

For the receiver to reliably terminate reception of a packet, the line-code violation reception period must have no transitions until the Neuron Chip detects the end of the packet. The receiving Neuron Chip terminates a packet if no clock transitions are detected after the last bit. Table 1.3 shows the minimum duration from the last clock edge to the moment when the Neuron Chip is guaranteed to recognize the line-code violation. **Note that data transitions are allowed during this period (and must fall within the data window).**

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

2. Differential Mode

In 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. The selectable hysteresis and filter permit optimizing receiver performance to line conditions. See the Electrical Specifications (Tables 14.2 and 14.3) for specific values.

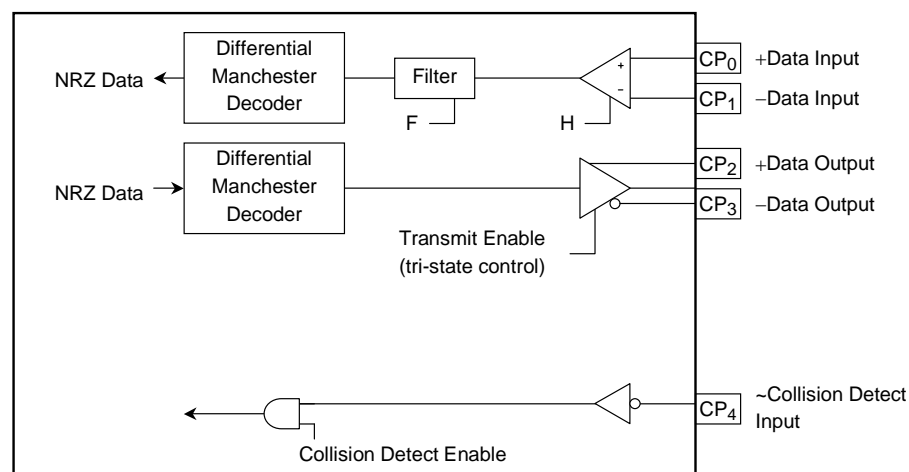


Figure 2.1 Network Communication Port Configuration in Differential Mode

Figure 2.2 shows a typical packet waveform in differential mode. **Note that the packet format is identical to that in Single-Ended mode described earlier.** The starting level for the data output is the inverse of the last received level, to ensure that the first transition occurs on the network as quickly as possible. The coding, jitter tolerance, and minimum time to receive a line-code violation, apply identically to Single-Ended mode.

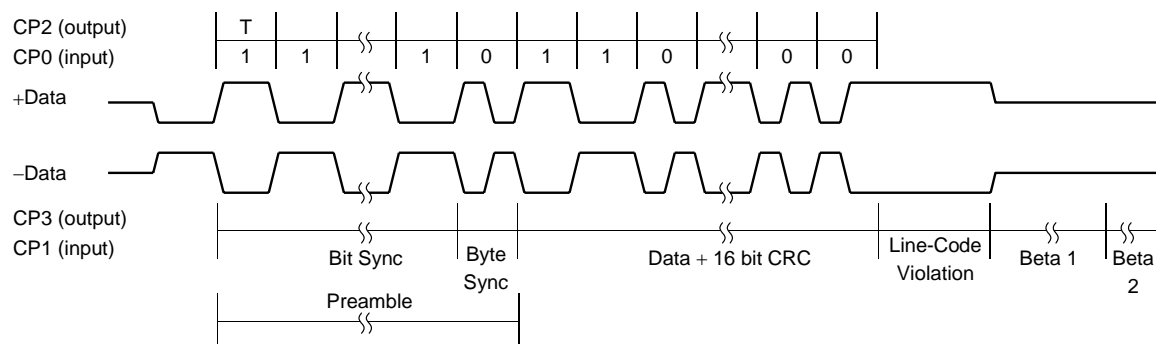


Figure 2.2 Differential Mode Data Format

3. Special-Purpose Mode

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

Patent Notice

The Special Purpose Mode is protected by U.S. Patent No.5,182,746 and foreign patents based on this patent. No express or implied license is granted herein with respect to such patents. If you are interested in obtaining a non-exclusive, royalty free license to these patents please call Echelon at (650) 855-7400 and ask for Contracts Management.

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

There are many other features that can be defined by and incorporated into a special-purpose transceiver, for example:

- Capability to configure various parameters of the transceiver from the Neuron Chip
- Capability 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

While the Special-Purpose mode offers custom features, it is expected that most transceivers will use the Single-Ended mode for most types of media, from coaxial cable to RF to fiber optics. This is because Single-Ended mode offers Differential Manchester encoding, which takes care of clock recovery, whereas Special-Purpose mode does not have this feature.

When the Special-Purpose mode is used, a particular protocol for this mode 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 bit of data, (see Figure 3.1) 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 at network bit rates up to 156 kbps. The maximum network bit rate is 156 kbps due to the overhead associated with the handshaking.

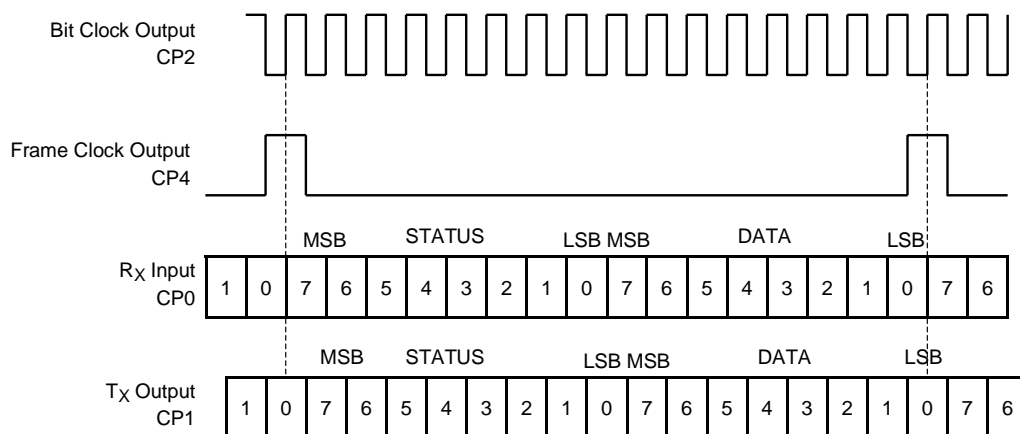


Figure 3.1 Special-Purpose Mode Data Format

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.

When the output on CP3 is high, it 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.

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

The Neuron Chip and the 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 both to sample CP0 and to change CP1 to the next bit. The transceiver should use the rising edge of the 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 samples the most significant bit of the next frame on CP0.

The first eight bits of each frame are interpreted as the status field, and the last eight bits are the data field. The status field is used to control transceiver operation and data transactions between the Neuron Chip and the transceiver. The interpretation of each status bit is shown in Table 3.1 below.

Table 3.1 Special-Purpose Mode Transmit and Receive Status Bits

T_X Output Status Bits

Bit	Name	Function
7	T _X _FLAG	Neuron Chip in the process of transmitting packet
6	T _X _REQ_FLAG	Neuron Chip requesting to transmit on the network
5	T _X _DATA_VALID	Neuron Chip passing network data to the transceiver in this frame
4	Don't Care	Unused
3	T _X _ADDR_R/W	If negated, Neuron Chip writing to the internal transceiver register
2	T _X _ADDR_2	Address bit 2 of internal transceiver register (1 to 7) (*)
1	T _X _ADDR_1	Address bit 1 of internal transceiver register (1 to 7) (*)
0	T _X _ADDR_0	Address bit 0 of internal transceiver register (1 to 7) (*)

*: Note that internal transceiver Register 0 is not valid. Registers 1 through 7 are defined by the transceiver implementation.

R_X Input Status Bits

Bit	Name	Function
7	SET_T _X _FLAG	Transceiver accepts request to transmit packet
6	CLR_T _X _REQ_FLAG	Transceiver acknowledges request to transmit packet
5	R _X _DATA_VALID	Transceiver is passing network data to the Neuron Chip in this frame
4	T _X _DATA_CTS	Transceiver indicates that Neuron Chip is clear to send byte of network data
3	SET_COLL_DET	Transceiver has detected a collision while transmitting the preamble
2	R _X _FLAG	Transceiver has detected a packet on the network
1	RD/WR_ACK	Transceiver acknowledges read/write to internal register
0	T _X _ON	Transceiver is transmitting on the network

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

- Network packet data: Actual data (8 bits at a time) that is to be transmitted or has been received
- Configuration data: Information from the Neuron Chip that tells the transceiver how it is to be set up or configured
- Status data: Parameters reported from the transceiver to the Neuron Chip, when 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 only send packet data 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.

If the transceiver detects a collision it may abort transmission 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 following its assertion of RX_FLAG. Each frame that contains valid data must be marked by asserting RX_DATA_VALID. When the transceiver is finished receiving a packet, it clears RX_FLAG and the Neuron Chip terminates reception of the packet.

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

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

To read a status register, the Neuron Chip asserts TX_ADDR_R/W and indicates the selected register with TX_ADDR_[2:0]. The transceiver must acknowledge that the operation is completed 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.

[**10**] Programming Model

[10] Programming Model

The programming language used to write applications for the Neuron Chip is a derivative of the ANSI 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 exist between Neuron C and ANSI C in the data types supported. The numeric data types supported by Neuron C are the following.

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

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 arithmetic. 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 multi-tasking scheduler, EEPROM variables, and miscellaneous functions. An extended arithmetic function library implementing IEEE754 floating point, 32-bit fixed point, and string operations is also available within Neuron C. For further details on the Neuron C programming language, see the *Neuron C Programmer's Guide*.

1. Timers

An 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 of the Network CPU, and are independent of the Neuron Chip input clock rate. The expiration of a timer is an event that may cause the execution of a user-written task (see below). This event is called timer expires. The value of a timer variable is an unsigned long with a value of (0-65,535). Note that when operating at a 20-MHz input clock, the value of the milli timer variable is a value between 0 to 32,767. The milli timer variable value is set between 0 to 65,535.

Timers can be set to any value in this range at any time by the application program.

2. Network Variables

The application program can declare a special class of static objects called network variables, which may be of input or output class. Assignment of a value to an output network variable causes propagation of that value to all nodes declaring an input variable that is connected to the output network variable. For example, a node that contains a temperature sending 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 tool. The LONWORKS Network Service (LNS) Products 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 bound. Nodes may also update their own binding information for simple networks, without network management tools. Tables containing binding information are in the EEPROM, and hence may be written after the node has been manufactured.

Nodes declaring an input network variable need only refer to that variable to determine the latest value propagated across the network. A node declaring an input network variable may also call the library routine “*poll ()*” to cause the latest value to be propagated to it. The library routine “*is_bound ()*” may be used to check whether a particular network variable is connected to (bound to) a certain other network variable on another node.

Note that declaring network variables within a node’s code occurs at compile time. The binding of the network variable outputs from one node to inputs on other nodes occurs at a later time, which may be 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 for very flexible applications of distributed data to be operated on by the nodes in a system. The programmer need not deal with message buffers, node addressing, request/response/retry processing, or 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 other nodes, and a single output network variable can send data of the same type to an unlimited number of other nodes. Additional network variables can be accessed from Neuron C by explicitly building a network variable update or fetch message 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 the next section. 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 their 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 5 of the LonTalk protocol, and moves layers 6 and 7 application support to the host processor. The LonTalk protocol includes the specification of a standard protocol between the host and network interface, which is called the LONWORKS network interface protocol.

Turn-key LONWORKS network interfaces are available, such as Echelon's PCC-10 PC Card Network Adapter, PCLTA PC LonTalk Adapter, and SLTA Serial LonTalk Adapter, to provide a ready-made network interface. A custom LONWORKS network interface can also be built using the LONWORKS Microprocessor Interface Program (MIP). The MIP is firmware for the Neuron Chip, that transforms the Neuron Chip into a high-performance communications processor for an attached host.

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

Network variable updates may be sent with four classes of service:

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

Network variables may be specified as authenticated, meaning that only authenticated messages are used to transmit their values (see section 6). Network variables may also be specified to have priority, meaning that a priority time slot is used to transmit their values (see section 7). Finally, network variables may be specified as synchronous, in which case all values assigned to the network variable are propagated. Normally when network variable updates are generated faster than they can be propagated, the network variables propagate only their most recent value. Intermediate values may be discarded.

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

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

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

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

```
const struct {
    unsigned domain      : 1;           // domain table reference
    unsigned flex_domain : 1;           // received on flexible domain
    unsigned format      : 6;           // 0 = broadcast, 1 = group,
                                         // 2 = subnet/node, 3 = NEURON ID
                                         // 4 = turnaround

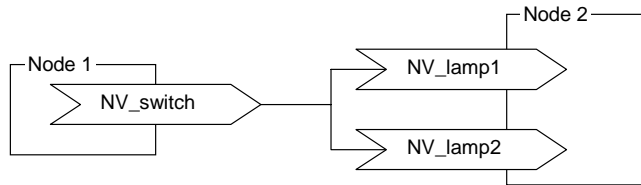
    struct {
        unsigned subnet;
        unsigned      : 1;
        unsigned node  : 7;
    } src_addr;           // source address
    struct {
        unsigned group;   // group destination address
    } dest_addr;
} nv_in_addr;
```

3. Network Variable Aliases

When using network variables, connection between network variables is limited due to restrictions on selector numbers (A.4.1). For example, to connect as shown in Figure 3.1 (a), connection cannot be made due to a restriction on the node 2 selector number. In such a case, two output network variables (NV_switch) can be used to solve the problem; however, the application program must be greatly modified.

To avoid such restrictions, at network variable connection, instead of modifying application programs, use network variable aliases to increase network variables with the same meaning. (example: Figure 3.1 (b))

(a) Example where connection is restricted



(b) Example where alias is used

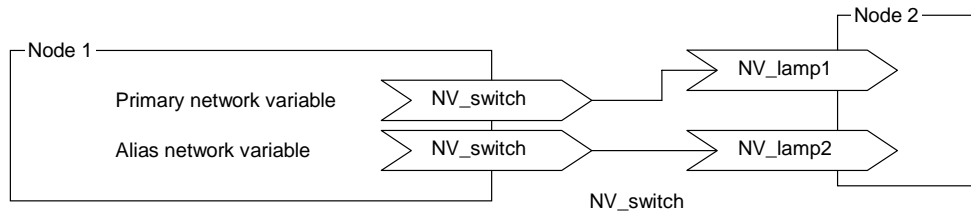


Figure 3.1 Alias Schematic

An alias is an abstract variable for networks controlled by the Neuron Chip firmware.

An alias corresponds to a non-alias network variable called a primary network variable. The alias inherits the attributes of the primary network variable such as data value, direction, and type. The primary network variable value is shared with the corresponding alias.

A network variable update message received by the primary input network variable or by any alias updates the primary network variable value and generates an update event for the primary network variable. A poll request for the primary output network variable or its alias generates a response based on the primary network variable value.

When the application program updates the primary output network variable, the Neuron Chip firmware sends the network variable update message for the related primary network variable and alias on the send node. When the application program starts polling of the primary input network variable, the Neuron Chip firmware sends the network variable poll request for the related primary network variable and alias on the polling node.

Note: ● When using aliases, the number of the alias table entries in the target node applications program must be specified in advance.

```
"#pragma num_alias_table_entries <number>"
```

For details of alias tables, see Chapter A.4.

- Some network management tools such as LonBuilder network manager do not support aliases.

4. Explicit Messages

Since LONWORKS network is based on using a network variable, it becomes the outside for support about this explicit message.

For most applications, network variables allow the most compact and simple possible application software implementation, often resulting in the smallest attainable code size. However, for applications in which data objects larger than 31 bytes need to be transmitted, where request/response service is desired, or when the network variable model is not suitable, then the application can use explicit messaging.

The application program can construct messages containing up to 228 bytes of data, addressed to other nodes or groups of nodes via implicit address connections called message tags. For further details, see the LONWORKS Engineering Bulletin *File Transfer*. Alternately, messages may be explicitly addressed to other nodes using subnet/node, group, broadcast, or Neuron ID addressing.

The messages may be sent under one of four classes of service:

<i>Acknowledged</i>	Acknowledged service with retry
<i>Unacknowledged</i>	Unacknowledged service
<i>Repeated</i>	Unacknowledged repeated service (message sent multiple times)
<i>Request/Response</i>	Request/response service

Request/response service allows the node receiving a message to respond with data in its response messages, as distinct from an acknowledgment which contains no application-level data. When a node receives a request message, it prepares a response to the incoming message and returns it to the original sender using the `resp_send()` call. If the response contains no data aside from the message code, then the network processor handles retries in such a way that the message is only delivered to the destination application once. If the response contains one or more bytes of application data, then the message is delivered to the destination application on each retry. Messages may be specified as authenticated (see section 6). Messages may also be specified to have priority, meaning that they use a priority time slot on the channel if the node has a priority slot allocated to it. In any case, priority messages are always sent by the node before non-priority messages (see section 7). The application program may explicitly assign the destination addresses, or it may use the default address associated with the message tag.

Messages are exchanged between nodes by calling run-time library support routines:

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

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

The following built-in objects are defined for use by application program:

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

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

```

typedef enum {
    ACKD = 0,                // acknowledged
    UNACKD_RPT = 1,          // unacknowledged repeated
    UNACKD = 2,              // unacknowledged
    REQUEST = 3              // 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
    boolean  duplicate;      // 1=> msg is a duplicate
    int      rcvtx;          // receive transaction index
} msg_in;

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

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

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

```

If an explicit address is used to address an outgoing message, the source node application creates an addr data structure in the msg_out buffer. If the destination node application 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 MSG_ADDR.H include file, 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

Use of the buffer allocation functions (`msg_alloc ()` and `msg_alloc_priority ()`) is optional. If they are not used and if buffers are not available when message construction commences, the node enters preemption mode. This suspends the application program except for completion events (e.g., `msg_complets`), incoming message events, and network variable array event processing. If no buffer becomes available within a configurable number of seconds known as the *maximum free buffer wait time*, the node resets. This is known as a *preemption mode timeout*. See Section A.6 for a description of the configuration data structure, where this is defined. Preemption mode can also be entered when synchronous network variables are updated or when the `flush_wait ()` function is used.

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

5. Scheduler

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

Events fall into five classes:

System-Wide Events

<i>reset</i>	Node has been reset
<i>offline</i>	Node has been set off-line
<i>online</i>	Node has been set on-line
<i>flush_completes</i>	Node has complicated preparations to enter sleep mode
<i>wink</i>	Node has received <i>wink</i> network management message with <i>wink</i> subcommand

Input/Output Events

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

Timer Events

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

Message and Network Variable Events

The following events associated with the transmission of network variables and messages are discussed above in Sections 2 and 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, evaluated as TRUE or FALSE

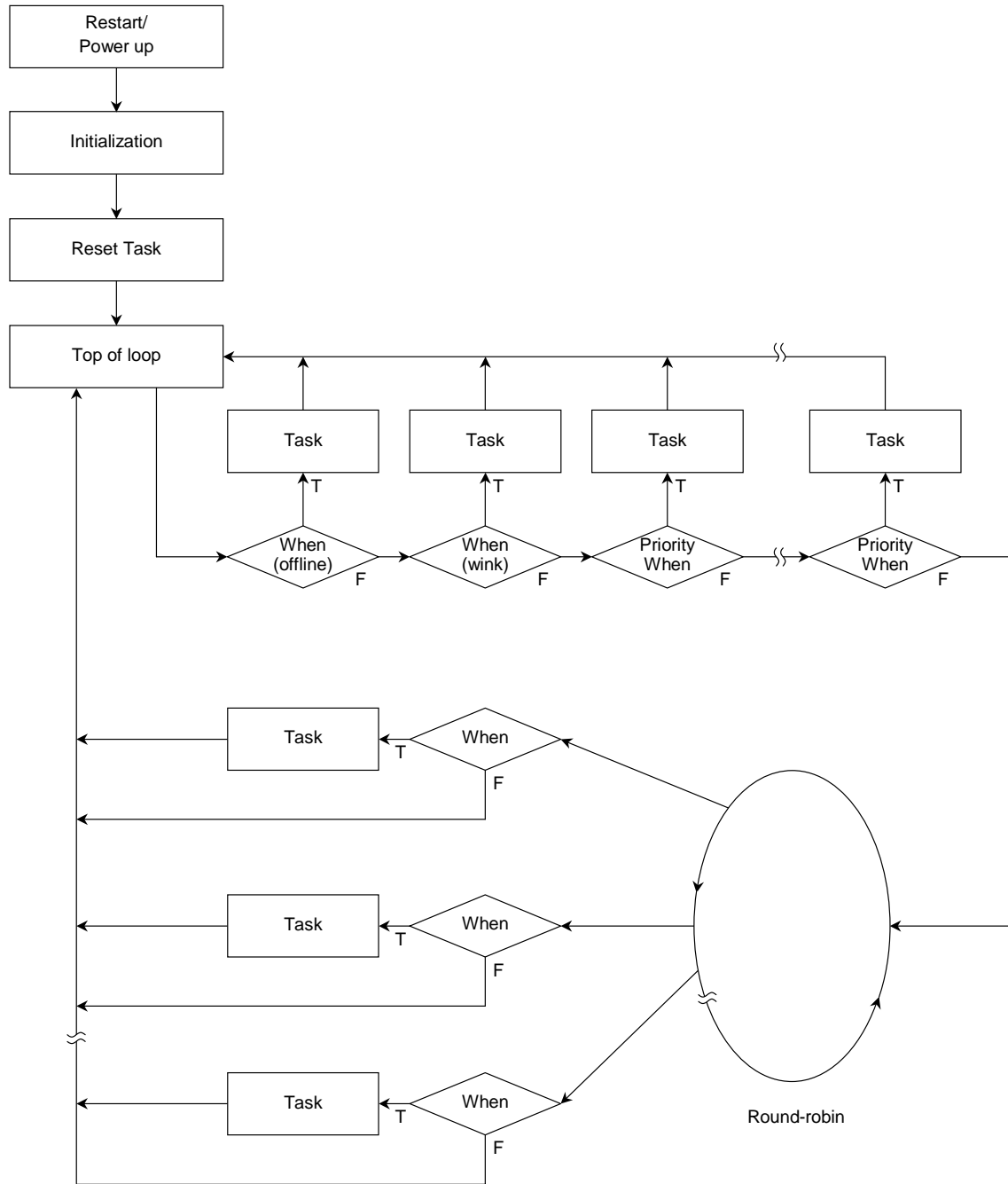


Figure 5.1 Neuron Chip Firmware Scheduling of Nonpriority and Priority When Clauses

6. Additional Functions

The following miscellaneous functions are available from the application program. These functions are built into the Neuron C Compiler, are part of the Neuron Chip system image, or are linked into the application image from a system library.

When a function is used, whether the function is in the system image (within firmware) or linked to the application image (application area is used) depends on the Neuron Chip type.

The table below lists the relations between functions and Neuron Chips.

○: Function is included in the system image

×: Function is linked to the application image

N/A: Function cannot be used

● Execution Control

Function	TMPN3150	TMPN3120 FE3MG/FE5MG
delay ()	○	○
flush ()	○	○
flush_cancel ()	○	○
flush_wait ()	○	○
get_tick_count ()	○	○
go_offline ()	○	○
post_events ()	○	○
power_up ()	○	○
preemption_mode ()	×	○
scaled_delay ()	○	○
sleep ()	○	○
timers_off ()	○	○
watchdog_update ()	○	○

● Network Configuration

Function	TMPN3150	TMPN3120 FE3MG/FE5MG
access_address ()	○	○
access_alias ()	○	○
access_domain ()	○	○
access_nv ()	○	○
addr_table_index ()	○	○
application_restart ()	○	○
go_unconfigured ()	○	○
node_reset ()	○	○
nv_table_index ()	○	○
offline_confirm ()	○	○
update_address ()	○	○
update_alias ()	○	○
update_clone_domain ()	○	○
update_config_date ()	○	○
update_domain ()	○	○
update_nv ()	○	○

● Inter Math

Function	TMPN3150	TMPN3120 FE3MG/FE5MG
abs ()	○	○
bcd2bin ()	○	○
bin2bcd ()	○	○
high_byte ()	○	○
low_byte ()	○	○
make_long ()	○	○
max ()	○	○
min ()	○	○
muldiv ()	○	○
muldiv24 ()	×	○
muldiv24s ()	×	○
muldivs ()	○	○
random ()	○	○
reverse ()	○	○
rotate_long_left ()	×	○
rotate_long_right ()	×	○
rotate_short_left ()	×	○
rotate_short_right ()	×	○
s32_abs ()	×	○
s32_add ()	×	○
s32_cmp ()	×	○

Function	TMPN3150	TMPN3120 FE3MG/FE5MG
s32_dec ()	×	○
s32_div ()	×	○
s32_div2 ()	×	○
s32_eq ()	×	○
s32_from_ascii ()	×	×
s32_from_slong ()	×	○
s32_from_ulong ()	×	×
s32_ge ()	×	×
s32_gt ()	×	×
s32_inc ()	×	×
s32_le ()	×	×
s32_lt ()	×	×
s32_max ()	×	×
s32_min ()	×	×
s32_mul ()	×	×
s32_mul2 ()	×	×
s32_ne ()	×	×
s32_neg ()	×	×
s32_rand ()	×	×
s32_rem ()	×	×
s32_sign ()	×	×
s32_sub ()	×	×
s32_to_ascii ()	×	○
s32_to_slong ()	×	○
s32_to_ulong ()	×	○
swap_bytes ()	○	○

● Floating Point Math

Function	TMPN3150	TMPN3120 FE3MG/FE5MG
fl_abs ()	×	○
fl_add ()	×	○
fl_ceil ()	×	○
fl_cmp ()	×	○
fl_div ()	×	○
fl_div2 ()	×	×
fl_eq ()	×	○
fl_floor ()	×	○
fl_from_ascii ()	×	×
fl_from_s32 ()	×	×
fl_from_slong ()	×	○
fl_from_ulong ()	×	○
fl_ge ()	×	○
fl_gt ()	×	○
fl_le ()	×	○
fl_lt ()	×	○
fl_max ()	×	○
fl_min ()	×	○
fl_mul ()	×	○
fl_mul2 ()	×	×
fl_ne ()	×	○
fl_neg ()	×	○
fl_rand ()	×	○
fl_round ()	×	○
fl_sign ()	×	○
fl_sqrt ()	×	○
fl_sub ()	×	○
fl_to_ascii ()	×	×
fl_to_ascii_fmt ()	×	×
fl_to_s32 ()	×	×
fl_to_slong ()	×	○
fl_to_ulong ()	×	○
fl_trunc ()	×	○

● Strings

Function	TMPN3150	TMPN3120 FE3MG/FE5MG
strcat ()	×	○
strchr ()	×	○
strcmp ()	×	○
strcpy ()	×	○
strlen ()	×	○
strncat ()	×	○
strncmp ()	×	○
strncpy ()	×	○
strrchr ()	×	○

● Utilities

Function	TMPN3150	TMPN3120 FE3MG/FE5MG
ansi_memcpy ()	×	○
ansi_memset ()	×	○
clear_status ()	×	○
clr_bit ()	×	○
crc8 ()	×	○
crc16 ()	×	○
eeprom_memcpy ()	○	○
error_log ()	×	○
memccpy ()	×	○
memchr ()	×	○
memcmp ()	×	○
memcpy ()	○	○
memset ()	○	○
refresh_memory ()	○	○
retrieve_status ()	○	○
retrieve_xcvr_status ()	×	○
set_bit ()	×	○
set_eeprom_lock ()	○	○
tst_bit ()	×	○

● **Input/Output**

Function	TMPN3150	TMPN3120 FE3MG/FE5MG
io_change_init ()	○	○
io_edgelog_preload ()	○	○
io_in () differs depending on the I/O object used		
Dualslope input	○	○
Edgelog input	○	○
Infrared input	○	×
Magcard input	○	×
Neurowire I/O slave mode	○	×
Neurowire I/O with invert option	×	×
Serial input	○	○
Touch I/O	×	×
Wiegand input	×	○
Others	○	○
io_in_ready ()	○	○
io_in_request ()	○	○
io_out () differs depending on the I/O object used		
Bitshift output	○	○
Neurowire I/O slave mode	○	×
Neurowire I/O with invert option	×	×
Serial output	○	○
Touch I/O	×	×
Others	○	○
io_out_ready ()	○	○
io_out_request ()	○	○
io_preserve_input ()	○	○
io_select ()	○	○
io_set_clock ()	○	○
io_set_direction ()	○	○

7. Built-In Variables

<i>activate_service_led</i>	Service LED state control
<i>config_data</i>	Read-only copy of the node's configuration data
<i>input_value</i>	Data read by last explicit <code>io_in()</code> call or by last input/output event (e.g., <code>io_changes</code>)
<i>input_is_new</i>	True if last input from a timer/counter object read an update value
<i>msg_tag_index</i>	Message tag index for the most recent explicit message response or completion event received
<i>nv_array_index</i>	Index of element in network variable array with updated value
<i>nv_in_addr</i>	Source address of the last network variable update
<i>nv_in_index</i>	Network variable index for the most recent network variable update
<i>read_only_data</i>	Copy of node's read only data structure
<i>read_only_data_2</i>	Copy of node's read only data structure extension

8. TMPN3120×× Chip Firmware Extensions

On the TMPN3120×× Chip, all application code is placed in on-chip EEPROM. System library functions specified in Section 6 are linked with the application and are also placed in on-chip EEPROM. In addition, when any of the following Neuron C features are used, object code is brought in from a system library and placed in on-chip EEPROM.

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

[**11**] Application I/O

[11] Application I/O

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. Pins IO4 through IO7 have programmable (on or off) pull-ups. Pins IO0 through IO3 have high current (20 mA) sink capability. All pins (IO0 through IO10) have TTL-level inputs with hysteresis. Pin IO0 through IO7 also have low-level detect latches.

The Neuron Chip has two 16-bit timer/counters on-chip. 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 may be connected 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, on the falling edge, or on 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.

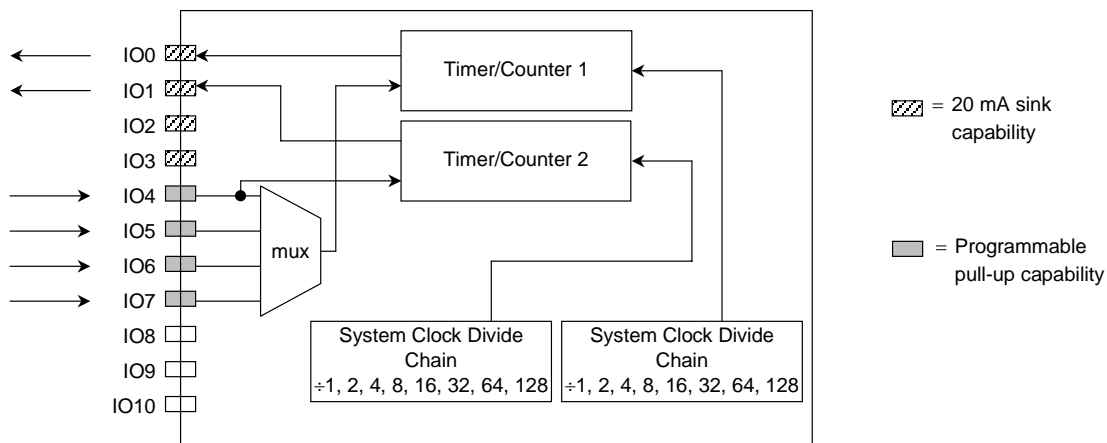


Figure 1.1 Neuron Chip Timer/Counter External Connections

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

Table 1.1 through Table 1.5 list the main I/O objects. Various I/O objects of different types may be used simultaneously. Table 1.6 summarizes the pin configuration for each of the I/O objects. Section 2 contains detailed descriptions of the I/O objects.

The I/O object is subject to change without notice.

Table 1.1 Summary of Direct I/O Objects

Object	Applicable I/O Pins	Input/Output Value
Bit Input	IO0 – IO10	0, 1 binary data
Bit Output	IO0 – IO10	0, 1 binary data
Byte Input	IO0..IO7	0-255 binary data
Byte Output	IO0..IO7	0-255 binary data
Leveldetect Input	IO0 – IO7	Logic zero level detected
Nibble Input	Any adjacent 4 in IO0 – IO7	0-15 binary data
Nibble Output	Any adjacent 4 in IO0 – IO7	0-15 binary data

Table 1.2 Summary of Parallel I/O Objects

I/O Object	Applicable I/O Pins	Input/Output Value
Muxbus I/O	IO0..IO10	Parallel bidirectional port using multiplexed address technique
Parallel I/O	IO0..IO10	Parallel bidirectional handshaking port

Table 1.3 Summary of Serial I/O Objects

I/O Object	Applicable I/O Pins	Input/Output Value
Bitshift Input	Any adjacent pair (except IO7 + IO8)	Up to 16 bits of clocked data
Bitshift Output	Any adjacent pair (except IO7 + IO8)	Up to 16 bits of clocked data
I ² C	IO8 + IO9	Up to 255 bytes of bidirectional serial data
Magcard Input	IO8 + IO9 + one of (IO0 – IO7)	Encoded ISO7811 track 2 data stream from a magnetic card reader
Magtrack 1	IO8 + IO9 + one of (IO0 – IO7)	Encoded ISO3554 track 1 data stream from a magnetic card reader
Neurowire I/O	IO8 + IO9 + IO10 + one of (IO0 – IO7)	Up to 256 bits of bidirectional serial data
Serial Input	IO8	8-bit characters at 600, 1200, 2400, 4800 or (9600*) bps
Serial Output	IO10	8-bit characters at 600, 1200, 2400, 4800 or (9600*) bps
Touch I/O	IO0 – IO7	Up to 2048 bits of input or output bits
Wiegand Input	Any adjacent pair in IO0 – IO7	Encoded data stream from Wiegand card reader

*: These values apply only to products for which a 20-MHz input clock is available (operating at 20 MHz)

Table 1.4 Summary of Timer/Counter Input Objects

I/O Object	Applicable I/O Pins	Input Signal
Dualslope Input	IO0, IO1 + one of (IO4 – IO7)	Comparator output of the dual slope converter logic
Edgelog Input	IO4	A stream of input transitions
Infrared Input	IO4 – IO7	Encoded data stream from an infrared demodulator
Ontime Input	IO4 – IO7	Pulse width of 0.2 μ s to 1.678 s
Period Input	IO4 – IO7	Signal period of 0.2 μ s to 1.678 s
Pulsecount Input	IO4 – IO7	0 to 65,535 input edges during 0.839 s
Quadrature Input	IO4 + IO5, IO6 + IO7	\pm 16,383 binary Gray code transitions
Totalcount Input	IO4 – IO7	0 to 65,535 input edges

Table 1.5 Summary of Timer/Counter Output Objects

I/O Object	Applicable I/O Pins	Input Signal
Edgedivide Output	IO0, IO1 + one of (IO4 – IO7)	Output frequency = the input frequency divided by a use-specified number
Frequency Output	IO0, IO1	Square wave of 0.3 Hz to 2.5 MHz
One-Shot Output	IO0, IO1	Pulse of duration 0.2 μ s to 1.678 s
Pulsecount Output	IO0, IO1	0 to 65,535 pulses
Pulsewidth Output	IO0, IO1	0 to 100% duty cycle pulse train
Triac Output	IO0, IO1 + one of (IO4 – IO7)	Delay of output pulse with respect to input edge
Triggeredcount Output	IO0, IO1 + one of (IO4 – IO7)	Output pulse controlled by counting input edges

Table 1.6 Pin Configuration for I/O Objects

		0	1	2	3	4	5	6	7	8	9	10
DIRECT I/O MODES	Bit Input, Bit Output											
	Byte Input, Byte Output	All Pins 0 to 7										
	Leveldetect Input											
	Nibble Input, Nibble Output	Any Four Adjacent Pins										
	Muxbus I/O	Data Pins 0 to 7								ALS	WS	RS
PARALLEL I/O MODES	Parallel I/O { Master/Slave A Slave B	Data Pins 0 to 7								CS	R/W	HS
		Data Pins 0 to 7								CS	R/W	A ₀
		Data Pins 0 to 7								CS	R/W	A ₀
	Magcard Input	Optional Timeout								C	D	
	Magtrack1 Input	Optional Timeout								C	D	
SERIAL I/O MODES	Bitshift Input, Bitshift Output	CD	CD	CD	CD	CD	CD	CD	CD		CD	CD
	Neurowire I/O { Master Slave	Optional Chip Select								C	D	D
		Optional Timeout								C	D	D
		Optional Timeout								C	D	D
	I ² C I/O									C	D	
	Serial Input											
	Serial Output											
	Touch I/O											
	Wiegand Input	01	01	01	01	01	01	01	01			
	Dualslope Input	control										
TIMER/COUNTER INPUT MODES	Edgelog Input											
	Infrared Input											
	Overtime Input											
	Period Input											
	Pulsecount Input											
	Quadrature Input					4 + 5	6 + 7					
	Totalcount Input											
	Edgedivide Output											
	Frequency Output											
TIMER/COUNTER OUTPUT MODES	One-Shot Output											
	Pulsecount Output											
	Pulsewidth Output											
	Triac Output	control										
	Triggeredcount Output	Sync Input										
		Sync Input										
		0	1	2	3	4	5	6	7	8	9	10
		High Sink			Pull-Ups				Standard			

Bitshift, I²C, Magcard, Magtrack, Neurowire: C = clock d = Data

Timer/Counter 1 Devices

```

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

```

Timer/Counter 2 Devices

```

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

```

1. I/O Timing Issues

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

- 1) The scheduler;
- 2) The I/O function block's firmware;
- 3) The Neuron Chip hardware.

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

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

1.1 Scheduler-Related I/O Timing Information

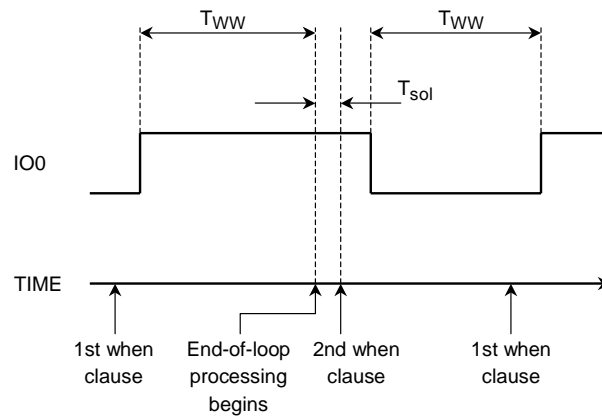
As part of the Neuron Chip firmware, the scheduler provides an orderly and predictable means to facilitate the evaluation of user defined events. The *when* clause, provided by the Neuron C language, is used to specify such events. For more information on the operation of the scheduler, refer to the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

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

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

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

Processing of *when* clauses is done in round-robin fashion; therefore, the Neuron C code above performs alternating activation of the IO0 pin in order to isolate and extract the timing parameters associated with the scheduler. The waveform seen on pin IO0 of the Neuron Chip as a result of the above code is shown in Figure 1.2.



Parameter	Description	Min	Typ.	Max
T_{ww}	When clause to when clause latency	—	940 μ s	—
T_{sol}	Scheduler overhead latency (see text)	—	54 μ s	—

Figure 1.2 *When Clause to When Clause Latency, T_{ww} , and Scheduler Overhead Latency, T_{sol} (not to scale).*

(times shown are for a 10 MHz neuron chip input clock.)

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

The above example not only measures the best-case minimum latency between consecutive *when* clauses (whose events evaluate to **TRUE**), T_{ww} , but also reveals the scheduler's end-of-the-loop overhead latency, T_{sol} . As shown in Figure 8.2, T_{ww} is the off-time period of the output waveform; and T_{sol} is the on-time period of the output waveform, minus T_{ww} . This shows that the scheduler overhead latency, or the scheduler end-of-the-loop latency, occurs just before the execution of the last *when* clause in the program.

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

1.2 Synchronization

To prevent metastability and to provide consistent behavior in response to external events, all eleven I/O pins of the Neuron Chip, when configured as inputs, are passed through a synchronization block controlled by the system clock. Therefore, to guarantee that the Neuron Chip captures a value on an I/O pin, the value must remain stable for at least one system clock cycle (200 ns for a Neuron Chip running at 10 MHz). The only exception to this rule is the Chip Select (\sim CS) input used in the Slave B mode of the parallel I/O object; this input will recognize rising edges asynchronously (see section 2.22).

Figure 1.3 shows the relationship between any external signal transition on an input pin, and the internal system clock of the Neuron Chip.

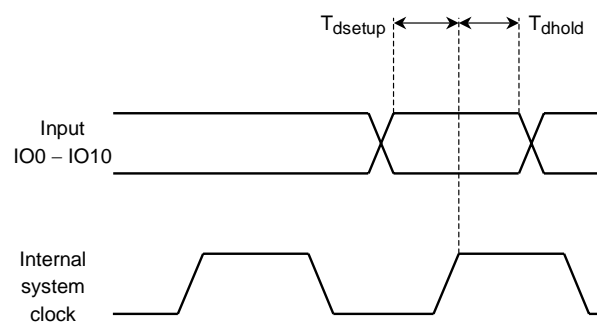


Figure 1.3 Case of All External Signals Being Synchronized by the Neuron Chip Before Being Used

T_{dsetup}	Input data setup	20 ns
T_{dhold}	Input data hold	0 ns

In the case of an I/O pin defined as output, Figure 1.4 shows the timing relationship between output transitions and the internal system clock.

The timing variable, T_{doe} , is the delay from the rising edge of the internal clock, to the transition of an output pin on the Neuron Chip.

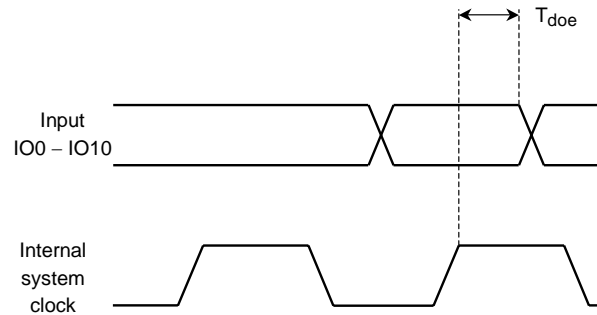


Figure 1.4 Case of All Output Signals Being Synchronized by the Neuron Chip

T_{doe}	Clock to data out	100 ns
-----------	-------------------	--------

1.3 Firmware and Hardware Related I/O Timing Information

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

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

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

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

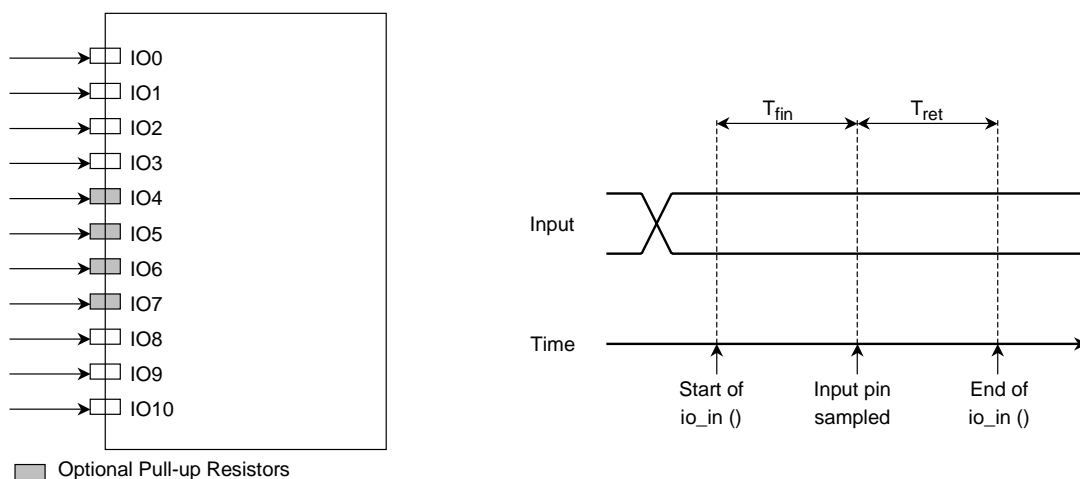
Note: Some input/output functions suspend application processing until the task is complete. This is because they are firmware driven. These are BIT SHIFT, NEUROWIRE, PARALLEL, and SERIAL I/O FUNCTIONS. However, they do not suspend network communication as this is handled by the NETWORK processor and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

2. I/O Objects

The following is a description of each of the I/O objects, listed in alphabetical order. The timing data shown in this section are valid for either an explicit I/O call, or an implicit I/O call through a *when* clause, and are assumed to be for a Neuron Chip running at 10 MHz.

2.1 Bit Input

Pins IO0 through IO10 may be individually configured as single-bit input ports. Inputs may be used to sense TTL-level compatible logic signals from external logic, or from contact closures and the like. The direction of bit ports may be dynamically changed between input and output under application control.

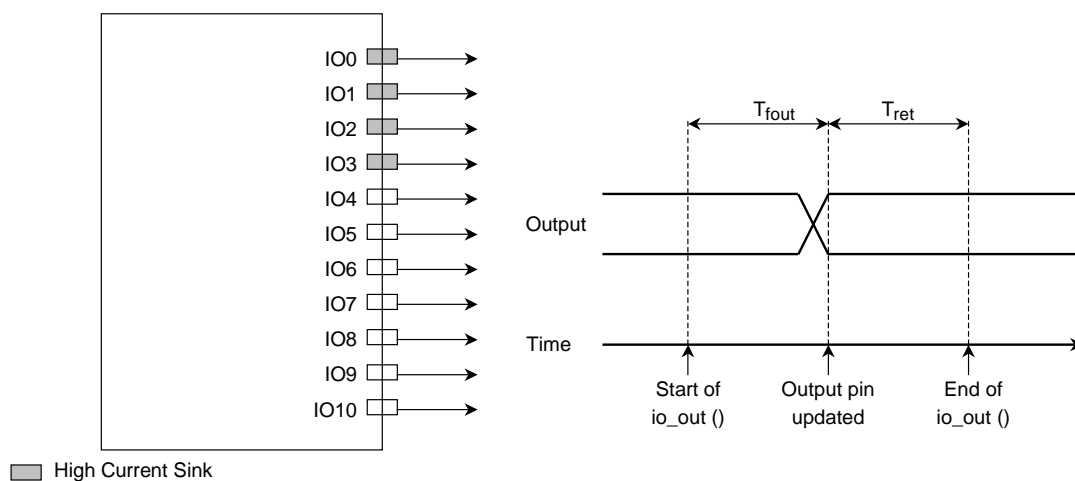


Parameter	Description	Min	Typ.	Max
T_{fin}	Function call to sample IO0 to IO10	—	41 μ s	—
T_{ret}	Return from function	—		—
	IO0		19 μ s	
	IO1		23.43 μ s	
	IO2		27.86 μ s	
	IO3		32.29 μ s	
	IO4		36.72 μ s	
	IO5		41.15 μ s	
	IO6		45.58 μ s	
	IO7		50 μ s	
	IO8		19 μ s	
	IO9		23.43 μ s	
	IO10		27.86 μ s	

Figure 2.1 Bit Input Object

2.2 Bit Output

Pins IO0 through IO10 may be individually configured as single-bit input or output ports. Outputs may be used to drive external CMOS-level compatible logic, and switch transistors and relays may be used to actuate higher-current external objects such as stepper motors and lights. The high current sink capability of pins IO0 through IO3 allows these pins to drive many I/O devices directly. The direction of bit ports may be dynamically changed between input and output under application control.

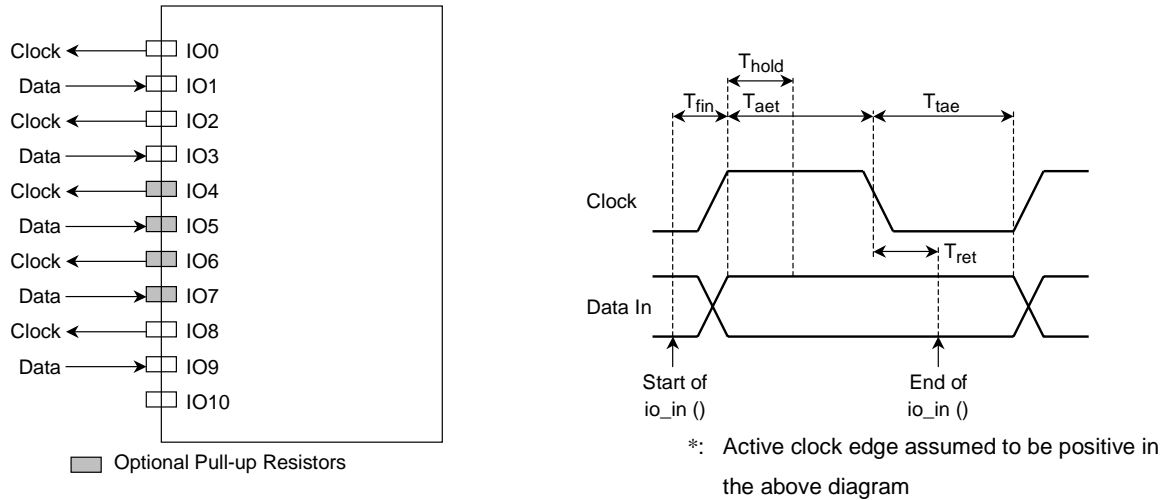


Parameter	Description	Min	Typ.	Max
T_{fout}	Function call to update IO3 to IO5	—	69 μ s	—
	All others	—	60 μ s	—
T_{ret}	Return from function IO0 to IO10	—	5 μ s	—

Figure 2.2 Bit Output Object

2.3 Bitshift Input

Pairs of adjacent pins may be configured as serial input lines, with the lower-numbered pin used as the clock (driven by the Neuron Chip) and the higher-numbered pin used for up to 16 bits of serial data. *The data transmission rate may be configured as 1, 10, or 15 kbps at maximum input clock rate (10 MHz). The data transmission 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 from external logic employing shift registers. This object suspends application processing until the operation is complete.



Parameter	Description	Min	Typ.	Max
T_{fin}	Function call to first clock edge	—	156.6 μ s	—
T_{ret}	Return from function	—	5.4 μ s	—
T_{hold}	Active clock edge to sampling of input data 15 kbps data transmission rate 10 kbps data transmission rate 1 kbps data transmission rate	—	9 μ s 40.8 μ s 938.2 μ s	—
T_{aet}	Active clock edge to next clock transition 15 kbps data transmission rate 10 kbps data transmission rate 1 kbps data transmission rate	—	31.8 μ s 63.6 μ s 961 μ s	—
T_{tae}	Clock transition to next active clock edge 15 kbps data transmission rate 10 kbps data transmission rate 1 kbps data transmission rate	—	14.4 μ s 14.4 μ s 14.4 μ s	—
F	Clock frequency = $1/(T_{aet} + T_{tae})$ 15 kbps data transmission rate 10 kbps data transmission rate 1 kbps data transmission rate	—	21.6 kHz 12.8 kHz 1.03 kHz	—

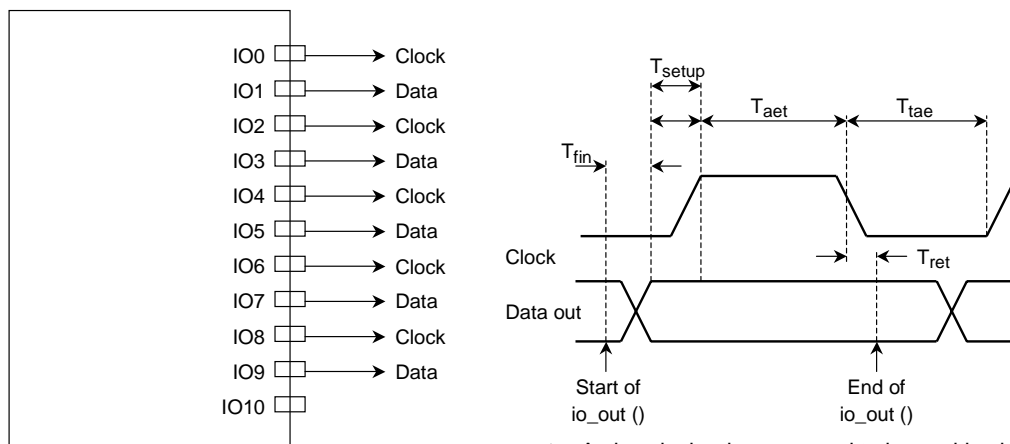
Figure 2.3 Bitshift Input Object

Data and clock are controlled directly by the processor rather than by a hardware-implemented shift register.

*: When a product supporting an input clock of 20 MHz operates at a 20-MHz clock, selection can be made among 2 Kbps, 20 Kbps, and 30 Kbps.

2.4 Bitshift Output

Pairs of adjacent pins may be configured as serial output lines, with the lower-numbered pin used as the clock (driven by the Neuron Chip) and the higher-numbered pin used for up to 16 bits of serial data. *The data transmission rate may be configured as 1, 10, or 15 kbps at maximum input clock rate (10 MHz). The data transmission 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 object suspends application processing until the operation is complete.



*: Active clock edge assumed to be positive in the above diagram

Parameter	Description	Min	Typ.	Max
T_{fin}	Function call to first data out stable 16 bit shift count 1 bit shift count	—	185.3 μ s 337.6 μ s	—
T_{ret}	Return from function	—	10.8 μ s	—
T_{setup}	Data output to active clock edge 15 kbps data transmission rate 10 kbps data transmission rate 1 kbps data transmission rate	—	10.8 μ s 10.8 μ s 10.8 μ s	—
T_{aet}	Active clock edge to next clock transition 15 kbps data transmission rate 10 kbps data transmission rate 1 kbps data transmission rate	—	10.2 μ s 42 μ s 939.5 μ s	—
T_{tae}	Clock transition to next active clock edge 15 kbps data transmission rate 10 kbps data transmission rate 1 kbps data transmission rate	—	34.8 μ s 34.8 μ s 34.8 μ s	—
F	Clock frequency = $1/(T_{aet} + T_{tae})$ 15 kbps data transmission rate 10 kbps data transmission rate 1 kbps data transmission rate	—	22 kHz 13 kHz 1.02 kHz	—

Figure 2.4 Bitshift Output Object

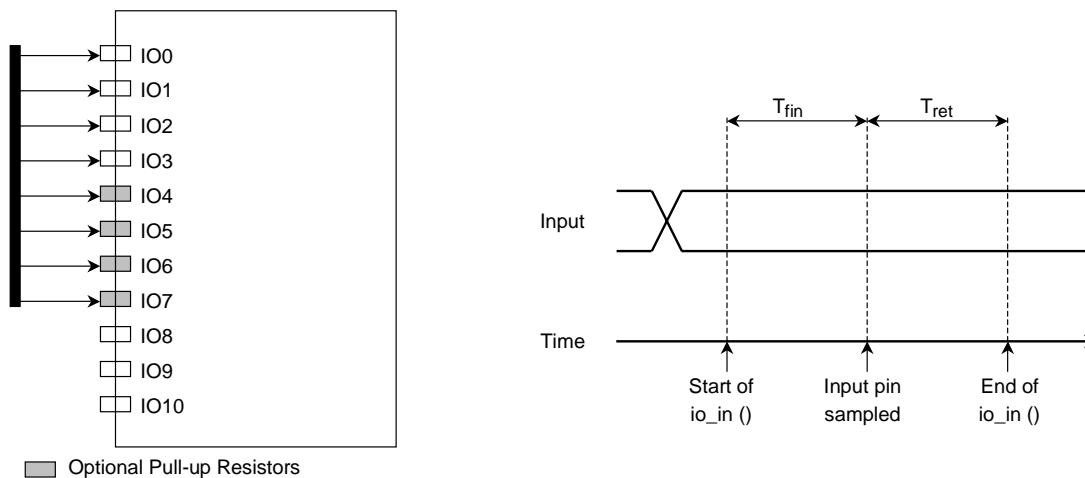
*: When a product supporting an input clock of 20 MHz operates at a 20-MHz clock, selection can be made among 2 Kbps, 20 Kbps, and 30 Kbps.

Data and clock are controlled directly by the processor rather than by a hardware-implemented shift register.

Note that the data-out pin on this function does not necessarily remain at the same level as the last bit shifted out. If this is a requirement, a bit-output overlay must be used on the data-out pin, to force the output to a user-determined state after the bitshift output operation.

2.5 Byte Input

Pins IO0 through IO7 may be configured as a byte-wide input port, which may be read using integers in the range from 0 to 255. This is useful for interfacing to devices that output ASCII data, or other data eight bits at a time. The direction of the byte port may be dynamically changed between input and output under application control. IO0 represents the LSB of the input data.

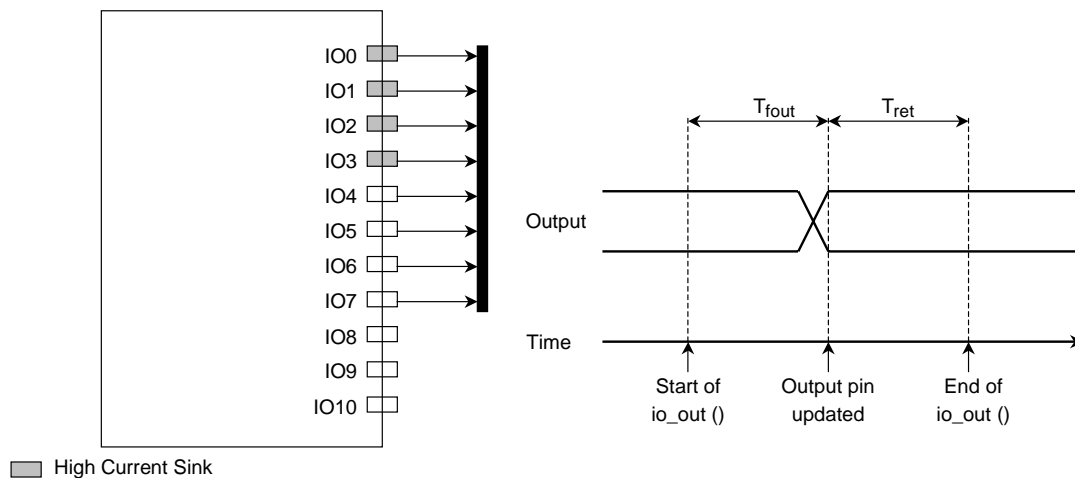


Parameter	Description	Min	Typ.	Max
T_{fin}	Function call to sample	—	24 μ s	—
T_{ret}	Return from function	—	4 μ s	—

Figure 2.5 Byte Input Object

2.6 Byte Output

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



Parameter	Description	Min	Typ.	Max
T_{fout}	Function call to update	—	57 μ s	—
T_{ret}	Return from function	—	5 μ s	—

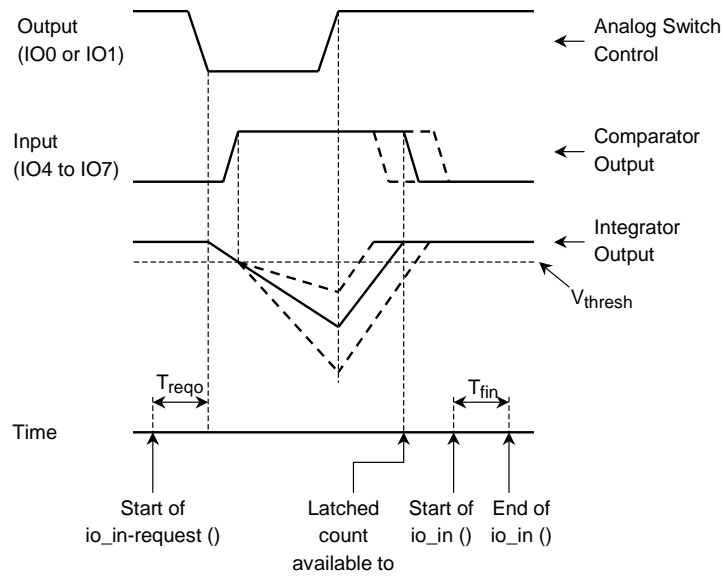
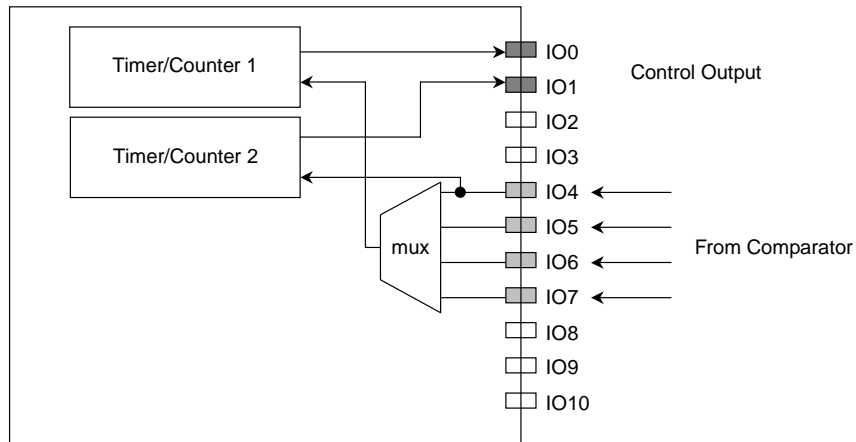
Figure 2.6 Byte Output Object

2.7 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. 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 3.1 (see section 3).

For additional information regarding the dualslope A/D conversion and the Neuron Chip, see the *Analog to Digital Conversion with the Neuron Chip* engineering bulletin (part no. 005-0019-02).



Parameter	Description	Min	Typ.	Max
T_{rego}	<code>io_in-request()</code> to output toggle	—	75.6 μ s	—
T_{fin}	Input function call and return	—	82.8 μ s	—

Figure 2.7 Dualslope Input Object

2.8 Edgedivide Output

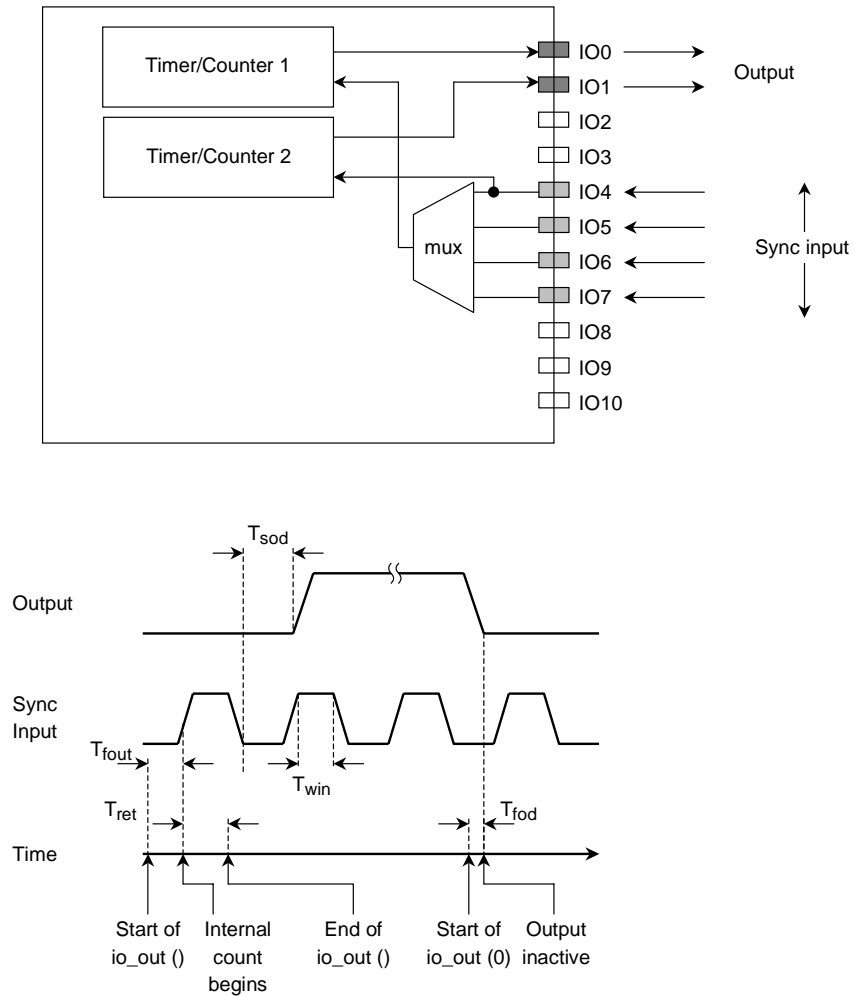
This output object acts as a frequency divider by providing an output frequency on either pin IO0 or IO1, which is a divided-down version of the input frequency applied on pins IO4 to IO7. The object is useful for any divide-by- n operation, where n is passed to the timer/counter object through the application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

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

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

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

Figure 2.8 shows the pinout and timing information for this output object.



Parameter	Description	Min	Typ.	Max
T_{fout}	Function call to start of timer	—	96 μ s	—
T_{fod}	Function to output disable	—	82.2 μ s	—
T_{sod}	Active sync edge to output toggle	550 ns	—	750 ns
T_{win}	Sync input pulse width (10 MHz)	200 ns	—	—
T_{ret}	Return from function	—	13 μ s	—

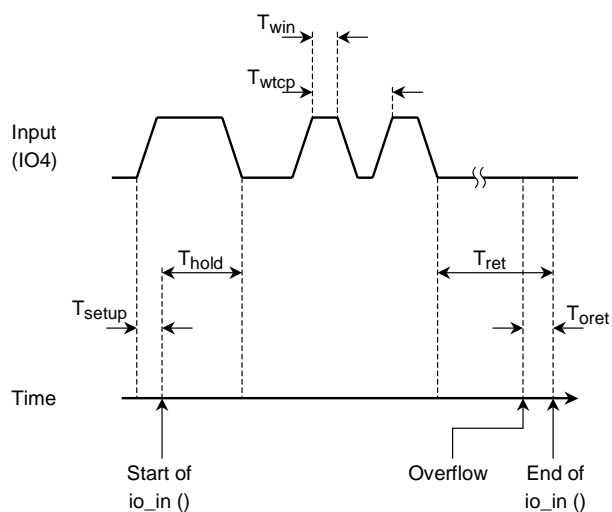
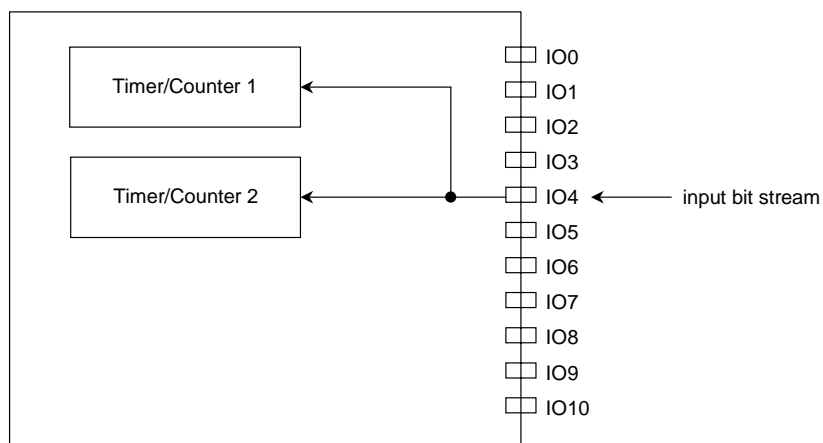
Figure 2.8 Edgedivide Output Object

2.9 Edgelog Input

The edgelog input object can record a stream of input pulses, measuring the consecutive low and high periods of the input and storing them in user-defined storage. The values stored represent the units of clock time between input signal edges, both rising and falling. Both timer/counters of the Neuron Chip are used for this object.

Each 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 3.1 (see section 3). 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.



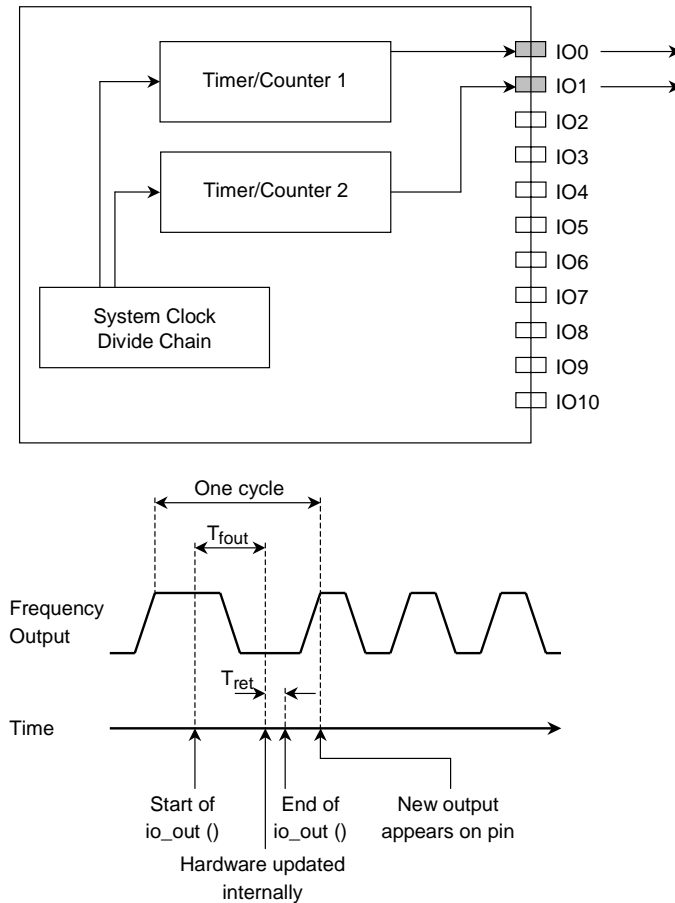
Parameter	Description	Min	Typ.	Max
T_{setup}	Input data setup	0	—	—
T_{win}	Input pulse width	1 T/C clk	—	65534 T/C clks
T_{hold}	<code>io_in()</code> call to data input edge for inclusion of that pulse	26.4 μs	—	—
T_{wtcp}	Two consecutive pulse widths	104 μs	—	—
T_{oret}	Return on overflow	—	42.6 μs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

Note: T/C clk represents the clock time expired during the declaration of the I/O object.

Figure 2.9 Edgelog Input Object

2.10 Frequency Output

A timer/counter may be configured to generate a continuous square wave with 50% duty cycle. The resolution and maximum value of the half-period of the signal is given by Table 3.1 (see section 3). 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.



Parameter	Description	Min	Typ.	Max
T_{fout}	Function call to output update	—	96 μ s	—
T_{ret}	Return from function	—	13 μ s	—

Figure 2.10 Frequency Output Object

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.

*: If selected "clock 0" for the clock and "1", "2" or "3" for the output-value, Neuron chip doesn't output correctly.

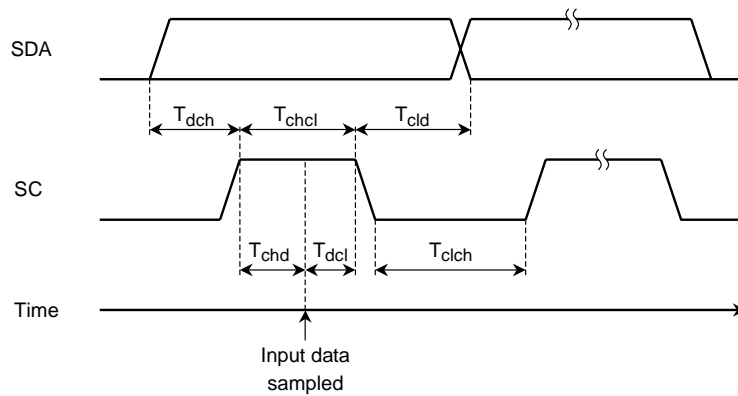
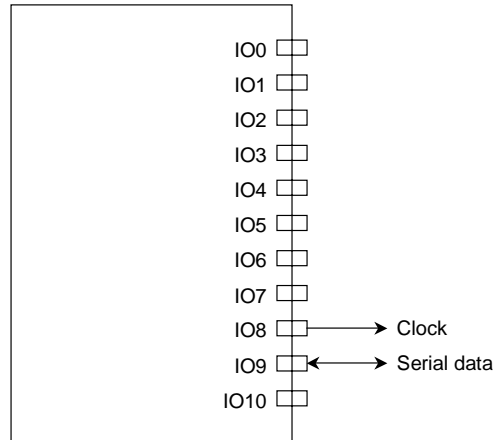
2.11 I²C I/O

This I/O object is used to interface the Neuron Chip to any device which adheres to Philips Semiconductor's Inter-Integrated Circuit (I²C) bus protocol.

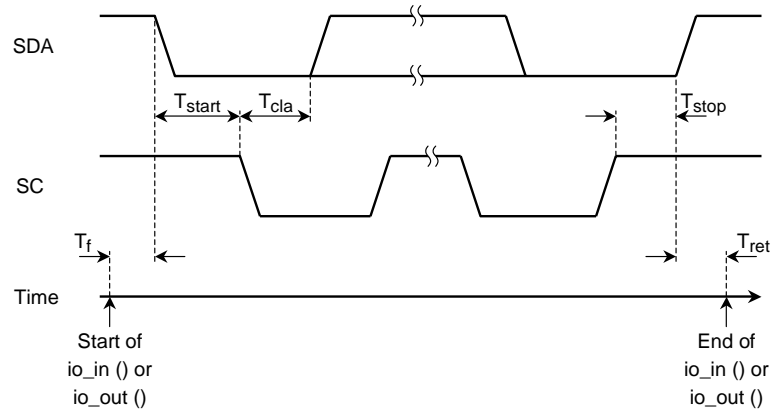
The Neuron Chip is always the master, with IO8 being the clock line (SCL) and IO9 being the serial data line (SDA). These I/O lines are operated in the open-drain mode in order to accommodate the special requirements of the I²C protocol. With the exception of two pull-up resistors, no additional external components are necessary for interfacing the Neuron Chip to an I² device.

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

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



Bit Transfer Timing



Start and Stop Timing

Parameter	Description	Min	Typ.	Max
T_f	I/O call to start condition io_in () io_out ()	—	54.6 μ s 43.4 μ s	—
T_{start}	End of start condition io_in () io_out ()	5.4 μ s 5.4 μ s	—	—
T_{cla}	End of start to start of address io_in () io_out ()	24.0 μ s 24.0 μ s	—	—
T_{cld}	SCL low to data for io_out ()	24.6 μ s	—	—
T_{dch}	Data to SCL high for io_out ()	7.2 μ s	—	—
T_{chcl}	Clock high to clock low for io_out ()	12.6 μ s	—	—
T_{chd}	SCD low to data sampling for io_in ()	13.2 μ s	—	—
T_{chl}	Data sample to SCL low for io_in ()	7.2 μ s	—	—
T_{clch}	Clock low to clock high for io_in ()	24.0 μ s	—	—
T_{stop}	Clock high to data io_in () io_out ()	12.6 μ s 12.6 μ s	—	—
T_{ret}	SDA high to return from function io_in () io_out ()	—	—	4.2 μ s 4.2 μ s

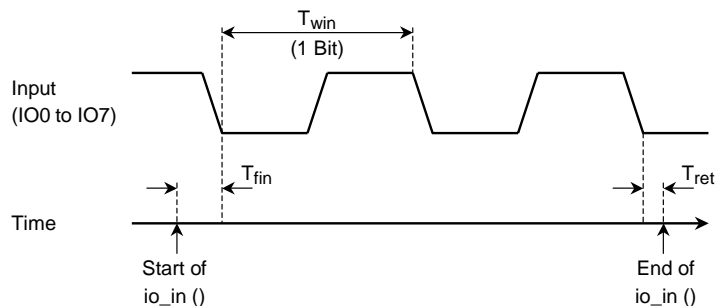
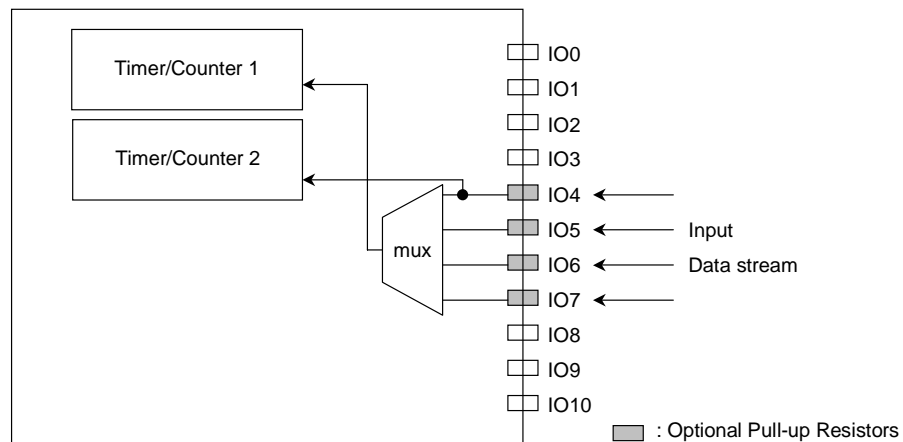
Figure 2.11 I²C I/O Object

2.12 Infrared Input

The infrared input object is used to capture a stream of data generated by a class of infrared remote control devices. 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, which can be used as 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 3.1 (see section 3).

This function can be used with an off-the-shelf IR demodulator (e.g. the Toshiba TPS832) to quickly develop an infrared interface to the Neuron Chip.



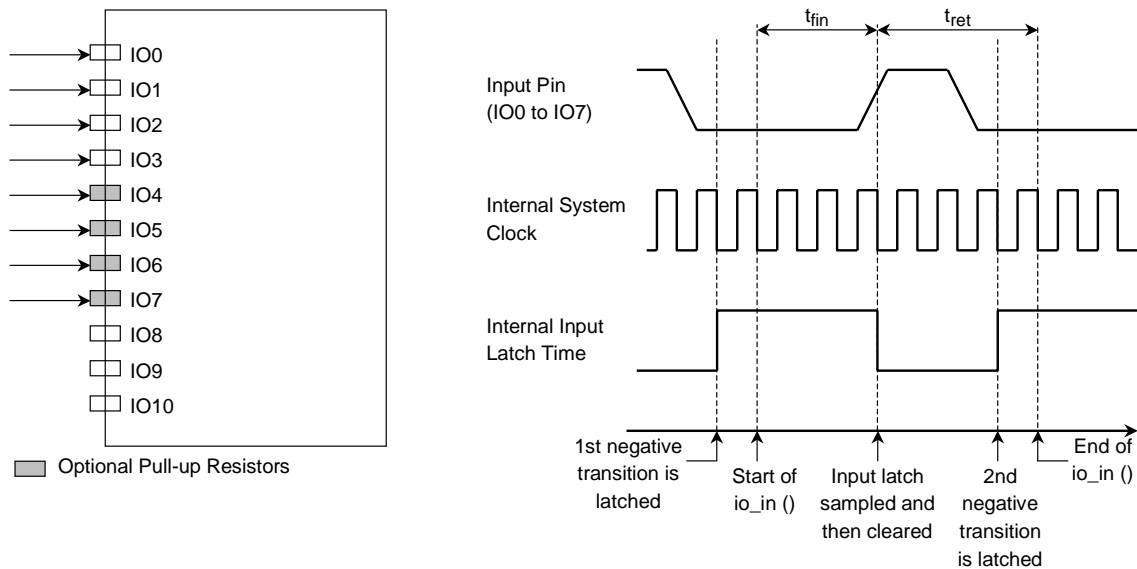
Parameter	Description	Min	Typ.	Max
T_{fin}	Function call to start of input sampling	—	82.2 μ s	—
T_{win}	Minimum input period width	—	93 μ s	—
T_{ret}	End of last valid bit to function return	max-period	max-period	—

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

Figure 2.12 Infrared Input Object

2.13 Leveldetect Input

Pins IO0 through IO7 may be configured as level-detect input pins, which latch a logic high-to-low-level transition on the input pin. The application can therefore detect short pulses on the input which might be missed by software polling. This is useful for reading devices such as proximity sensors. **Note that this is the only I/O function which is latched before it is sampled.** The latch is cleared during the when statement sampling, and can be set again immediately after, if another transition should occur. See Figure 2.13.



Parameter	Description	Min	Typ.	Max
T_{fin}	Function call to start of input sampling			
	IO0		35 μ s	
	IO1		39.43 μ s	
	IO2		43.86 μ s	
	IO3	—	48.29 μ s	—
	IO4		52.72 μ s	
	IO5		57.15 μ s	
	IO6		61.58 μ s	
	IO7		66 μ s	
T_{ret}	Return from function	—	32 μ s	—

Figure 2.13 Level Detect Input Object

2.14 Magcard Input

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

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

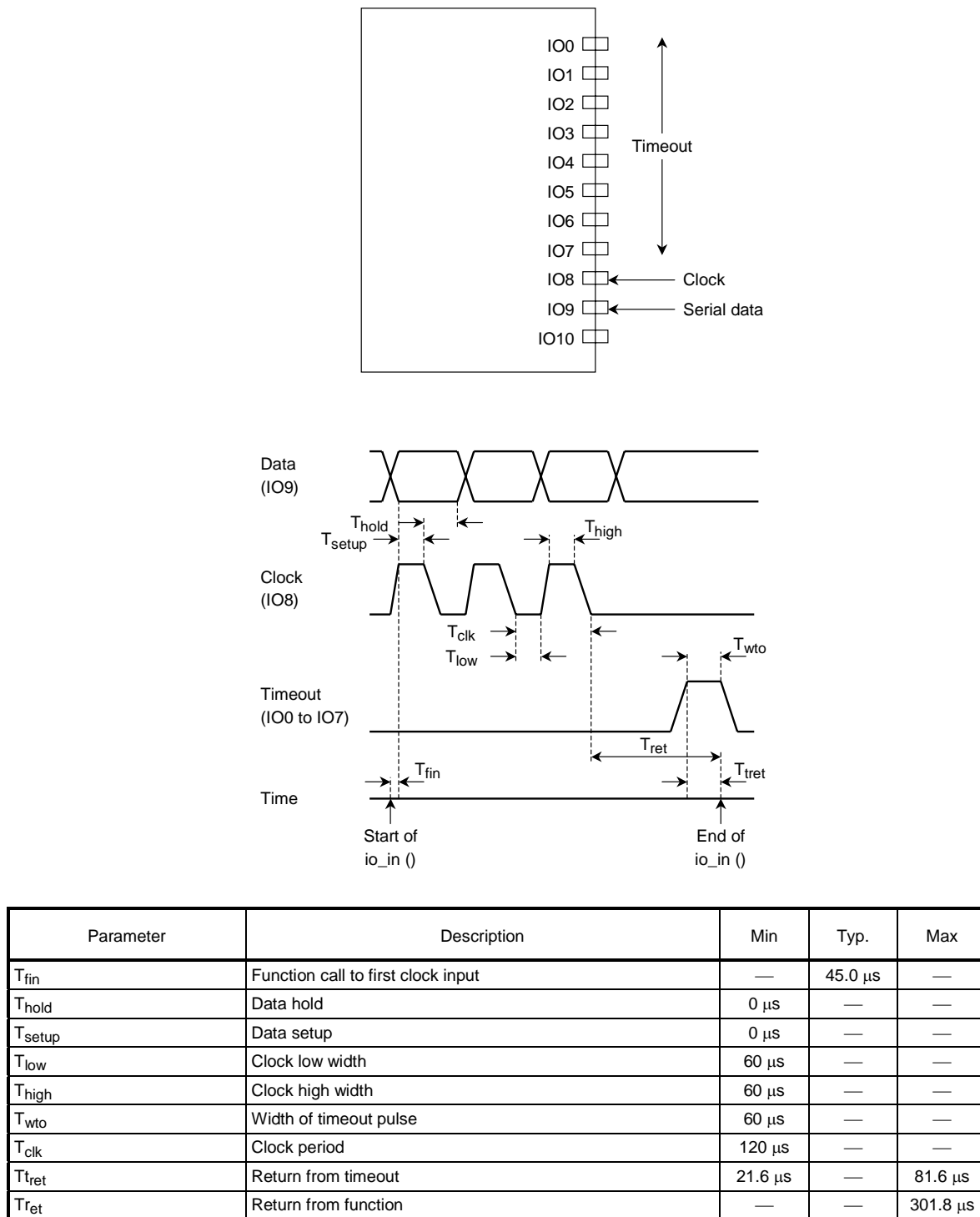


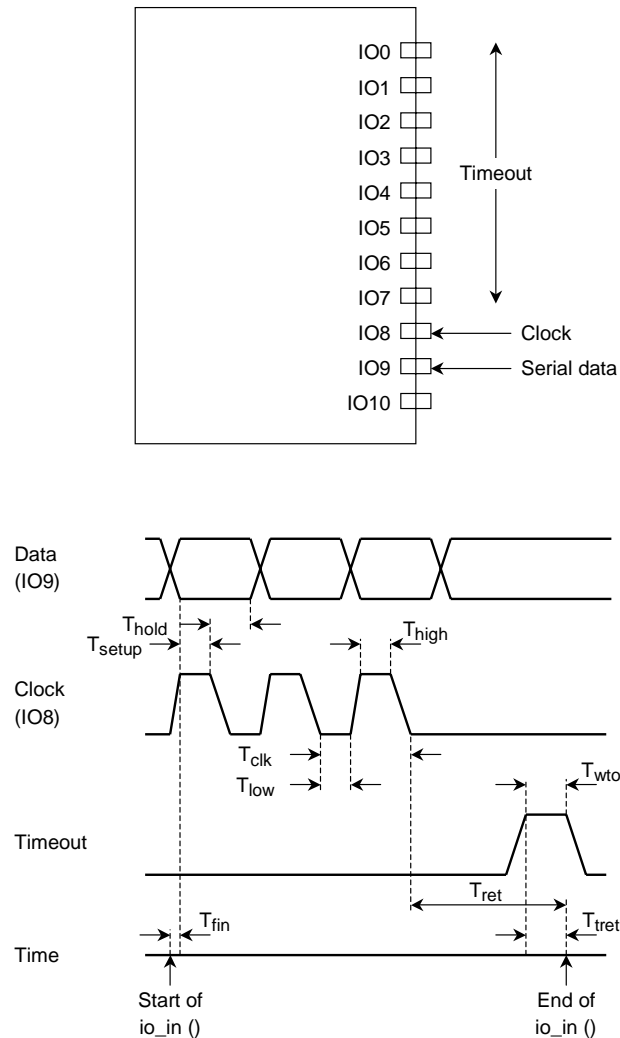
Figure 2.14 Magcard Input Object

A Neuron Chip operating at 10 MHz can perform bit processing at a rate of up to 8334 bps* (at a bit density of 75 bits/inch). This equates to a card velocity of 111 inches/s. 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.

*Up to 16,668 bps at 20 MHz

2.15 Magtrack 1 Input

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



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

Figure 2.15 Magtrack1 Input Object

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

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

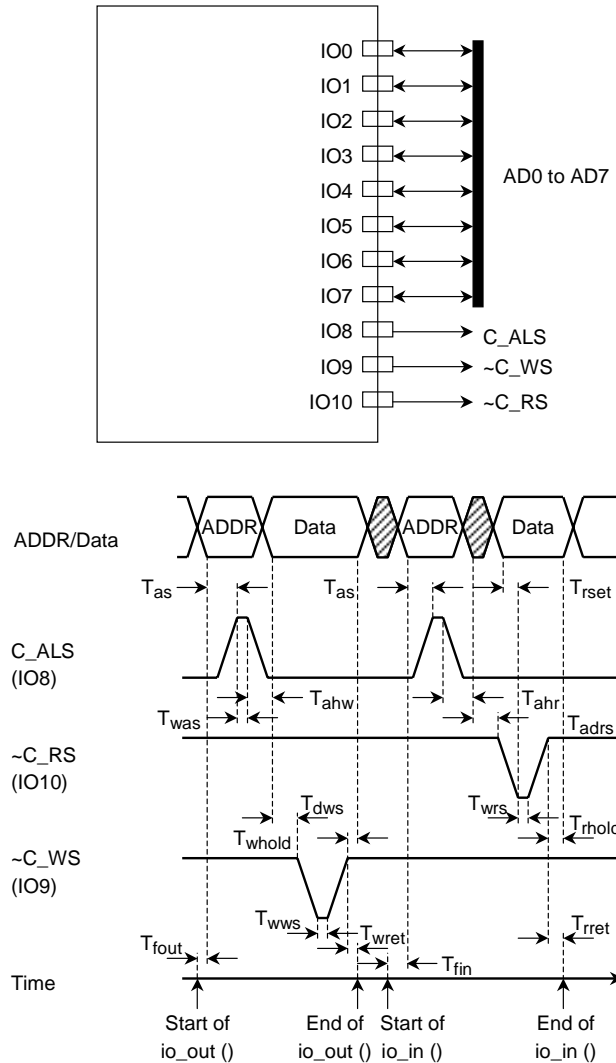
The Magtrack1 input object optionally uses one of the I/O pins IO0 through IO7 as a timeout/abort pin. Use of this feature is suggested since the `io_in()` function will update the watchdog timer during clock wait states, and could result in a lockup if the card were to stop moving in the middle of the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

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

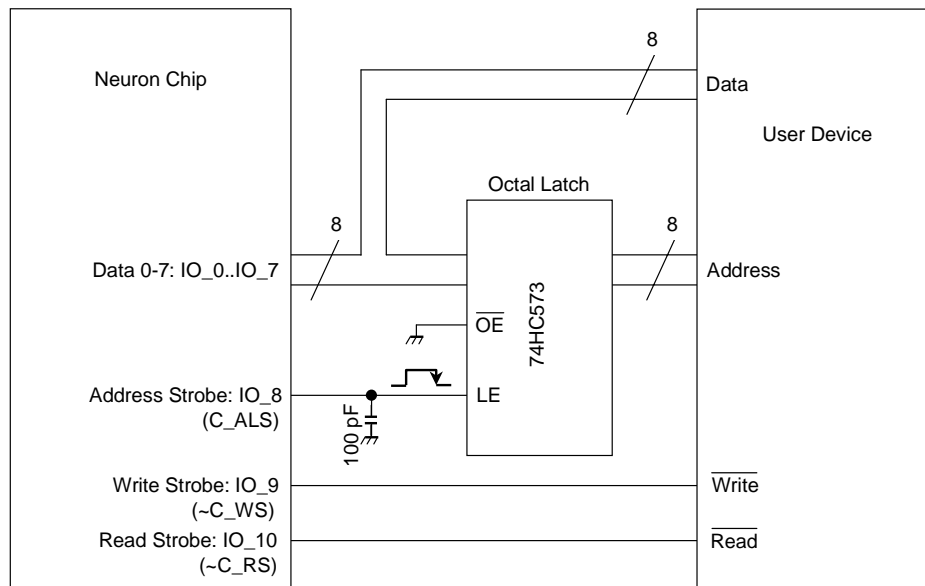
*: When a product supporting an input clock of 20 MHz operates at a 20-MHz clock, a maximum bit rate of 14, 492 bps can be processed.

2.16 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. 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 be essentially in control of all read and write operations at all times. This relieves the attached device from the burden of protocol handling, and results in an easier-to-use interface, although at the expense of some degree of decrease in data throughput capacity.



The address bus can be demultiplexed by installing a 74HC573 octal latch, using the following schematic.



Parameter	Description	Min	Typ.	Max
T_{fout}	io_out() to valid address	—	26.4 μ s	—
T_{as}	Address valid to address strobe	—	10.8 μ s	—
T_{ahw}	Address hold for write	—	4.8 μ s	—
T_{ahr}	Address hold for read	—	6.6 μ s	—
T_{was}	Address strobe width	—	6.6 μ s	—
T_{wrs}	Read strobe width	—	10.8 μ s	—
T_{wws}	Write strobe width	—	6.6 μ s	—
T_{dws}	Data valid to write strobe	—	6.6 μ s	—
T_{rset}	Read data to rising edge of ~C_RS (see (Note))	4.8 μ s	—	—
T_{whold}	Write hold time	4.2 μ s	—	—
T_{rhold}	Read hold time	0 μ s	—	—
T_{adrs}	Address disable to read strobe	—	7.2 μ s	—
T_{fin}	io_in() to valid address	—	26.4 μ s	—
T_{rret}	Function return from read	—	4.2 μ s	—
T_{wret}	Function return from write	—	4.2 μ s	—

Note: Data are actually latched 4.8 μ s after the falling edge of ~C_RS.

Figure 2.16 Muxbus I/O Object

2.17 Neurowire I/O

Pins IO8 through IO10 may be configured as a serial bidirectional port. 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 are clocked out on pin IO9 at the same time as data are clocked in from pin IO10. The default option is to clock data by the rising edge of the clock signal. The invert keyword changes the active edge of the clock to negative. Any one 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 the input clock.

*: When a product supporting an input clock of 20 MHz operates at a 20 MHz clock, selection can be made among 2 Kbps, 20 Kbps, and 40 Kbps.

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 are clocked out on pin IO9 at the same time as data are clocked in from pin IO10. Data are clocked by the rising edge of the clock signal, which may be up to 18 kbps*. The invert keyword changes the active clock edge to positive. One of the pins IO0 through IO7 may be designated as a time-out pin.

*: When a product supporting an input clock of 20 MHz operates at a 20-MHz clock, a maximum bit rate of 36 Kbps can be processed.

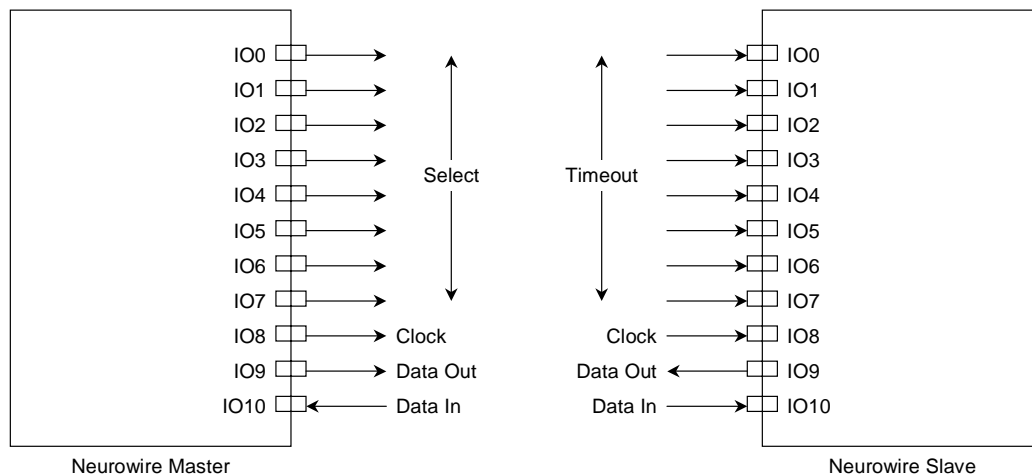
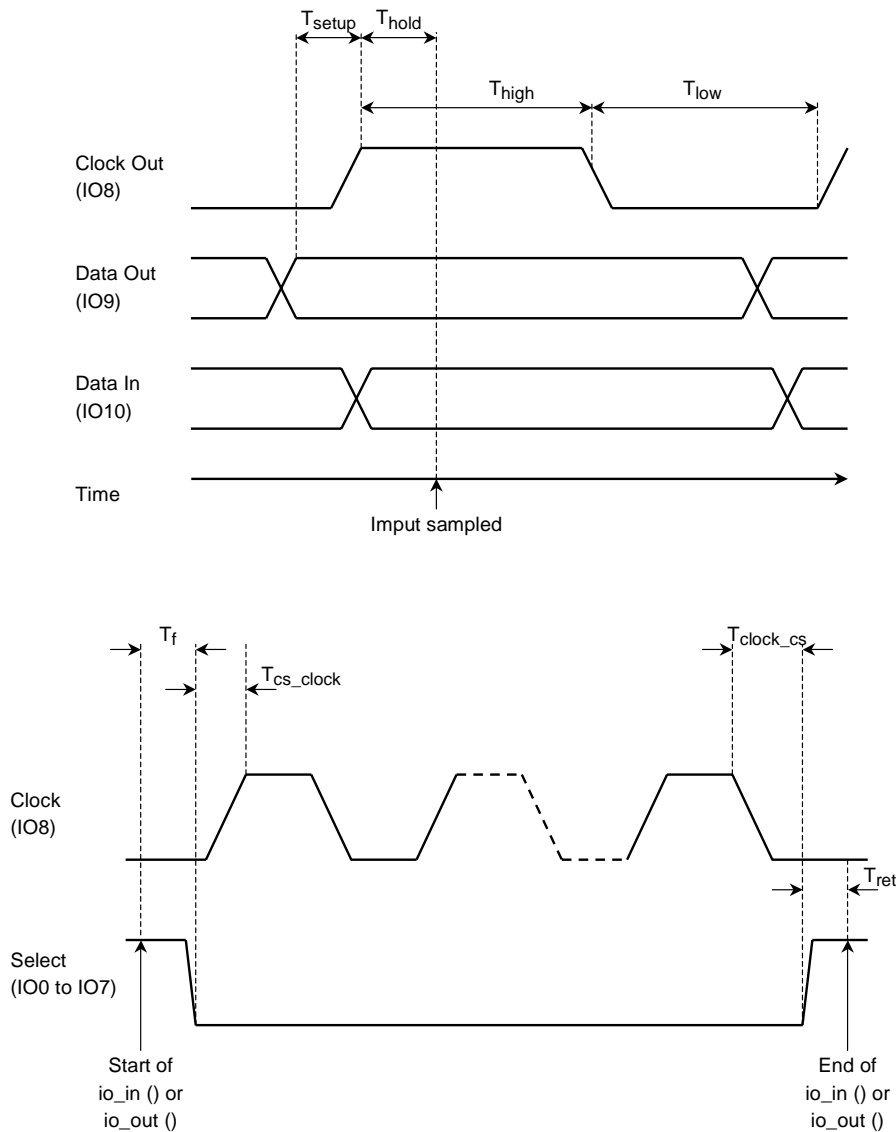


Figure 2.17 Neurowire I/O Object

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.

In both master and slave modes, up to 255 bits of data may be transferred at a time. Neurowire I/O suspends application processing until the operation is complete. The Neurowire object is useful for external devices, such as A/D and D/A converters, and display drivers incorporating serial interfaces that conform with National Semiconductor's MicrowireTM or Motorola's SPI interface.

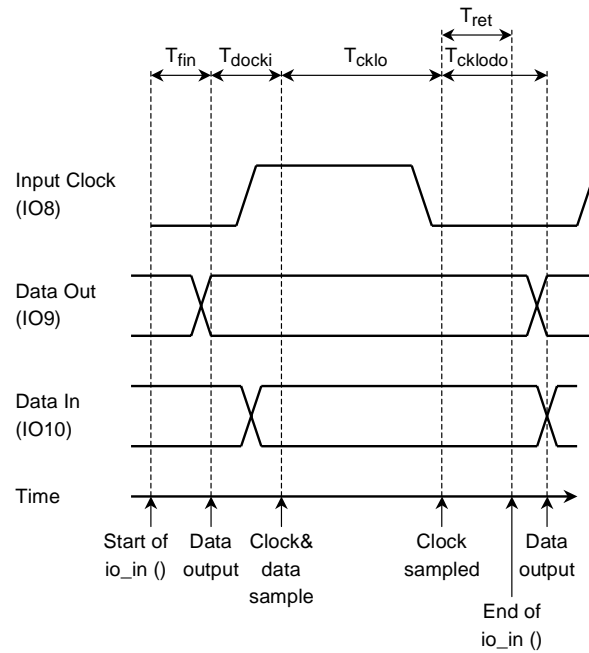
Master Mode



Parameter	Description	Min	Typ.	Max
T_f	Function call to $\sim CS$ active	—	69.9 μs	—
T_{ret}	Return from function	—	7.2 μs	—
T_{hold}	Active clock edge to sampling of input data 20 kbps data transmission rate 10 kbps data transmission rate 1 kbps data transmission rate	—	11.4 μs 53.4 μs 960.6 μs	—
T_{high}	Clock high period (active clock edge = 1) 20 kbps data transmission rate 10 kbps data transmission rate 1 kbps data transmission rate	—	25.8 μs 67.8 μs 975.0 μs	—
T_{low}	Clock low period (active clock edge = 1)	—	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_{clok_cs}	Last clock transition to select inactive	—	81.6 μs	—
F	Clock frequency = $1/(T_{high} + T_{low})$ 20 kbps data transmission rate 10 kbps data transmission rate 1 kbps data transmission rate	—	17.0 kHz 9.92 kHz 992 kHz	—

Figure 2.18 Neurowire Master Mode

Slave Mode



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

Figure 2.19 Neurowire Slave Mode

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

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

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

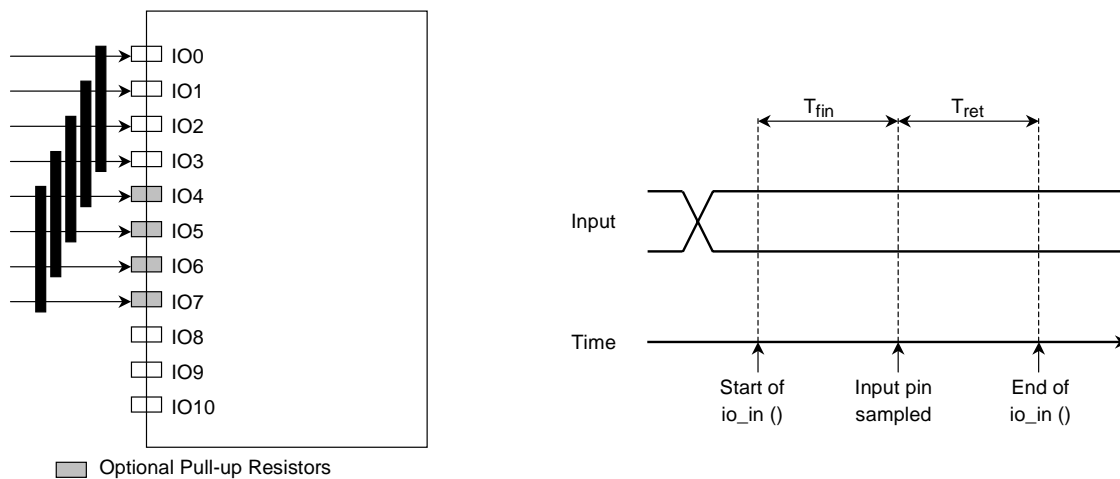
It is assumed that the chip select logic for the Neurowire slave will be handled by the user through a separate bit input object, along with an appropriate handshaking algorithm implemented by the user application program. In order to prevent unnecessary timeouts, it is necessary to provide for the setup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

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

2.18 Nibble Input

Groups of four consecutive pins between IO0 and IO7 may be configured as nibble-wide input ports, which may be read as integers in the range from 0 to 15. This is useful for interfacing to devices that output BCD data, or other data four bits at a time. For example, a 4×4 key switch matrix may be scanned by using one nibble (see nibble output I/O object) to generate a row select output (one of four rows); and using the other nibble to read the input from the columns of the switch matrix.

The direction of nibble ports may be dynamically changed between input and output 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.



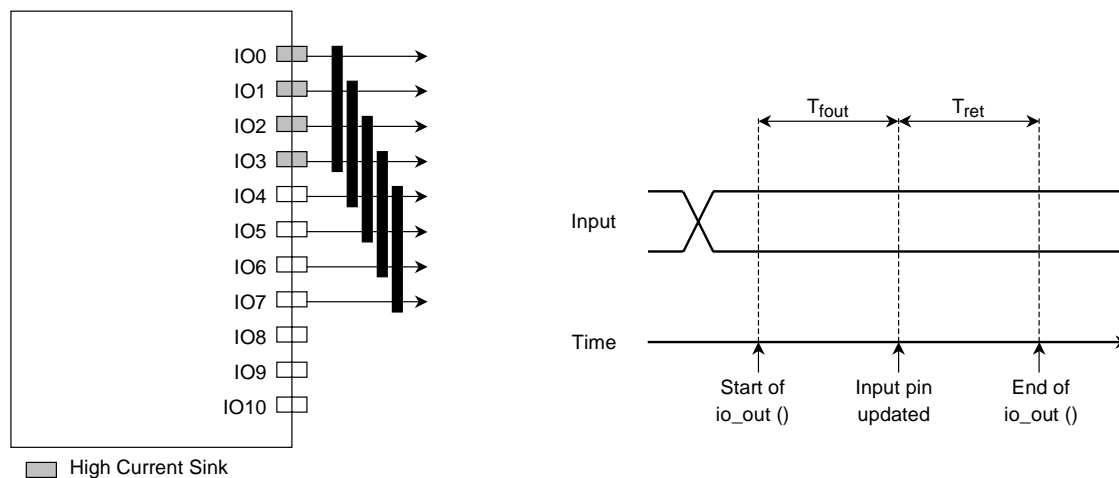
Parameter	Description	Min	Typ.	Max
T_{fin}	Function call to sample IO0 to IO4	—	41 μ s	—
T_{ret}	Return from function IO0 IO1 IO2 IO3 IO4	—	18 μ s 22.75 μ s 27.5 μ s 32.25 μ s 37 μ s	—

Figure 2.20 Nibble Input Object

2.19 Nibble Output

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

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

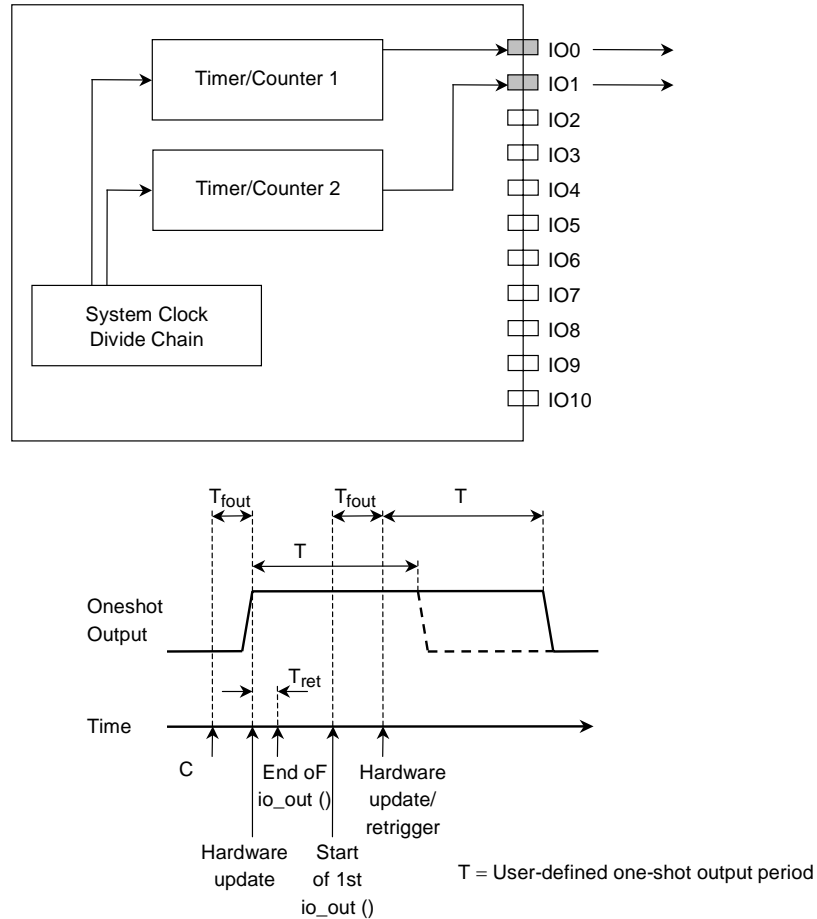


Parameter	Description	Min	Typ.	Max
T_{fout}	Function call to update			
	IO0		78 μ s	
	IO1		89.75 μ s	
	IO2	—	101.5 μ s	—
	IO3		113.25 μ s	
	IO4		125 μ s	
T_{ret}	Return from function			
	IO0 to IO4	—	5 μ s	—

Figure 2.21 Nibble Output Object

2.20 Oneshot 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 one pulse causes it to continue for the duration of the next pulse. Table 3.1 (see section 3) 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 CPU.



Parameter	Description	Min	Typ.	Max
T_{fout}	Function call to output update	—	96 μ s	—
T_{ret}	Return from function	—	13 μ s	—
T_{jit}	Output duration jitter	—	—	1 timer/counter clock period (*)

*: Timer/counter clock period = $2000 \times 2^{(\text{clock selected number})} / \text{input clock (MHz)}$

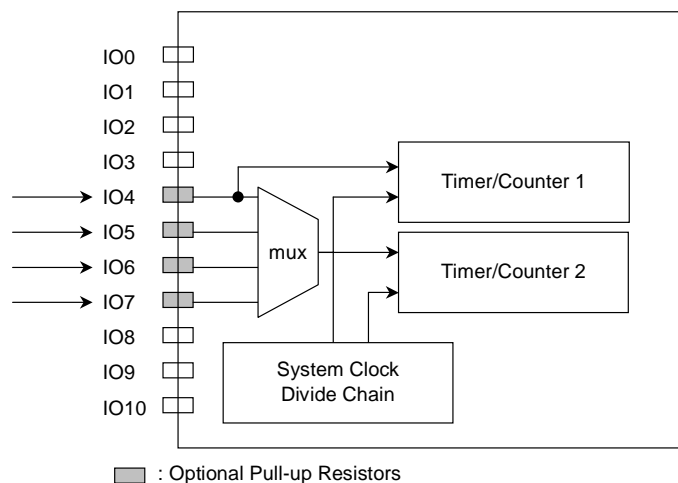
Figure 2.22 One-Shot Output Object

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.

*: If selected “clock 0”, “clock 1” or “clock 2” for the clock and “1”, “2” or “3” for the output-value, Neuron chip doesn’t output correctly.

2.21 Ontime Input

A timer/counter may be configured to measure the time for which its input is asserted. Table 3.1 (see section 3) shows resolution and the maximum times for different I/O clock selections. Assertion may be defined as either logic high or logic low. This object may be used to implement a simple analog-to-digital converter with a voltage-to-pulsewidth circuit, or for measuring velocity by timing motion past a position sensor.



Parameter	Description	Min	Typ.	Max
T_{fin}	Function call to counter output sample	—	86 μ s	—
T_{ret}	Return from function	—	52/22 μ s	—

Figure 2.23 On-Time Input Object

For T_{ret} , the first number represents the return latency in the case where there is a new measurement involved. The second number is for the case where no new on-time value is being returned. In the latter case, the value corresponding to the previous on-time measurement is returned.

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.

Notice: Timer/Counter Input Functions**(on-time, period, pulsecount input, quadrafire, totalcount)**

Input timer/counter functions have the advantage (over non-timer/counter functions) that input events will be captured even if the application processor is occupied doing something else when the event occurs. However, be aware that single events cannot be measured with the timer/counters. 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.* Figure 2.24 shows an example of how the timer/counter functions are processed with a Neuron C *when* statement.

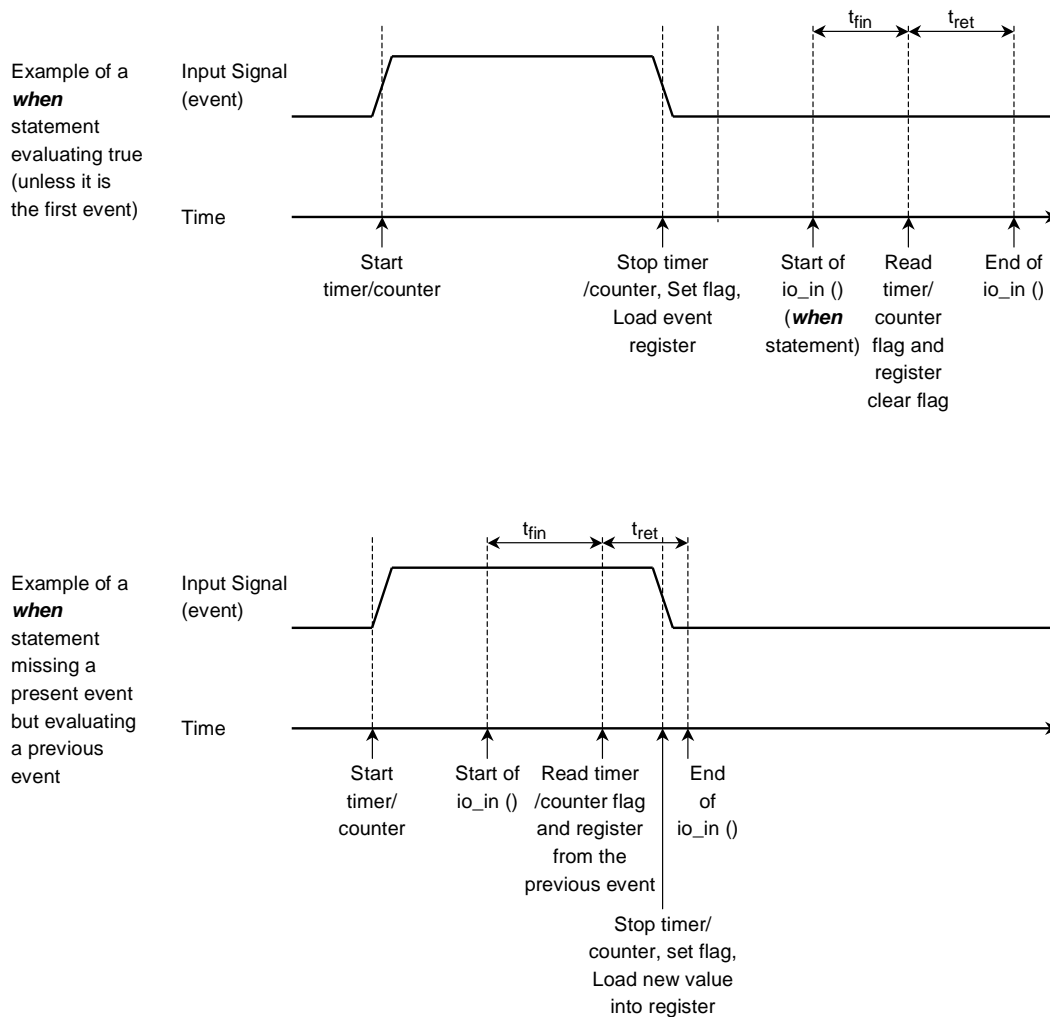


Figure 2.24 When Statement Processing Using The On-Time Input Function

2.22 Parallel I/O

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, micro-controller, or another Neuron Chip for gateway applications.

The parallel interface on the Neuron Chip may be configured in either master, slave A or slave B mode. In all cases pins IO0 through IO7 are used as an eight bit-wide, bidirectional data bus. When configured in master mode, the Neuron Chip drives IO8 low as a chip select, and IO9 to specify whether this is a read (high) or a write (low) cycle. Accepts IO10 as the handshake acknowledgment signal.

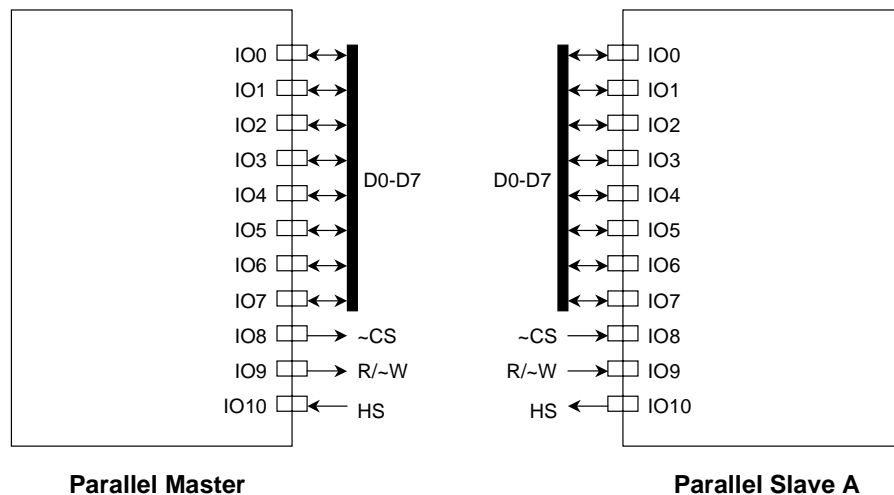


Figure 2.25 Parallel I/O-Master and Slave A

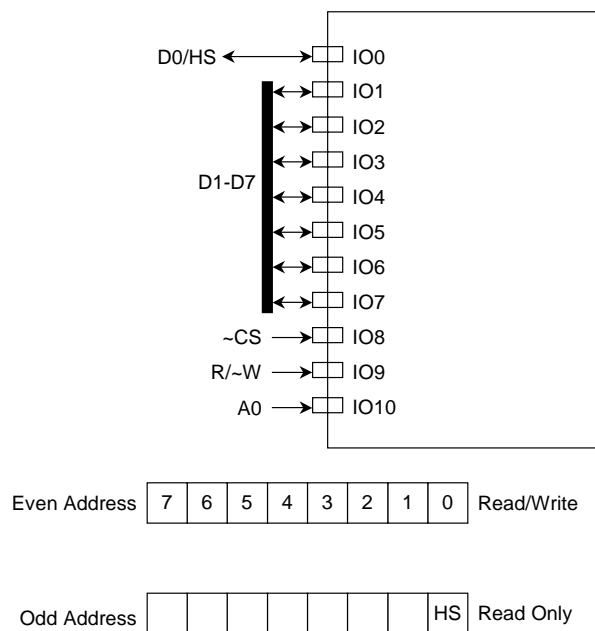


Figure 2.26 Slave B-Neuron Chip as Memory-Mapped I/O Device on Host

When configured in slave A mode, the Neuron Chip accepts IO8 as a chip select, and accepts IO9 to specify whether this is a read or a write cycle (with respect to the master). The Neuron Chip drives IO10 as the handshake acknowledgment signal. This is designed for use with a master processor that has an eight-bit data port plus a three-bit control port. A Neuron Chip in Master mode may be directly connected to a Neuron Chip in Slave A mode to create an application-level gateway.

When configured in Slave B mode, the Neuron Chip accepts IO8 as a chip select, accepts IO9 to specify whether the master will read or write, and accepts IO10 as a selector input. When IO10 is low and \sim CS is asserted, pins IO0 through IO7 form the bidirectional data bus. When IO10 is high, while R/ \sim W is high and \sim CS is asserted, IO0 is driven as the handshake acknowledgment signal to the master. This is designed for use with a master processor that uses memory-mapped I/O. The least significant bit of the master's address bus is connected to the Neuron Chip's IO10 pin. The Neuron Chip appears 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. The least significant bit of the status register is the handshake (HS) bit.

The data transfer operation between the master and the slave is accomplished through the use of a virtual write token-passing protocol. The write token is passed alternatively between the master and the slave on the bus in an infinite ping-pong fashion. The owner of the token has the option of writing a series of data bytes, or alternatively passing the write token without any data. Figure 8.28 illustrates the sequence of operations for this token-passing protocol.

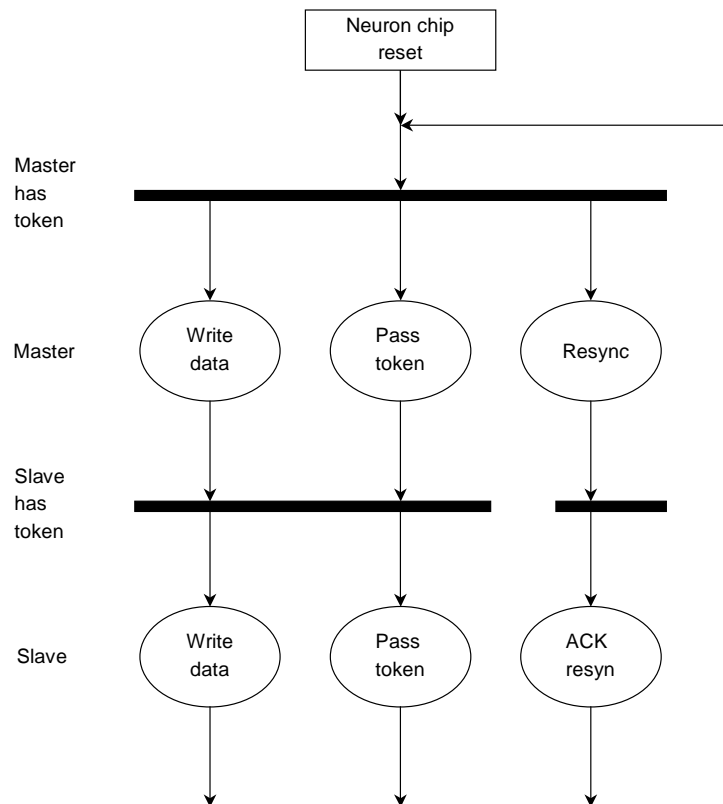


Figure 2.27 Handshake Protocol Sequence between the Master and Slave

The diagram illustrates the timing for Read and Write cycles. The signals shown are:

- CS (OUT) (IO8)**: Chip select output.
- HS (IN) (IO10)**: High-speed input.
- R/~W (OUT) (IO9)**: Read/Write control output.
- Data (OUT) (IO0 to IO7)**: Data bus output.
- Data (IN) (IO0 to IO7)**: Data bus input.

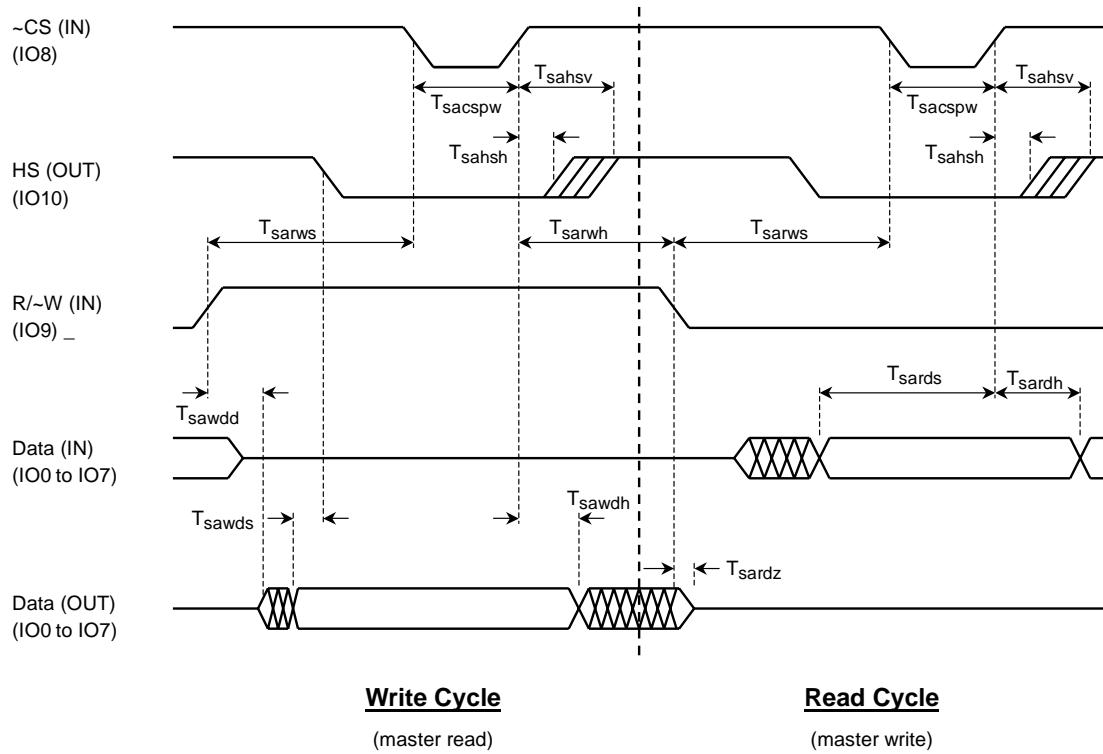
Key timing parameters are indicated:

- T_{mcspw} : Minimum setup time before CS transitions from low to high.
- T_{mhsh} : Minimum hold time after CS transitions from low to high.
- T_{mhshy} : Minimum hold time after HS transitions from high to low.
- T_{mrws} : Minimum read wait state.
- T_{mrdz} : Minimum read data delay.
- T_{mrds} : Minimum read data setup time.
- T_{mrdd} : Minimum read data delay.
- T_{mwdz} : Minimum write data delay.
- T_{mwdh} : Minimum write data hold time.
- T_{mwdh} : Minimum write data hold time.

Parameter	Description	Min	Typ.	Max
T _{mrws}	R/~W set-up before falling edge of ~CS	150 ns	3 CLK1	—
T _{mrwh}	R/~W hold after rising edge of ~CS	100 ns	—	—
T _{mcspw}	~CS Pulse Width	150 ns	2 CLK1	—
T _{mhsh}	Handshake hold after falling edge of ~CS	0 ns	—	—
T _{mhsv}	Handshake valid after rising edge of ~CS	150 ns	10 CLK1	—
T _{mr dz}	Master tri-state DATA after rising edge of R/~W	—	—	25 ns
T _{mrds}	Read data setup before falling edge of HS	0 ns	—	—
T _{mr dh}	Read data hold after falling edge of ~CS	0 ns	—	—
T _{mwdd}	Master drive of DATA after falling edge of R/~W	150 ns	2 CLK1	—
T _{mwds}	Write data setup before rising edge of ~CS	150 ns	2 CLK1	—
T _{mw dh}	Write data hold after rising edge of ~CS	(Note1)	—	—

Note2: CLK1 represents the period of the Neuron Chip input clock (100 ns at 10 MHz)

Figure 2.28 Master Mode Timing Diagram



Parameter	Description	Min	Typ.	Max
T_{sarws}	R/~W set-up before falling edge of ~CS	25 ns	—	—
T_{sarwh}	R/~W hold after rising edge of ~CS	0 ns	—	—
T_{sacspw}	~CS Pulse Width	45 ns	—	—
T_{sahsh}	Handshake hold after rising edge of ~CS	0 ns	—	—
T_{sahsv}	Handshake valid after rising edge of ~CS	—	—	50 ns
T_{sawdd}	Slave-A drive of DATA after rising edge of R/~W	0 ns	5 ns	—
T_{sawds}	Write data setup before falling edge of HS	150 ns	$2T_{CLK1}$	—
T_{sawdh}	Write data hold after rising edge of ~CS	T_{sarwh}	$2T_{CLK1}$	—
T_{sardz}	Slave-A tri-state DATA after falling edge of R/~W	—	—	50 ns
T_{sards}	Read data setup before rising edge of ~CS	25 ns	—	—
T_{sardh}	Read data hold after rising edge of ~CS	10 ns	—	—

Note1: T_{CLK1} represents the period of the Neuron Chip input clock (100 ns at 10 MHz).

Figure 2.29 Slave-A Mode Timing Diagram



Note: CLK1 represents the period of the Neuron Chip input clock (100 ns at 10 MHz)

184

The maximum data transfer rate for the parallel I/O object is one byte per 2.4 μ s,* or 3.3 Mbps, for a NEURON CHIP operating at 10 MHz. The data rate scales proportionally to the input clock rate. This is for Slave A mode data transfers. The overhead associated with the command and length bytes, and the latency for reading the status register in the Slave B mode, must be taken into account when calculating the average data rate.

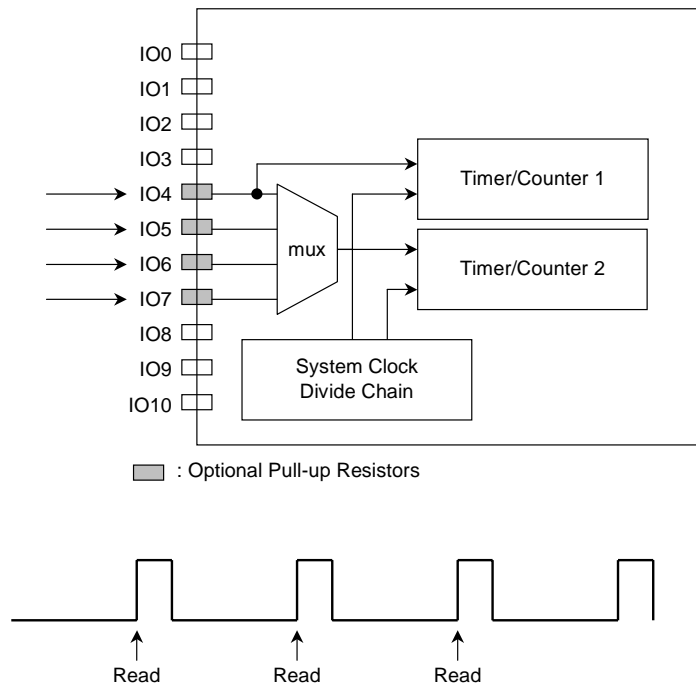
Hardware handshaking is used to control the instruction execution, and application processing is suspended for the duration of the transfer (up to 255 bytes per transfer). If the Neuron Chip is waiting for parallel I/O, a watchdog timeout will occur if the transaction does not complete within 0.84 seconds, with a 10 MHz input clock (see section 3).

For more specific information on the operation and usage of the parallel I/O object, refer to Echelon's *parallel I/O Interface to the Neuron Chip* engineering bulletin.

*: When a product supporting an input clock of 20 MHz operates at a 20-MHz clock, one byte takes 1.2 μ s.

2.23 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 3.1 (see section 3) shows the resolution and maximum time measured for various clock selections. This object is useful for instantaneous frequency or tachometer applications. Analog-to-digital conversion can be implemented using a voltage-to-frequency converter with this object.



Parameter	Description	Min	Typ.	Max
T_{fin}	Function call to input sample	—	86 μ s	—
T_{ret}	Return from function	—	52/22 μ s	—

Figure 2.31 Period Input Object

For T_{ret} , the first number represents the return latency in the case that there is a new measurement involved. The second number is for the case in which no new period value is being returned. In the latter case, the value corresponding to the previous period measurement is returned.

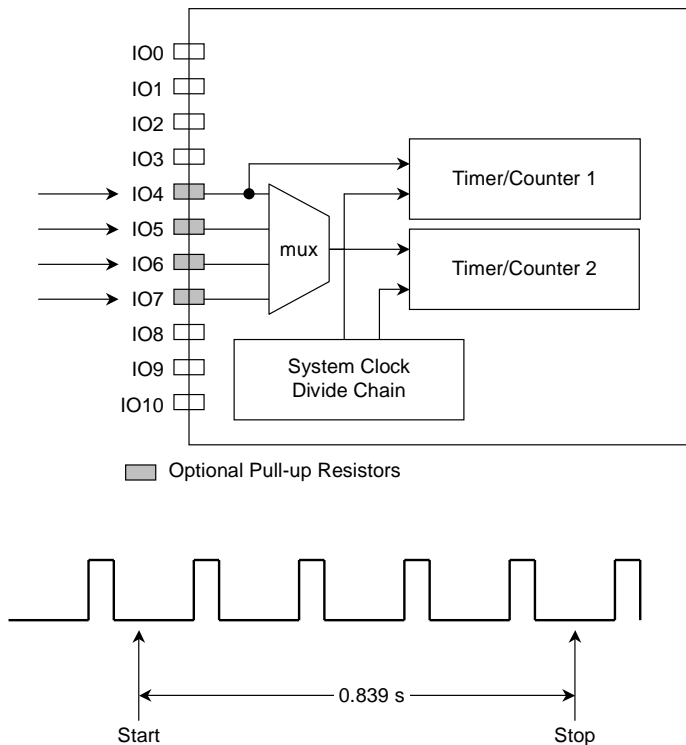
This is an edge-sensitive function. The clock driving the internal counter in the Neuron Chip is free running. The detection of active input edges stops and resets the counter each time.

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

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

2.24 Pulsecount Input

A timer/counter may be configured to count the number of input edges (up to 65,535) in a fixed time (0.839 second) at all allowed input clock rates. Edges may be defined as rising or falling. The measurement period is not synchronized with the input waveform. This object is useful for average frequency measurements, or in tachometer applications.



Parameter	Description	Min	Typ.	Max
T_{fin}	Function call to input sample	—	86 μ s	—
T_{ret}	Return from function	—	52/22 μ s	—

Figure 2.32 Pulsecount Input Object

For T_{ret} , the first number represents the return latency in the case that there is a new measurement involved. The second number is for the case in which no new pulsecount value is being returned. In the latter case, the value corresponding to the previous pulsecount measurement is returned.

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

The internal counter increments with every occurrence of an active input edge. Every 0.839 second the content of the counter is saved and the counter is then reset to 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 declaration of the function block. The default option is the negative edge.

2.25 Pulsecount Output

A timer/counter may be configured to generate a series of pulses. The number of possible pulses of output ranges from 0 to 65,535, and the output waveform is a square wave with 50% duty cycle. This object suspends application processing until the pulse train is complete. The frequency of the waveform may be any one of the eight values given by Table 3.2 (see section 3), with clock select values of 1 through 7. This object is useful for external counting devices that can accumulate pulse trains, such as stepper motors.

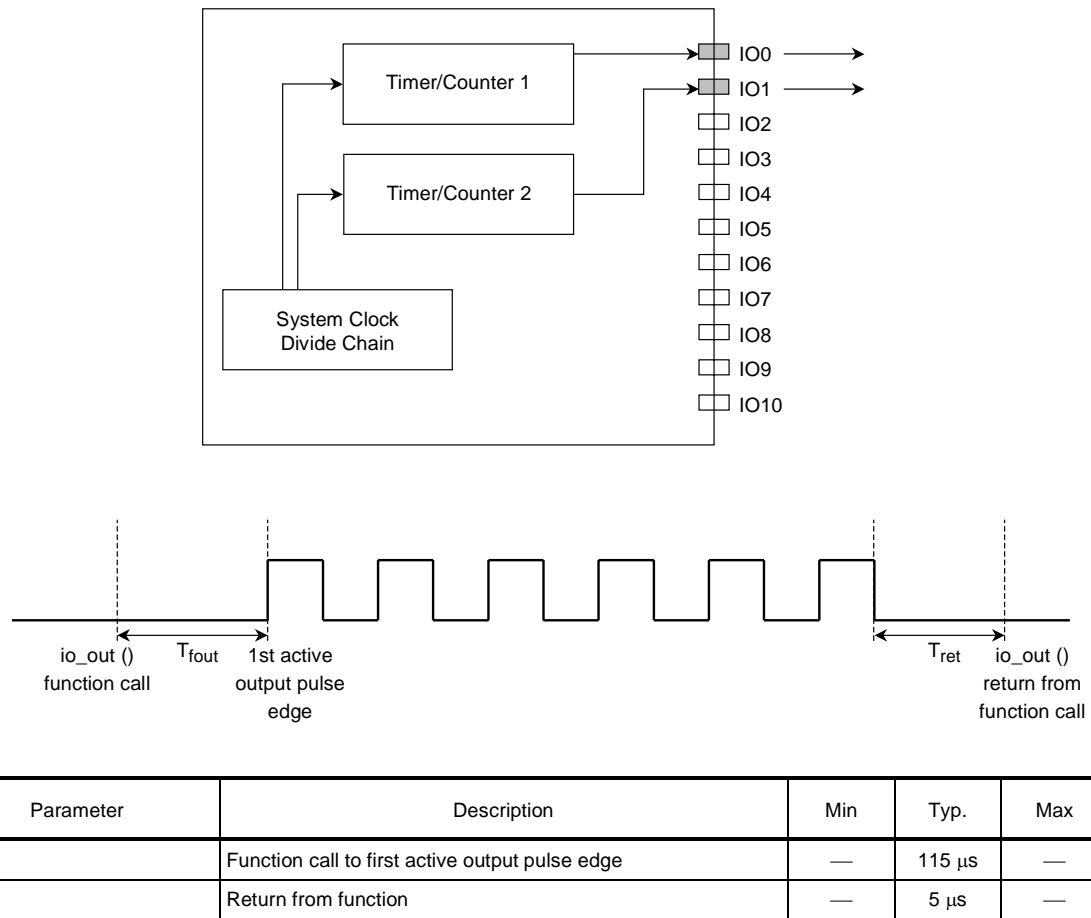


Figure 2.33 Pulsecount Input Object

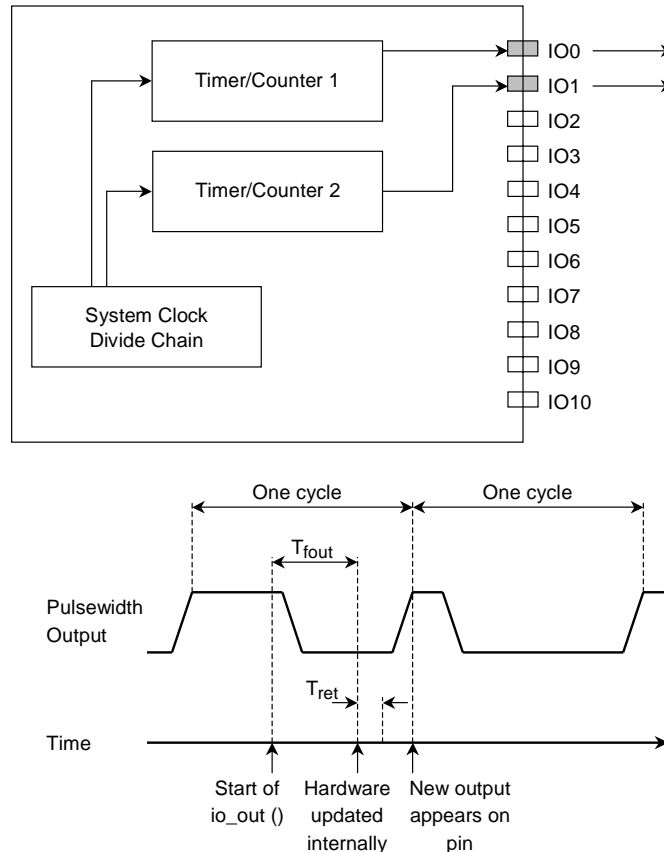
The return from this function does not occur until all output pulses have been produced.

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

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

2.26 Pulsewidth Output

A timer/counter may be configured to generate a pulse-width-modulated repeating waveform. In Pulsewidth Short mode, 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 mode, the frequency of the waveform may be any one of the eight values given by Table 3.2 (see section 3). In Pulsewidth Long mode, 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 mode, the frequency of the waveform may be any one of the eight values given by Table 3.3 (see section 3). The asserted state of the waveform may be either logic high or logic low. A pulsewidth modulated signal provides a simple means of digital-to-analog conversion.



Parameter	Description	Min	Typ.	Max
T_{fout}	Function call to output update	—	101 μ s	—
T_{ret}	Return from function	—	13 μ s	—

Figure 2.34 Pulsewidth Output Object

The new output value will not take effect until the end of the current cycle. There are two exceptions to this rule. One is that if the output is disabled, the new (non-zero) output will start immediately after T_{fout} . Secondly, 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.

*: If selected "clock 0" for the clock and "1", "2" or "3" for the output-value Neuron chip doesn't output correctly.

2.27 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 eighth of the input clock rate, for example 1.25 MHz at the maximum 10 MHz Neuron Chip input clock frequency. Quadrature devices may be connected to timer/counter 1 via pins IO6 and IO7, and to timer/counter 2 via pins IO4 and IO5.

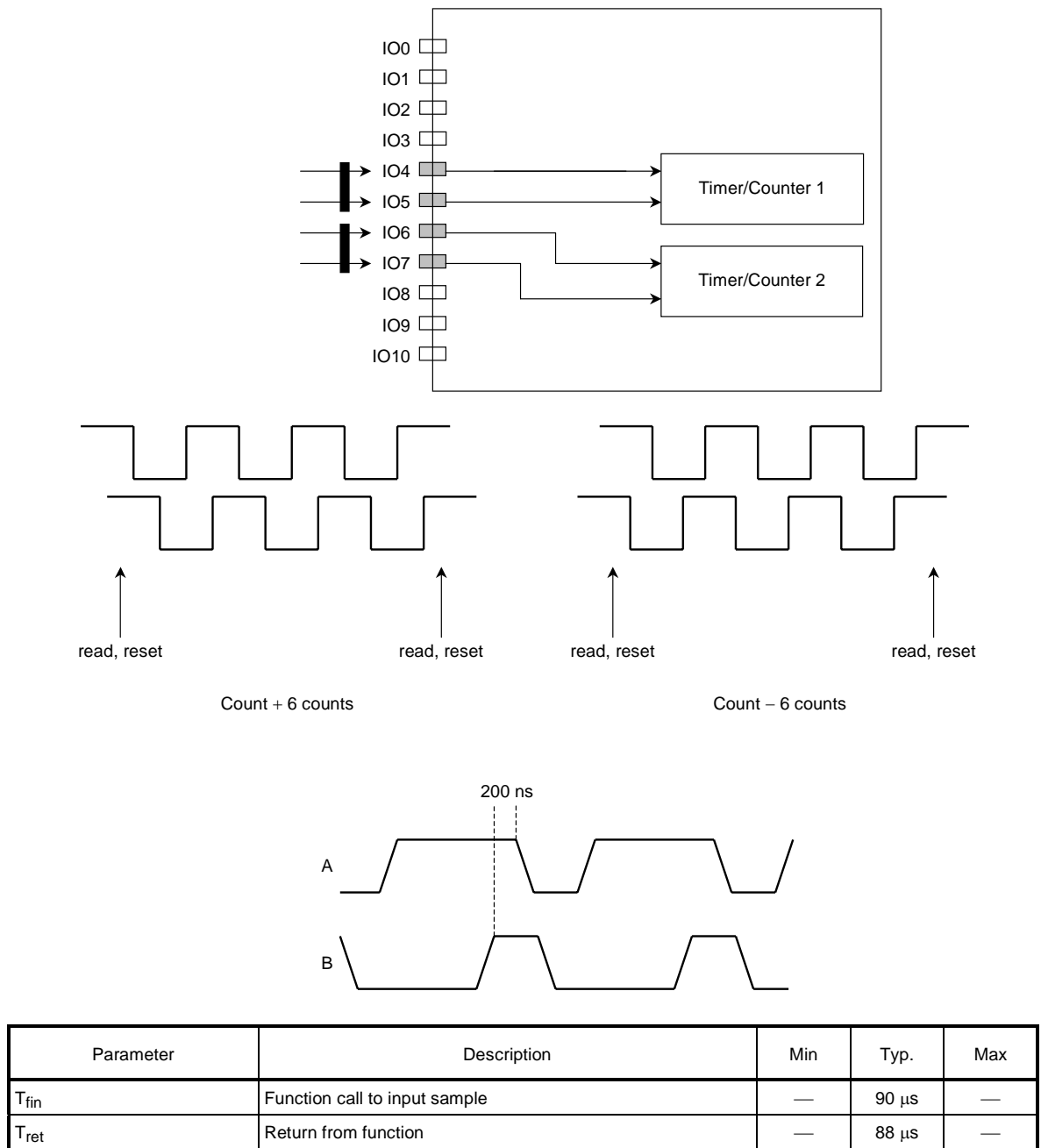


Figure 2.35 Quadrature Input Object

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.

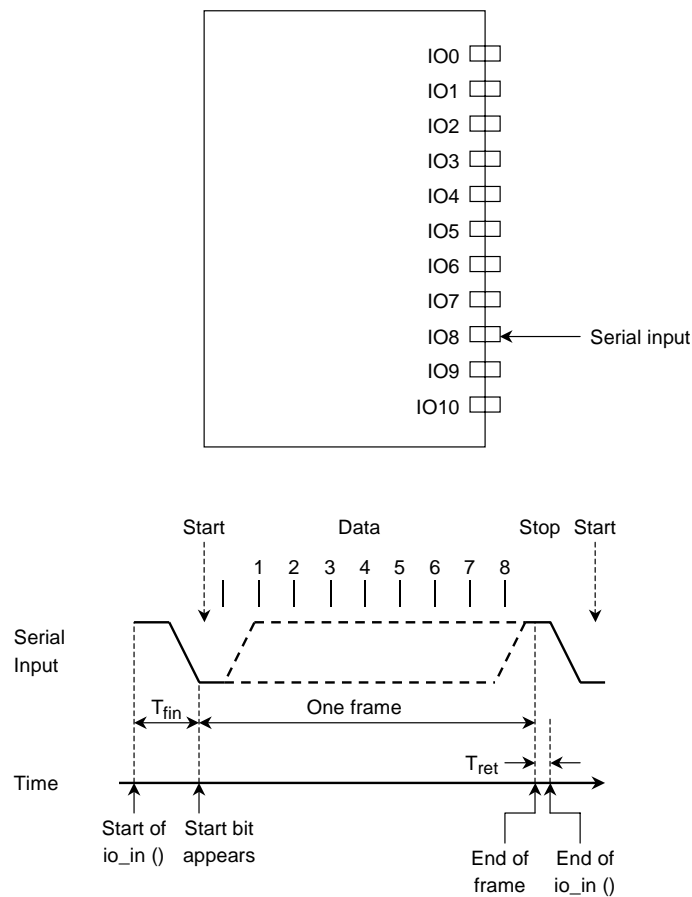
The count returned is a 16-bit signed binary number, which can range from -16,384 to 16,383.

The number shown in the diagram above is the minimum time allowed between consecutive transitions at either input of the quadrature functional block.

2.28 Serial Input

Pin IO8 may be configured as an asynchronous serial input line. The bit rates for input on this pin may be independently to be 600, 1200, 2400, or 4800 bps* at the maximum input clock rate (10 MHz). The bit 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 may be active at any one time, but not both because the interface is half-duplex only. This object suspends application processing until the operation is complete. 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 object is useful for acquiring data from serial devices such as terminals, modems, and computer serial interfaces.

*: When a product supporting an input clock of 20 MHz operates at a 20-MHz clock, selection can be made among 1200 bps, 2400 bps, 4800 bps or 9600 bps.



Parameter	Description	Min	Typ.	Max
T_{fin}	Function call to input sample min (first sample) max (time-out)	—	67 μ s 20 byte frames	—
T_{ret}	Return from function	—	10 μ s	—

Figure 2.36 Serial Input Object

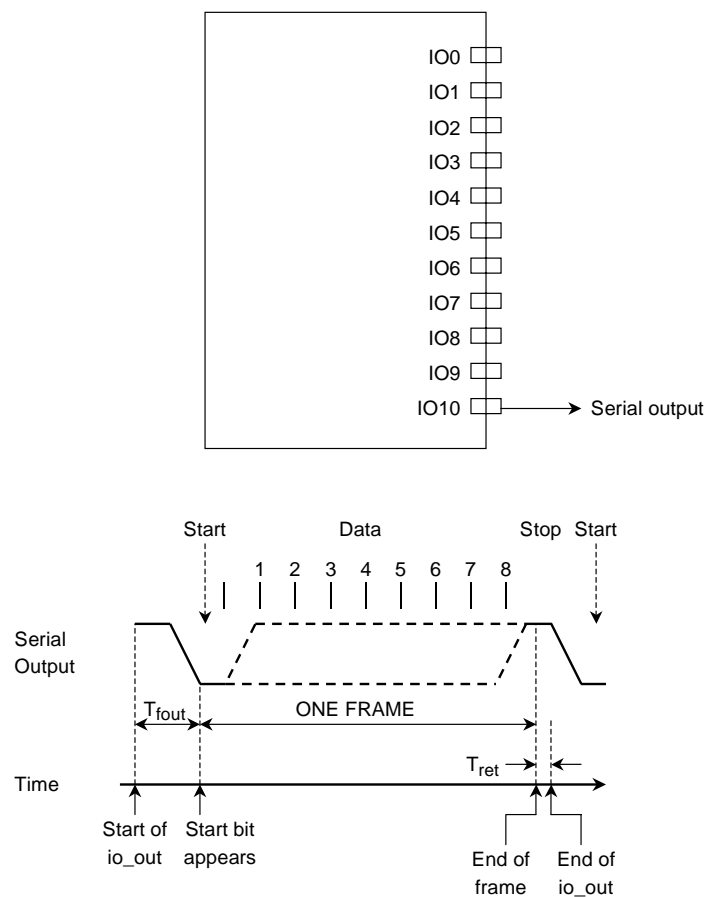
The duration of this function call is a function of the number of data bits transferred and the transmission bit rate. T_{fin} (max) refers to the maximum amount of time this function will wait for a start bit to appear at the input. After this time has elapsed, the function will return a zero as data. T_{fin} (min) is the time to the first sampling of the input pin.

As an example, the time-out period at 2400 bps is $(20 \times 10 \times 1/2400) + T_{fin}$ (min).

2.29 Serial Output

Pin IO10 may be configured as an asynchronous serial output line. The bit rates for output may be independently specified to be 600, 1200, 2400, or 4800 bps* at the maximum input clock rate (10 MHz). The bit 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 may be active at any one time, but not both because the interface is half-duplex only. This object suspends application processing until the operation is complete. The application code can use bit I/O pins flow-control handshaking if required. This object is useful for transferring data to serial devices such as terminals, modems, and computer serial interfaces.

*: When a product supporting an input clock of 20 MHz operates at a 20-MHz clock, selection can be made among 1200 bps, 2400 bps, 4800 bps or 9600 bps.



Parameter	Description	Min	Typ.	Max
T_{fout}	Function call to first start bit	—	79 μ s	—
T_{ret}	Return from function	—	10 μ s	—

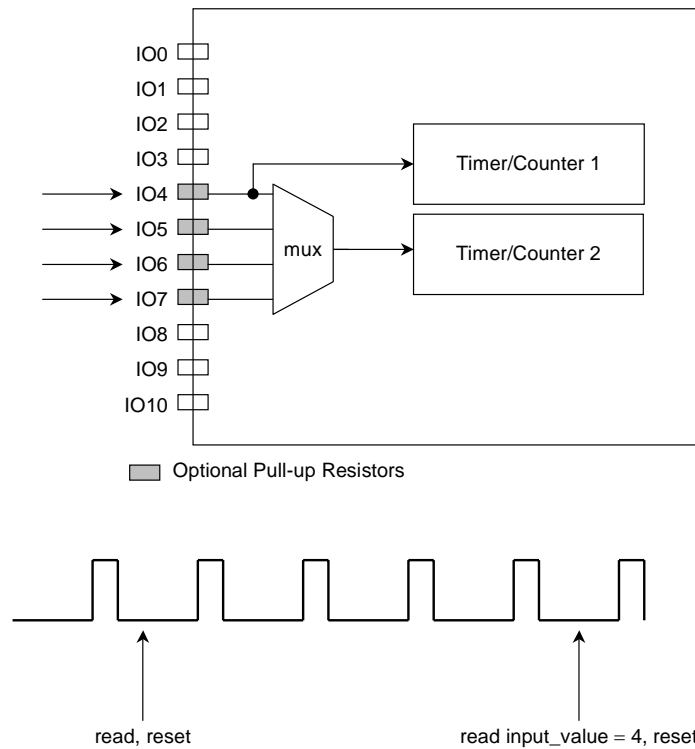
Figure 2.37 Serial Output Object

The duration of this function call is a function of the number of data bits transferred and the transmission bit rate. As an example, to output 100 bytes at 600 bps would require a time duration of $(100 \times 10 \times 1/600) + T_{\text{fout}} + T_{\text{ret}}$.

Note: when using this serial output object, the first serial output after reset may fail, because the IO10 output will become "0" in this case. Prevent this by also defining the Bit Output (bit output = 1) (overlay definition), after defining the Serial Output using the Neuron C program. The IO10 output will thus become "1" after reset and the first Serial Output will also operate correctly.

2.30 Totalcount 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 10 MHz Neuron Chip input clock frequency. This object is useful for counting external events such as contact closures, where it is important to keep an accurate running total.



Parameter	Description	Min	Typ.	Max
T_{fin}	Function call to input sample	—	92 μ s	—
T_{ret}	Return from function	—	61 μ s	—

Figure 2.38 Totalcount Input Object

A call to this function returns the current value of the totalcount variable 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 option is the negative edge.

2.31 Touch I/O

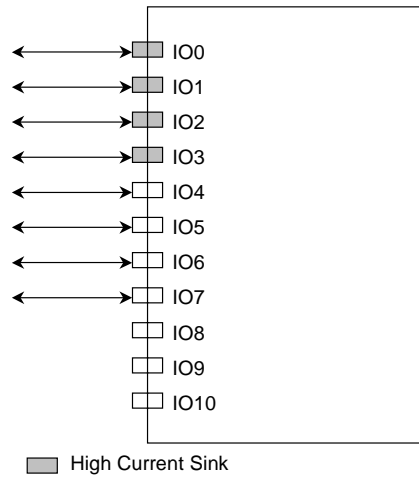
The Touch I/O object enables easy interfacing to any slave device which adheres to Dallas Semiconductor's Touch Memory™ standard. This interface is a one-word, open-drain, bi-directional connection.

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

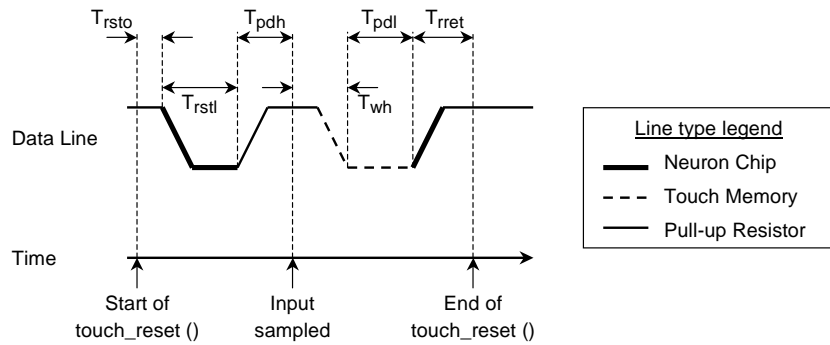
The slave acquires all the power necessary for its operation from the data line. Upon physical connection of a Touch Memory device to a master (in this case the Neuron Chip), the Touch Memory generates a low presence pulse to inform the master that it is awaiting a command. The Neuron Chip can also request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

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

Figure 2.39 shows the details of the reset pulse in addition to the read/write bit slots.



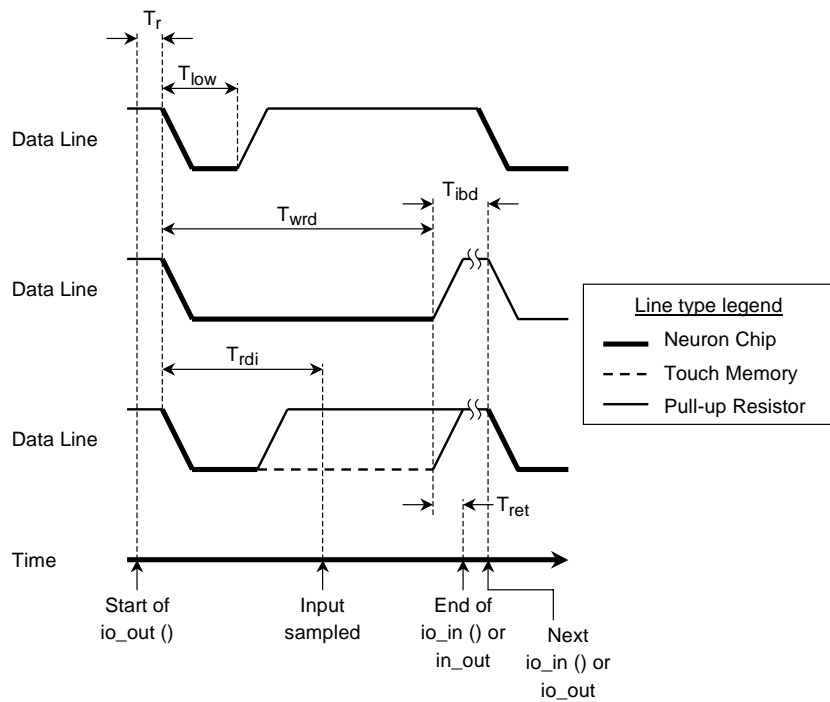
Reset
and
Presence



Write '1'

Write '0'

Read



Parameter	Description	Min	Typ.	Max
T_{rst0}	Reset call to data line low	—	60.0 μ s	—
T_{rstl}	Reset pulse width	—	500 μ s	—
T_{pdh}	Reset pulse release to data line high 10 MHz 5 MHz	4.8 μ s 9.6 μ s	—	—
T_{pdl}	Presence pulse width	—	120 μ s	275 μ s
T_{wh}	Data line high detect to presence pulse	—	80 μ s	—
T_{rret}	Return from reset function	—	12.6 μ s	—
T_f	I/O call to data line low (start of bit slot)	—	125.4 μ s	—
T_{low}	Start pulse width 10 MHz 5 MHz	—	4.2 μ s 8.4 μ s	—
T_{rdi}	Start pulse edge to sampling of input (read operation) 10 MHz 5 MHz	—	15.0 μ s 18.0 μ s	—
$T_{ wrd}$	Start pulse edge to Neuron Chip release of the data line 10 MHz 5 MHz	—	66.6 μ s 72.0 μ s	—
T_{ibd}	Inter-bit delay 10 MHz 5 MHz	—	61.2 μ s 122.4 μ s	—
T_{ret}	Return form I/O call	—	42.6 μ s	—

Figure 2.39 Touch I/O Object

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

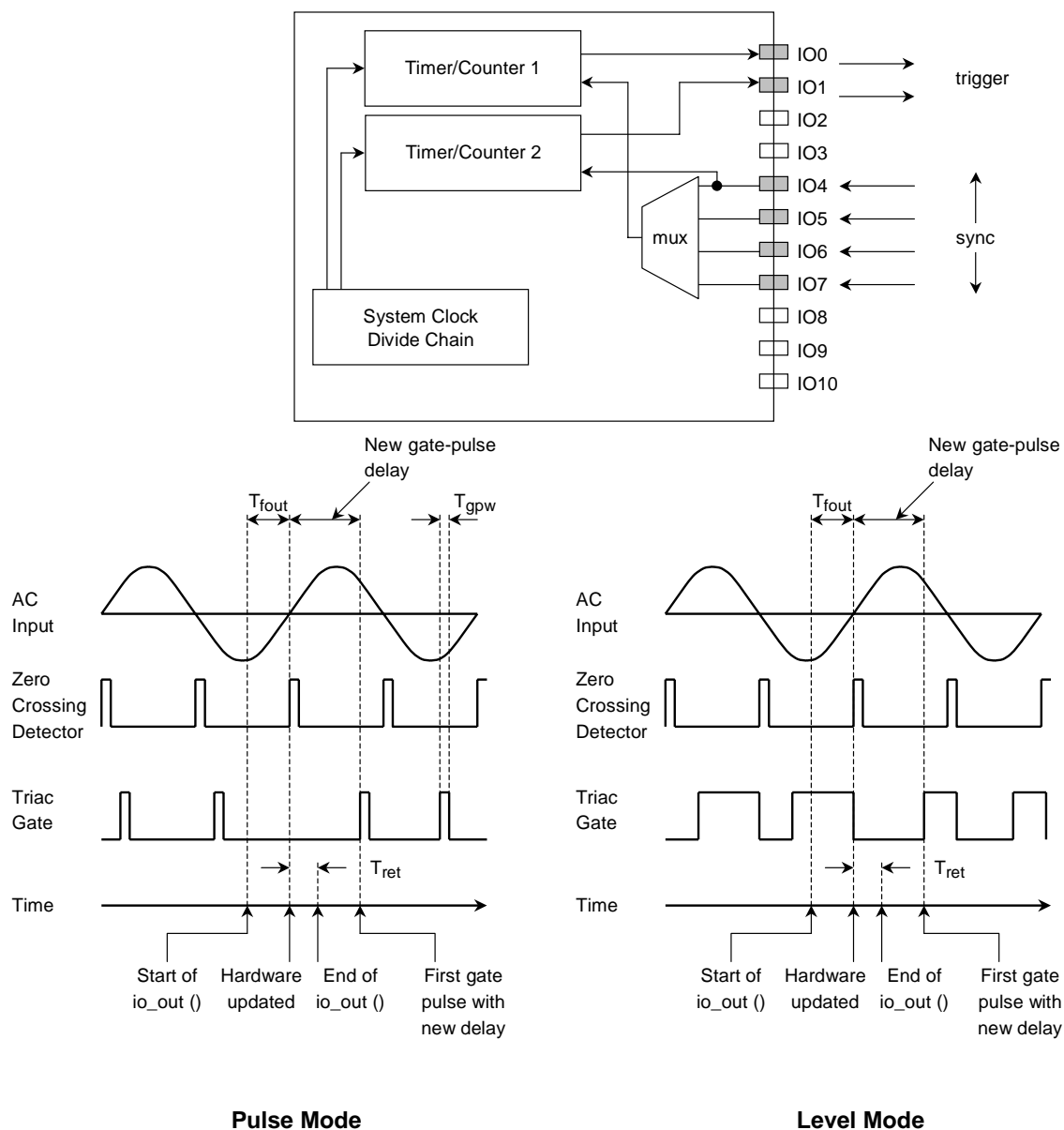
Note that the Touch I/O object can only run at Neuron Chip rates of 5 MHz and 10 MHz. This is because the Touch I/O object is designed to meet the Touch Memory timing specification only at those Neuron Chip clock speeds.

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

2.32 Triac Output

On the TMPN3150 Chip, a timer/counter may be configured to control the delay of an output signal with respect to a synchronization input. This synchronization input can occur on the rising edge, on the falling edge, or on both the rising and falling edges of the input signal. On the TMPN3120 Chip, this sync input 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/level output is the triac trigger signal.

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- μ s-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, can be useful, for example when the triac is driving a highly inductive load and the gate signal must remain active for the duration of the half cycle. Table 3.1 (see section 3) shows the resolution and maximum range of delay.



Parameter	Description	Min	Typ.	Max
T_{fout}	Function call to first start bit min (first sample) max (time-out)	—	185 μ s 10 ms	—
T_{gpw}	Gate output pulse width	—	25.6 μ s	—
T_{jit}	Gate output jitter	0	—	26.5 μ s
T_{ret}	Return from function	—	17 μ s	—

Figure 2.40 Serial Output Object

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 the triac gate pulse delay value is again updated.

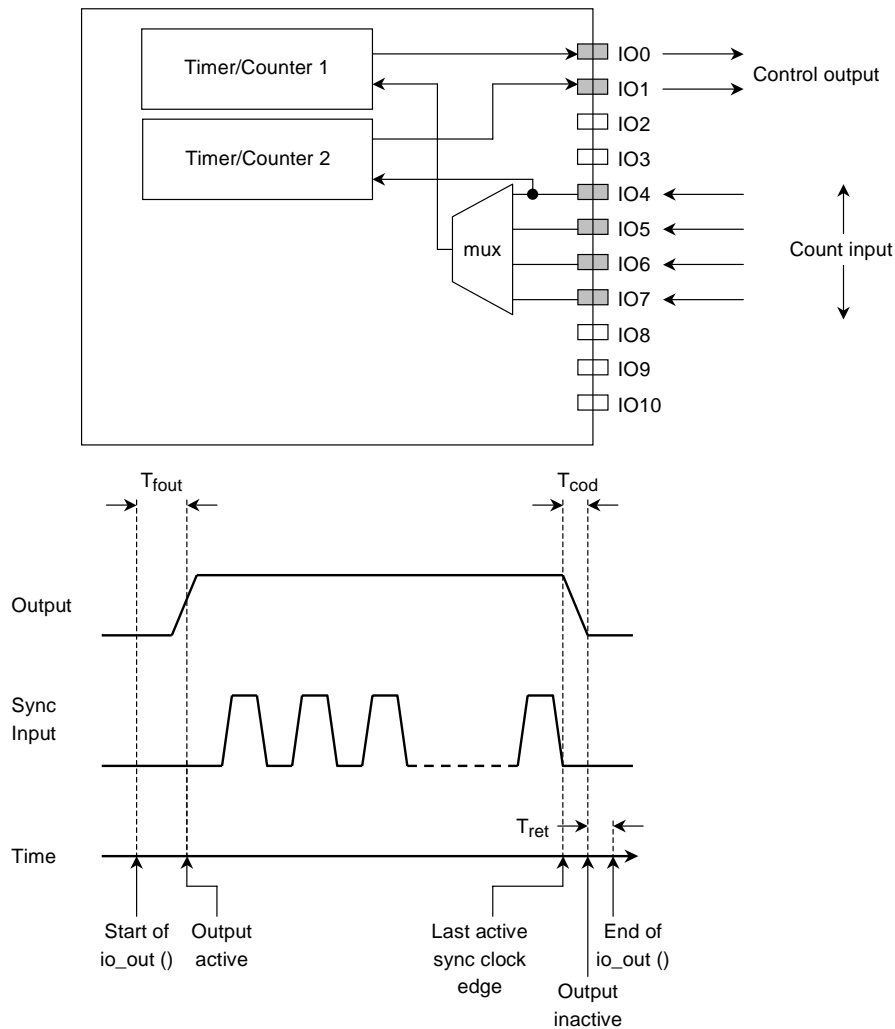
$T_{\text{fout}}(\text{min})$ refers to the delay from the initiation of the function call to the first sampling of the sync input. In the absence of an active sync clock edge, the input is repeatedly sampled for 10 ms (1/2 wave of a 50 Hz line cycle time, $T_{\text{fout}}(\text{max})$), during which application processing 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). Because 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 edges of the sync input and the triac gate output can be set by using the clock edge or invert parameters respectively.

2.33 Triggeredcount 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 a second pin. Assertion may be either high or logic low. This object is useful for controlling stepper motors or positioning actuators which provide position feedback in the form of a pulse train. The drive to the device is enabled until it has moved the required distance, and then the device is disabled.



Parameter	Description	Min	Typ.	Max
T_{fout}	Function call to output update	—	109 μ s	—
T_{cod}	Last negative sync clock edge to output inactive	550 ns	—	750 ns
T_{ret}	Return from function	—	7 μ s	—

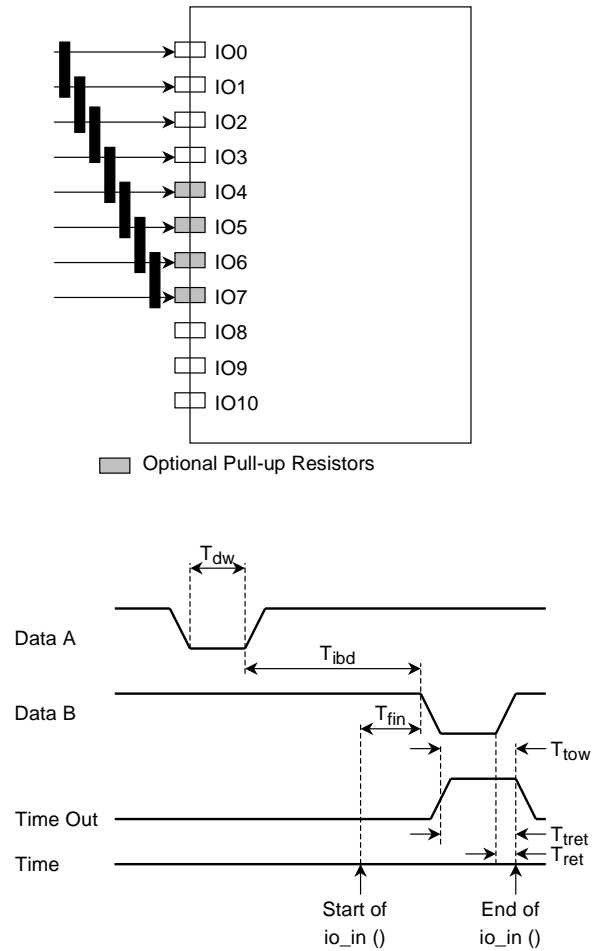
Figure 2.41 Triggeredcount Output Object

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

2.34 Wiegand Input

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

Wiegand data start as a negative going pulse on one of the two pins selected. One input is selected through the I/O declaration to represent a logical '0' bit and the other pin the represents logical '1'. The bit data on the two lines are mutually exclusive and must be spaced at least 150 μ s apart. Figure 2.42 shows the timing relationships between the two data lines and with respect to the Neuron Chip.



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

Figure 2.42 Wiegand Input Object

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

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

3. 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 I/O objects, is due to the shift-and-mask operation performed by the Neuron Chip firmware.

For dualslope input, edgelog input, ontime input, and period input the timer/counter returns a value (or a table of values, in the case of edgelog input) in the range from 0 to 65,535, representing elapsed times from zero up to the maximum of the range given in Table 3.1.

For one-shot output, frequency output, and triac output, the timer/counter may be programmed with any number in the range from 0 to 65,535. This number represents the waveform on-time for one-shot output, the waveform period for frequency output, and the control period from sync input to pulse or level output for the triac output. Table 3.1 gives the ranges and resolutions for these timer/counter objects. 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 3.1 Timer/Counter Resolution and Maximum Range

Clock Select	One-Shot and Triac Outputs; Dualslope, Edgelog, Ontime, and Period Inputs		Frequency Output	
	Resolution (μs)	Maximum Range (ms)	Resolution (μs)	Maximum Range (ms)
0	0.2	13.1	0.4	26.21
1	0.4	26.2	0.8	52.42
2	0.8	52.4	1.6	104.86
3	1.6	105	3.2	209.41
4	3.2	210	6.4	419.42
5	6.4	419	12.8	838.85
6	12.8	839	25.6	1677
7	25.6	1,678	51.2	3,355

This table is for a 10 MHz input clock. To scale the table values correctly for other clock rates:

$$\text{Resolution } (\mu\text{s}) = 2^{(\text{clock selected number} + n)} / \text{Input Clock Rate (MHz)}$$

$$\text{Maximum Range } (\mu\text{s}) = 65,535 \times \text{Resolution } (\mu\text{s}) \times n$$

$n = 1$ for one-shot and triac output, and for dualslope, edgelog, ontime, and period input

$n = 2$ for frequency output

*: When a product supporting an input clock of 20 MHz operates at a 20-MHz clock, the frequency output resolution is from 0.2 μs to 25.6 μs and the maximum range is from 13.1 ms to 1,677 ms. Likewise, the resolution of other timers/counters is from 0.1 μs to 12.8 μs and the maximum range is from 6.55 ms to 893 ms.

for pulsewidth short output and pulsecount output, Table 3.2 gives the possible choices of repetition frequencies, except that pulsecount output may not be used with clock select 0.

Table 3.2 Timer/Counter for Pulse Train Output

Clock Select	Frequency (Hz)	Period (μs)
0	19,531	51.2
1	9,766	102
2	4,883	205
3	2,441	410
4	1,221	819
5	610	1,638
6	305	3,277
7	153	6,554

This table is for a 10 MHz input clock. To scale the table values correctly for other clock rates:

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

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

Table 3.3 gives the possible choices of repetition frequencies for pulsewidth long output.

Table 3.3 Timer/Counter Pulsewidth Long 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

This table is for a 10 MHz input clock. To scale the table values correctly for other clock rates:

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

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

As with all CMOS devices, floating I/O pins can cause excessive current consumption. To avoid this, simply declare all unused I/O pins as bit output. Alternatively, unused I/O pins may be connected to +5 V or to GND.

[12] Additional Functions

[12] Additional Functions

1. Service Pin

The service pin is used during configuration, installation, and maintenance of a LONWORKS node. The pin has both output and input functions. As an output, the service pin is driven active-low to light an external LED. The LED is lit when the node has no valid application code, or when there is an on-chip failure. The LED blinks at a 1/2 Hz rate when the node has not been configured with network address information. A logic-low input at the service pin causes the Neuron Chip to transmit a network management message on the network, containing its own 48-bit Neuron ID. To accomplish both of these functions, the pin is multiplexed between input and output at a 76 Hz rate with a 50% duty cycle (see Figure 1.1). The service pin has an optional on-chip pull-up to bring the input to an inactive-high state for use when the LED and pull-up resistor are not connected.

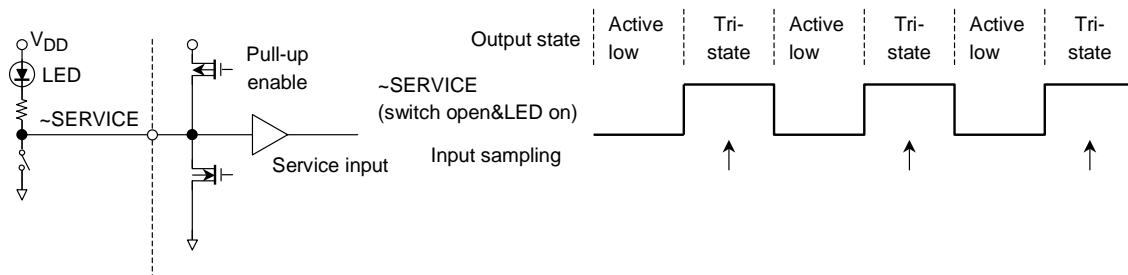


Figure 1.1 Service Pin Circuit

2. Sleep/Wake-Up Circuitry

The Neuron Chip may be put into a low-power sleep mode under software control. In this mode, the oscillator, system clock, communications port, 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 pins:

- Service pin (not maskable)
- I/O pins (maskable):
 - Any one from IO4 through IO7, selected by the timer/counter multiplexer
- Communications port (maskable):

Single-Ended Mode	-CP0
Differential Mode	-CP0 or CP1
Special-Purpose-mode	-CP3

The application software may optionally specify that the programmable pull-ups on the service pin and on I/O pins IO4 to IO7 are to 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 \sim E pin is held inactive (high) during sleep to disable memory operations during this period. The address bus is driven high (0xFFFF) during sleep to de-select all external devices. The data bus is in the output state (retaining the last value it had just before going to sleep) to keep the data lines from floating and to conserve the need for additional current.

When a wake-up event is detected (a transition on the service pin, selected I/O pin, or communications port wake-up pin), the Neuron Chip allows the oscillator to startup, waits for it to stabilize, performs internal maintenance, and then resumes operation. Typical oscillator start-up times are discussed below (in section 5, *Clocking System*). After the oscillator has started up, the Neuron Chip allows up to 15 transitions on CLK1 for the oscillator to stabilize.

The amount of time required for the Neuron Chip to perform internal maintenance after wakingup, before it resumes execution of the application, depends on several application parameters and on whether the Network CPU is servicing application timers during this period. In this respect the most important application parameters are the comm ignore option (see *neuron programmer's guide*), the number of receive transactions (if comm ignore is used), and the number of application timers (if the Network CPU 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 the worst case is about 47,000 CLK1 cycles. The typical case assumes that the Network CPU does not service the application timers during this period. The worst case assumes that the Network CPU 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 for internal maintenance, and the worst case is about 66,000 CLK1 cycles. The typical case assumes that 4 receive transactions are specified, and that the Network CPU does not service the application timers during this period. The worst case assumes that the maximum number of receive transactions (16) are specified, and that the Network CPU must service the maximum number of application timers (15) during this period.

3. Watchdog Timer

The Neuron Chip CPUs are protected against malfunctioning software or memory faults by three watchdog timers (one per CPU). 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 suspended.

*: When a product supporting an input clock of 20 MHz operates at a 20-MHz clock, the watchdog cycle is about 0.42 second.

4. Reset Circuit

The RESET pin for the Neuron Chip (\sim RESET) is configured for an I/O circuit having open-drain output with an internal pull-up resistor. When the reset pin is held at a voltage of at least 20 ns and less than 0.8 V, the reset operation begins.

The internal RESET circuit for TMPN3150B1AFG, TMPN3120FE3MG and TMPN3120FE5MG are shown as Figure 4.1, Figure 4.2 and Figure 4.3.

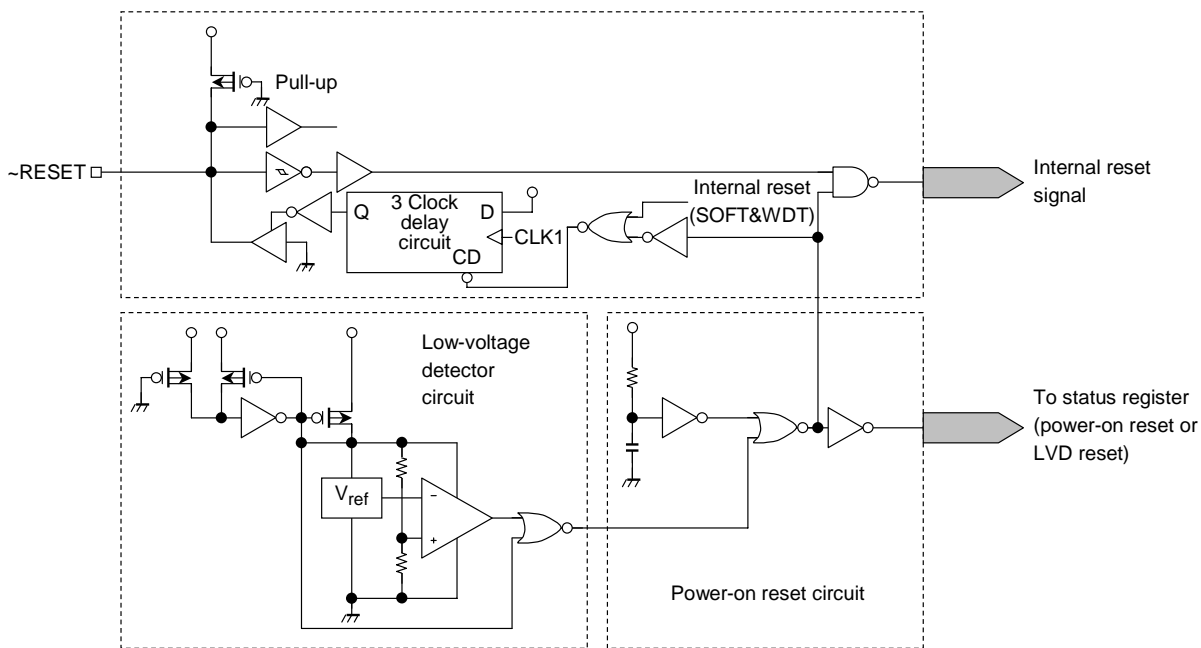


Figure 4.1 Internal RESET Circuits for TMPN3150B1AFG

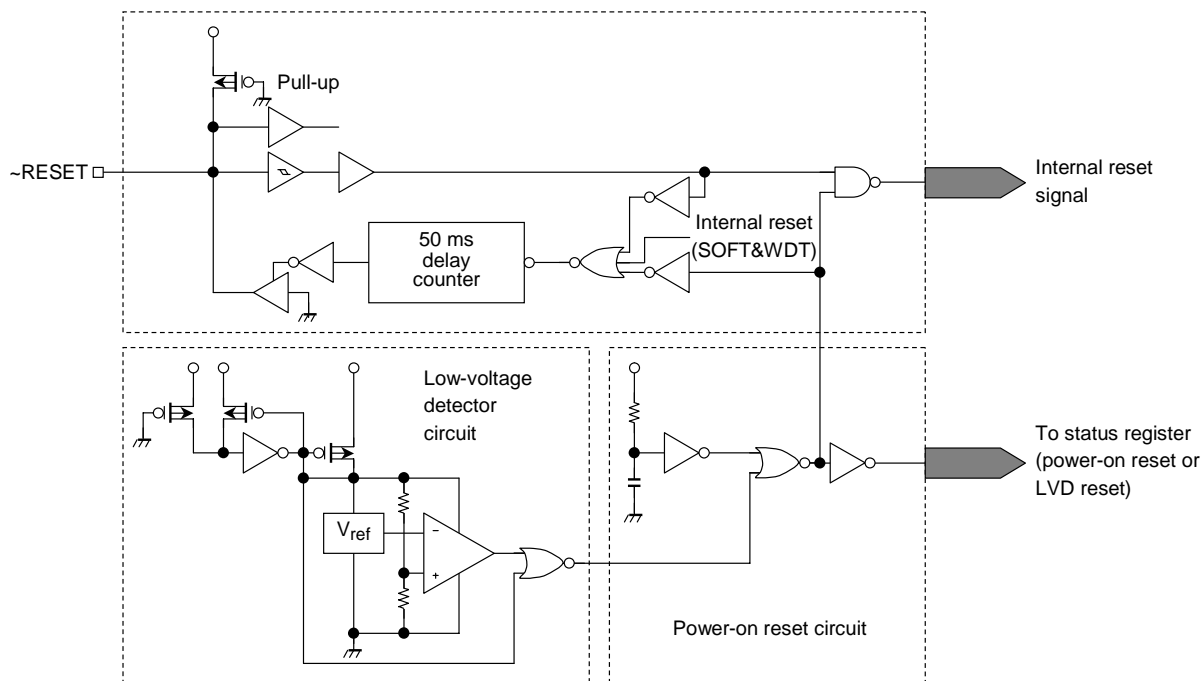


Figure 4.2 Internal RESET Circuits for TMPN3120FE3MG

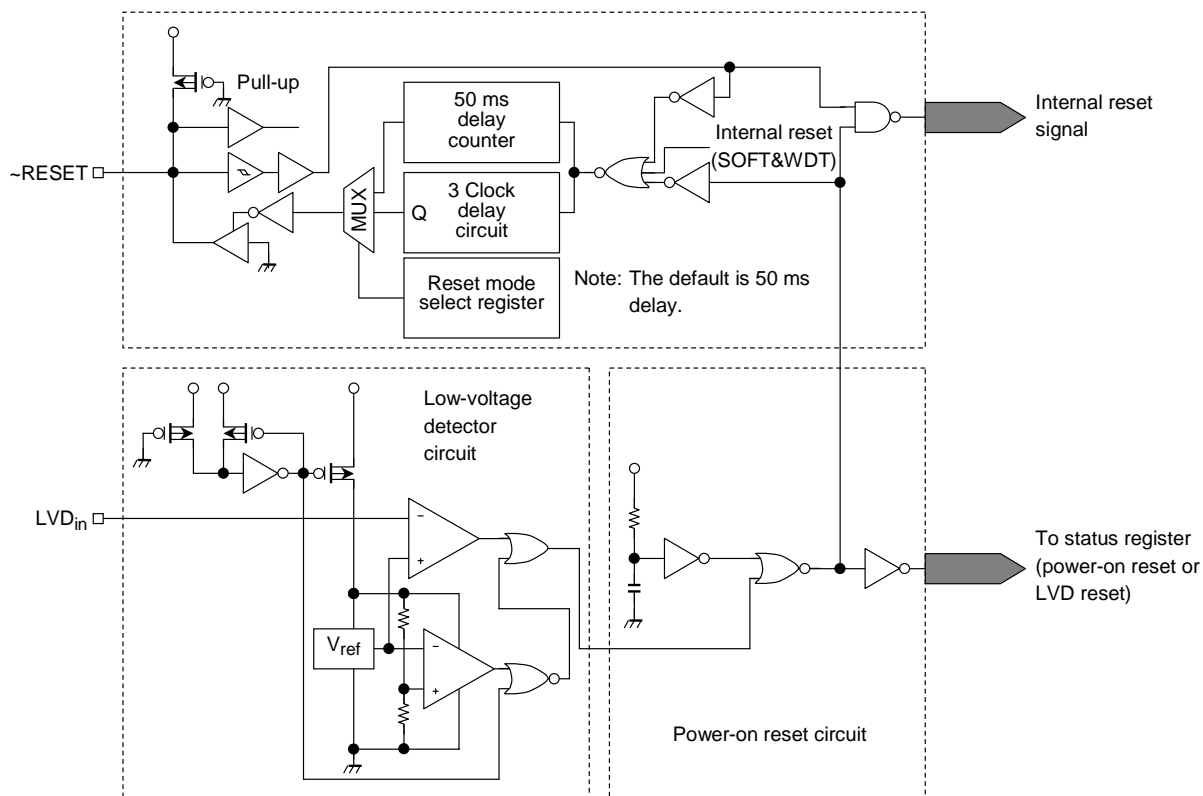


Figure 4.3 Internal RESET Circuits for TMPN3120FE5MG

The LVD (low voltage detector) circuit is incorporated in the reset circuits shown in Figure 4.1 to Figure 4.3. When the supply voltage (V_{DD}) is less than the threshold value (V_{LVD}), the \sim RESET pin is set low and internal reset signals are generated simultaneously. The LVD reset, the internal software reset the watchdog timer (WDT) reset, and the internal power-on reset signals drive the \sim RESET signals low, and then the \sim RESET pin is held for three clock cycles ($3 \times CLK1$) by the internal three-clock delay circuits or held about 50 ms at least by the 50 ms delay counter. This creates the reset signal hold time for external devices in the case of reset operations for external devices from Neuron Chip \sim RESET signals. When such a reset hold time is insufficient, or in order to protect the reset operation from interference, the \sim RESET pin should be connected to ground through capacitor C_e . Refer to Figure 4.4, where the C_e and the external reset are shown to be connected to the \sim RESET pin. When the Neuron Chip outputs reset signals, the C_e capacitor needs to be discharged, so make sure that it does not exceed 1000 pF.

In the case that Neuron Chip is accidentally reset due to noise, adding external pull-up resistor to \sim Reset might be one of measures to prevent it.

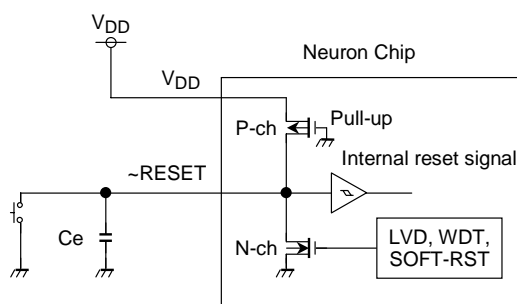


Figure 4.4 Neuron Chip Reset Circuit

The above circuit is sufficient when TMPN3120 $\times\times$, which incorporates an LVD circuit, operates by itself at 10 MHz or less. However, when a product which supports an input clock of 20 MHz operates at 20 MHz, or when as in the case of TMPN3150B1AFG external memory directly drives operation of the Neuron Chip, an external circuit such as LVD circuit may be required.

When connecting external LVD or LVI (low voltage interrupt), open-drain or open-collector type devices must be directly connected to the \sim RESET pin of the Neuron Chip.

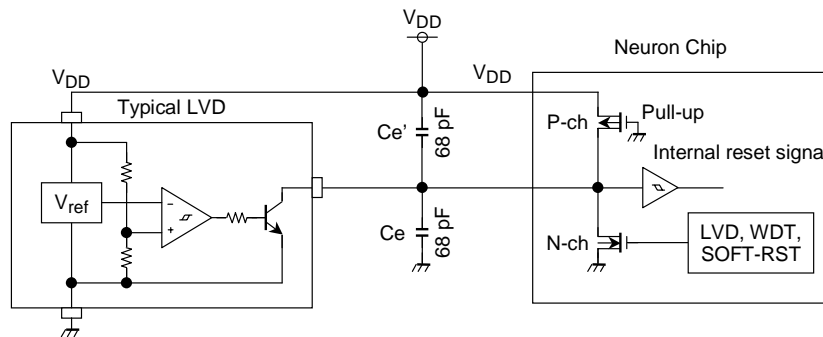


Figure 4.5 External ReACset Circuit with Low-Voltage Detection

Typical low-voltage detector devices include the Mitsumi PST572 series, the Motorola MC34064 and MC34164, and the Dallas Semiconductor DS1233 series. Select any of them after determining the trip point reset voltage, etc.

Conditions for external LDV:

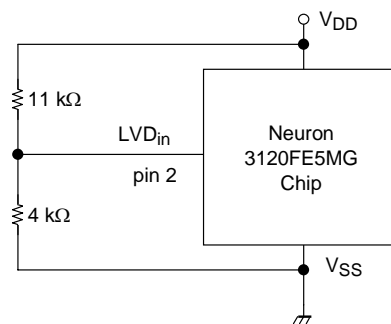
1. Open-drain or open-collector type
2. Minimum operational voltage: $V_{DD} \geq 1.5 \text{ V}$
3. Reset pulse stretching type (when using external EEPROM)

5. Programmable Low Voltage Detection

The existing low voltage detector (LVD) on the Neuron Chip trips whenever the V_{DD} input is less than 4.15 ± 0.35 V. The programmable LVD feature allows the user to select a higher trip voltage if necessary. One of the V_{DD} pins of the 3120 chips is reassigned as the LVD control pin (LVD_{in}). The programmable LVD trips if the LVD_{in} input is below 1.2 ± 0.2 V or V_{DD} is below 4.15 ± 0.35 V. This behavior is backward compatible with earlier Neuron Chips; if the LVD_{in} input is connected to V_{DD} , only the 4.15 V LVD is operational. The LVD_{in} is pin 2 for the SOP-32 package.

Example 1

Increasing Default LVD Trip Point



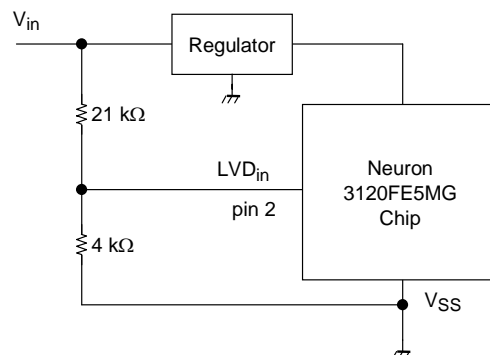
The voltage divider sets the LVD_{in} voltage to $V_{DD} \times 4/(11 + 4)$

The LVD_{in} trip point is 1.2 ± 0.2 V. This translates to a V_{DD} trip point of 4.5 ± 0.75 V.

Note: TMPN3120FE5MG only have the programmable low voltage detection.

Example 2

Early Warning Low Voltage Detection



The voltage divider sets the LVD_{in} voltage to $V_{in} \times 4/(21 + 4)$

The LVD_{in} trip point is 1.2 ± 0.2 V. This translates to a V_{DD} trip point of 7.5 ± 1.25 V.

6. Programmable Reset Time

The reset signal is asserted for either 3 clock cycles (same as earlier neurons), or 50 ms minimum.

This feature is controlled by bit 0 of I/O address 0xFFFF5 as follows:

0 = 50 ms, 1 = 3 cycles.

After every reset, this control cleared, setting the reset time to 50 ms. If the application desires the shorter (3 cycle) reset time, it should set the control bit in its reset processing task.

Example;

```
when ( reset ) {  
    *(int *) 0xFFFF5 = 1;  
}
```

7. Reset Processes and Timing

Asserting the \sim RESET pin causes the Neuron Chip to perform a fixed set of hardware and firmware initialization tasks. These tasks configure the state of the Neuron Chip pins and memories so that it is ready to begin execution of the application. These tasks and their effects on the Neuron Chip pins are shown in Figure 4.5 The tasks are:

- Oscillator Start-up
- Oscillator Stabilization
- Stack Initialization and Built-in Self-Test
- Service Pin Initialization
- State Initialization
- Off-Chip RAM Initialization
- Random Number Seed Calculation
- System RAM Setup
- Communications Port Initialization
- Checksum Initialization
- One Second Timer Initialization
- Scheduler Initialization

During oscillator start-up, the Neuron Chip allows the oscillator enough time to create a signal swing of greater than approximately 1.7 volts. This duration depends on the type of oscillator used and its frequency (see section 6). This period begins as soon as power is applied to the oscillator, and is independent of the \sim RESET pin. The oscillator start-up period may end before or after \sim RESET is released, depending on the duration of reset and on the time required by the oscillator to start up.

Oscillator stabilization begins when both the \sim RESET pin has released, and the oscillator has started up. The Neuron Chip waits for up to 15 transitions on CLK1 to allow the oscillator's frequency to stabilize. From the time \sim RESET is asserted until the end of the oscillator stabilization period, the I/O pins, communications port pins, and \sim SERVICE pin are in a high-impedance state. The \sim E signal goes inactive (high) immediately after \sim RESET goes low, and the address bus becomes high (0xFFFF) to de-select external devices. **Note that pulling the \sim RESET pin low externally while \sim E is low could result in the \sim E signal going high prematurely. For external devices that depend on a full low duration of the \sim E signal, the external reset signal should be synchronized with the rising edge of \sim E.**

The Stack Initialization and Built-in Self-Test Task tests the on-chip RAM, the timer/counter logic, and the counter logic. If the RAM fails its self-test, the node goes offline, the Service LED comes on solid, and an error is logged in the node status structure (see the query status command description in section B.12). To pass the test, all three CPUs as well as the ROM must be functioning. 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 \sim SERVICE pin oscillates between a solid low a weak high. The memory interface signals reflect execution of these tasks.

The Service Pin Initialization Task simply turns off the \sim SERVICE pin and disables IO [7:4] pull-ups.

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 the boot ID is blank, if the boot ID does not match the boot ID in ROM (for the TMPN3150 chip only), or if the reboot word in ROM (defined by the LonBuilder export process) specifies such action. Refer to the *LonBuilder User's Guide* for more information on the reboot word.

When data at the specific address in EEPROM are 0X00 or 0XFF, the Neuron Chip decides that the data are blank. The byte is at offset 0X09 in the configuration structure (appendix A.6).

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 to begin by clearing these two bytes and then resetting the Neuron Chip.

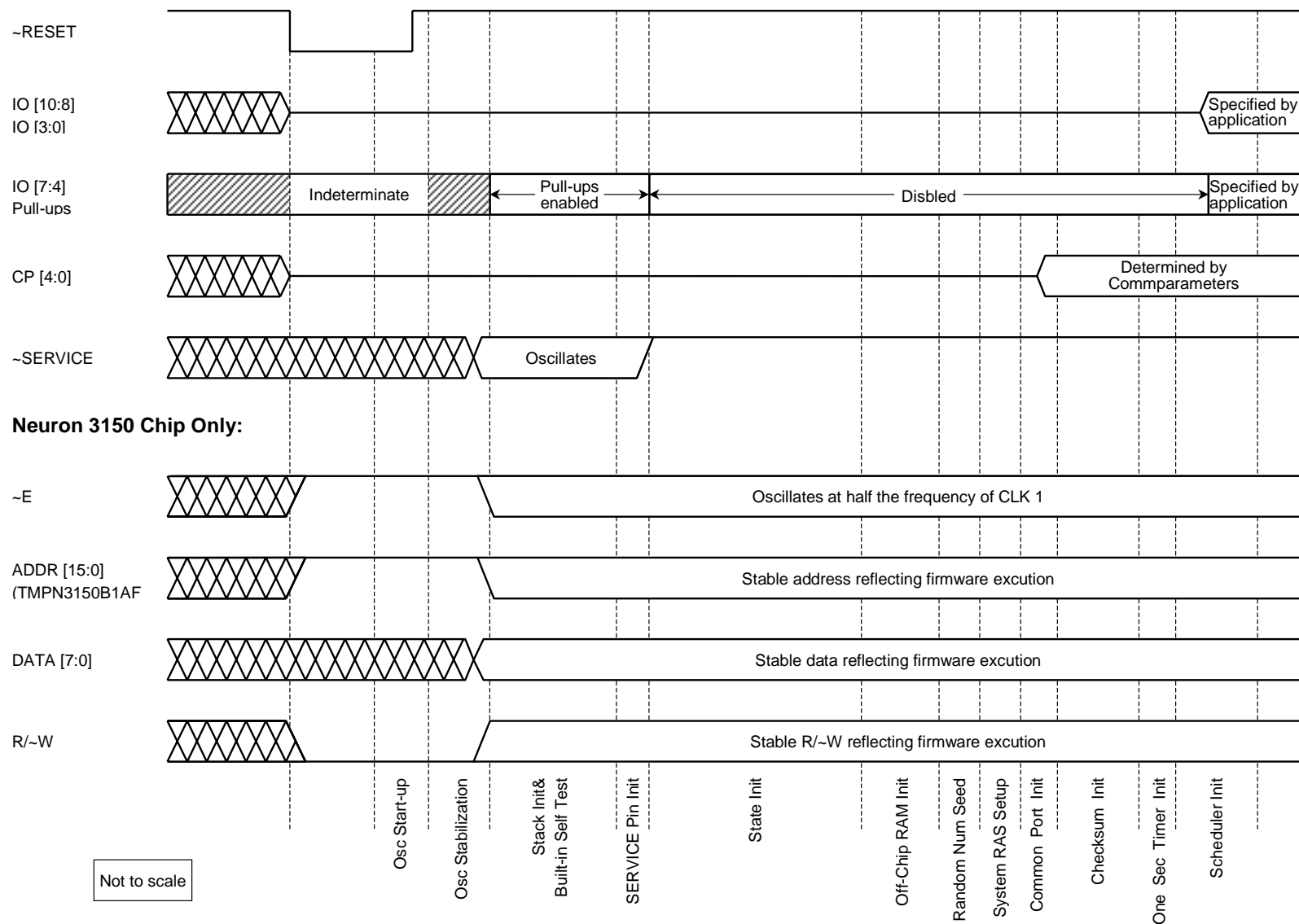


Figure 7.1 Reset Timing

The boot process varies according to the particular model of Neuron Chip being used. For the TMPN3120×× Chips, it simply copies the default communication parameters and mode table from ROM into EEPROM. For the TMPN3150 Chip, it copies a variable amount of data from ROM to EEPROM. The amount of data varies as a function of the target state of the node. If the node is to come up applicationless, the boot process is the same as that for the TMPN3120 Chip. To come up in either the configured or unconfigured state, the boot process must also copy the configuration and code areas of on-chip EEPROM which 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 Neuron C pragma statement. This task applies only to the TMPN3150 Chip.

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 Communications port Initialization Task initializes the communications port according to the application-specified communications 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 non-volatile writeable memories. If the boot process was executed for the configured or unconfigured states, in the State Initialization Task, then new checksums are generated; otherwise, they are only checked. This process includes on-chip EEPROM, off-chip EEPROM, and off-chip non-volatile RAM. ROM is not checksummed. There are two checksums, one for the configuration image, and one for the application image.

Note that the application image checksum includes the configuration image; that is, the configuration image is checksummed twice. 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 images.

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 procedures as follows:

- State Wait—wait for the node to leave the applicationless state.
- Pointer Initialization—perform a global pointer initialization.
- Initialization Step—execute an initialization task which is created by the compiler/linker, to initialize static variables and the timer/counters.
- I/O Pin Initialization Step—initialize I/O pins based on an application definition. Prior to this point, I/O pins are tri-stated.
- State Wait II—wait for the node to leave the unconfigured or hard-offline state. If waiting was required, a flag is set to indicate that the node should come up offline.
- Parallel I/O Synchronization—nodes using parallel I/O attempt to execute the master/slave synchronization protocol.
- Reset Task—execute the application reset task.
- 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 applicationless, configured, or unconfigured; amount of off-chip RAM; whether the off-chip RAM is tested, or simply cleared; the number of buffers allocated; and application initialization. Tables 6.1 and 6.2 summarize the number of input clock cycles (CLK1) required for each of these steps, for the TMPN3120B1AM Chip and the TMPN3150 Chip. The times are approximate and are given as functions of the most significant application variables.

Table 7.1 Time Required for the TMPN3120B1AM Chip to Perform Reset Sequence

Step	Approximate Number of CLK1 Cycles	Notes
Oscillator Start-up	See Section 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	$21,000 + 600 \times B$	2
Communications Port Initialization	0	1
Checksum Initialization	$3400 + 175 \times M$	3
One-Second Timer Initialization	6100	
Scheduler Initialization	≥ 7400	4

Note1: These tasks run in parallel with other tasks

Note2: B is the number of buffers allocated

Note3: M is the number of bytes to be checksummed

Note4: Assumes a trivial initialization task, no reset task, and the configured state

For example, the timing of each of these steps is shown below for a TMPN3120B1AM Chip application with the following parameters: 10 MHz input clock, crystal oscillator, no boot required, minimum number of buffers, and checksum performed on 500 bytes of EEPROM.

Oscillator Start-up	1.6 ms	(from Table 8.3)
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	
Communications Port Initialization	0	
Checksum Initialization	10.8 ms	
One Second Timer Initialization	0.61 ms	
Scheduler Initialization	<u>0.74 ms</u>	
* Total	36.6 ms	

*: The total is added approximately 45 ms for TMPN3120FE3MG and added approximately 100 ms for TMPN3120FE5MG.

Table 7.2 Time Required for the TMPN3150 Chip to Perform Reset Sequence

Step	Number of CLK1 Cycles	Notes
Oscillator Start-up	See Section 5, Clocking System	
Oscillator Stabilization	15	
Stack Initialization and Built-in Self-Test	425,000	
Service Pin Initialization	1,000	
State Initialization	1,300 (for no boot) 325,000 + 25 ms*E (for boot)	1
Off-Chip RAM Initialization	24,000 + 214*R (for test and clear) 24,000 + 152*Ra (for clear only)	2 3
Random Number Seed Calculation	max 50,000	
System RAM Setup	27,000 + 1500*B	4
Communications Port Initialization	0	5
Checksum Initialization	36,000 + 175*M (for no boot) 82,000 + 100 ms + 175*M (for boot)	6 6, 7
One-Second Timer Initialization	6,100	
Scheduler Initialization	≥ 7,400	8

Note1: E is the number of non-zero bytes being written (ranges from 10 to 504)

Note2: R is the number of off-chip RAM bytes

Note3: Ra is the number of non-system off-chip RAM bytes

Note4: B is the number of buffers allocated

Note5: These tasks run in parallel with other tasks

Note6: M is the number of bytes to be checksummed

Note7: Only if booting to the configured or unconfigured state; if booting to the applicationless state, use the “no boot” equation

Note8: 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 TMPN3150 Chip application with the following parameter: 10 MHz input clock, crystal oscillator, no boot required, 16 Kbytes external RAM, test and clear external RAM, minimum number of buffers, and checksum performed on 500 bytes of EEPROM.

Oscillator Start-up	1.6 ms	(from Table 8.3)
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	
Communications Port Initialization	0	
Checksum Initialization	12.5 ms	
One Second Timer Initialization	0.61 ms	
Scheduler Initialization	<u>0.74 ms</u>	
Total	420 ms	

8. Clocking System

The Neuron Chip includes an oscillator that may be used to generate an input clock signal, using an external crystal or ceramic resonator circuit. The transconductance of this oscillator is 2.1 milli-Siemens at minimum. The Neuron Chip may operate over a range of input clock rates from 10 MHz *(20 MHz) down to 625 kHz for low-power applications. The valid input clock frequencies are *20 MHz, 10 MHz, 5 MHz, 2.5 MHz, 1.25 MHz, and 625 kHz. Alternatively, an externally-generated clock signal may drive the CMOS input pin CLK1 of the Neuron Chip, in which case CLK2 should be left unconnected. In this case, set the rise (t_{cr}) and fall (t_{cf}) of the input clock waveform below 20 ns, respectively (refer to Figure 8.2). As for ordinary CMOS inputs, make sure there is no interference when the input waveform rises and falls. Also, limit the waveform duty ratio to within 40 to 60 %. The accuracy of the input clock frequency must remain within bounds of $\pm 1.5\%$ or better (even with temperature and voltage variations) to ensure that nodes correctly communicate on the network.

Check the transceiver manual because some transceivers require higher precision or cannot use a ceramic resonator.

The Neuron Chip divides the input clock by a factor of two, provide a symmetrical on-chip system clock. The system clock is further divided by powers of two to provide clocks for the applications I/O section, the network communications port, and the CPU watchdog timers.

*: Only a product supporting an input clock of 20 MHz

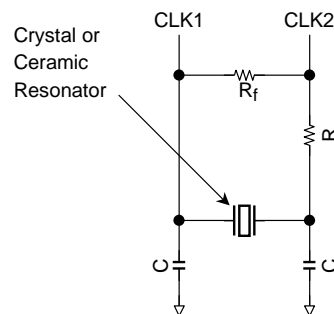


Figure 8.1 Neuron Chip Clock Generator Circuit

Note: The crystal or ceramic resonator needs to be installed as close to the Neuron Chip pin as possible in order to prevent interference from noise or other signals.

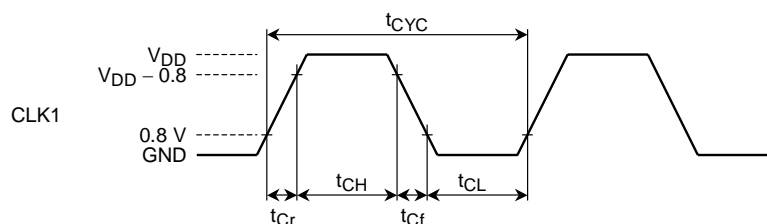


Figure 8.2 Waveform of External Clock Input to NEURON CHIP

Table 8.1 Clock Generator Component Values (see following notes)

Input Clock Frequency	Crystal		Ceramic Resonator	
	R	C	R	C
20.0 MHz (*1)	120 Ω	15 pF	120 Ω	7 pF
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	1.0 k Ω	36 pF	1.0 k Ω	36 pF
1.25 MHz	1.2 k Ω	47 pF	1.2 k Ω	47 pF
0.625 MHz	2.7 k Ω	47 pF	1.2 k Ω	100 pF

*1: These values apply only to products for which a 20-MHz input clock is available. Rf is not connected (provisional value).

Note1: The capacitor values include stray capacitances. Crystal or ceramic resonator manufacturers may recommend other values.

Note2: Rf = 100 k Ω . Rf is required with ceramic resonator configurations. With crystal configurations Rf is not required but may be used.

Note3: Crystal or ceramic resonator frequency = Input clock frequency.

Note4: Crystal may be parallel or series resonant. NPO-type ceramic capacitors are recommended. Resistor and capacitor tolerance is $\pm 5\%$. Table 8.2 lists a source for the ceramic resonators as an aid in the selection of components.

Table 8.2 Available Ceramic Resonators

Supplier	Murata Electronics North America, INC 2200 Lake Park Drive, Smyrna, GA 30080-7604, USA Phone: 1-770-436-1300 Fax: 1-770-436-3030
20.0 MHz	CSA 20.00 MTZ040
10.0 MHz	CSA 10.0 MTZ040
5.0 MHz	CSA 5.00 MG040
2.5 MHz	CSA 2.50 MG040
1.25 MHz	CSB 1250J (*2)
0.625 MHz	CSB 625P

*2: Only for TMPN3150

Typical start-up times for this clock circuit with the Neuron Chip are shown in Table 8.1. Actual start-up times vary depending on the crystal of the ceramic resonator used.

Table 8.3 Typical Start-Up Times

Input Clock Frequency	Crystal	Ceramic Resonator
20.0 MHz (*3)	0.5 ms	8 μ s
10.0 MHz	0.5 ms	8 μ s
5.0 MHz	1.0 ms	15 μ s
2.5 MHz	2.0 ms	30 μ s
1.25 MHz	3.0 ms	60 μ s
0.625 MHz	10.0 ms	380 μ s

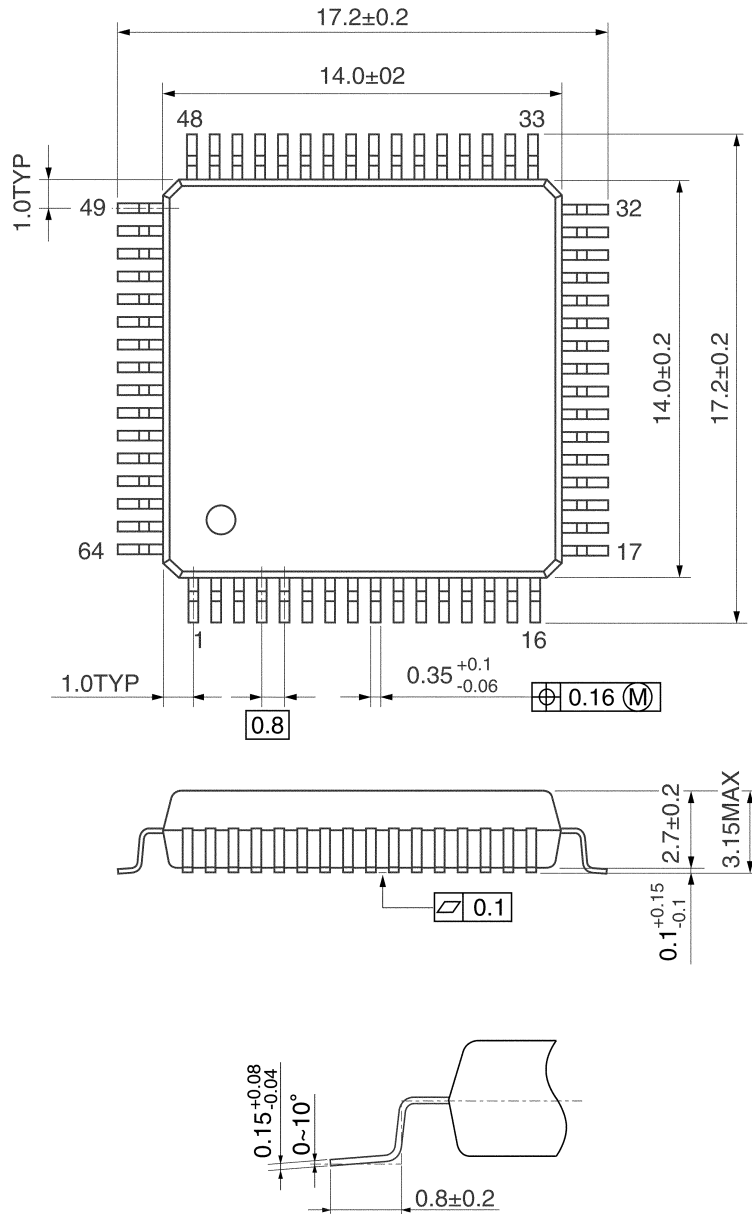
*3: These values apply only to products for which a 20-MHz input clock is available.

[13] Package Types and Dimensions

[13] Package Types and Dimensions

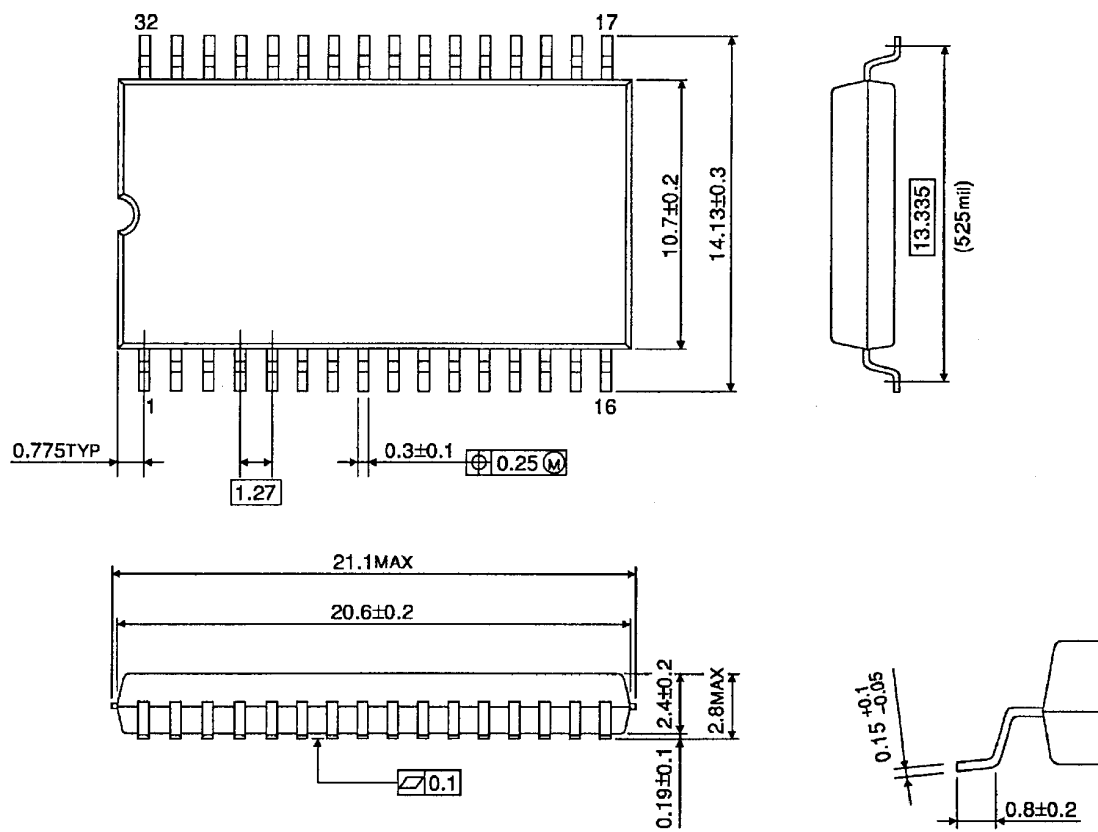
QFP64-P-1414-0.80C

Unit: mm



SOP32-P-525-1.27

Unit : mm



1. PCB Neuron Chip Pad Layout

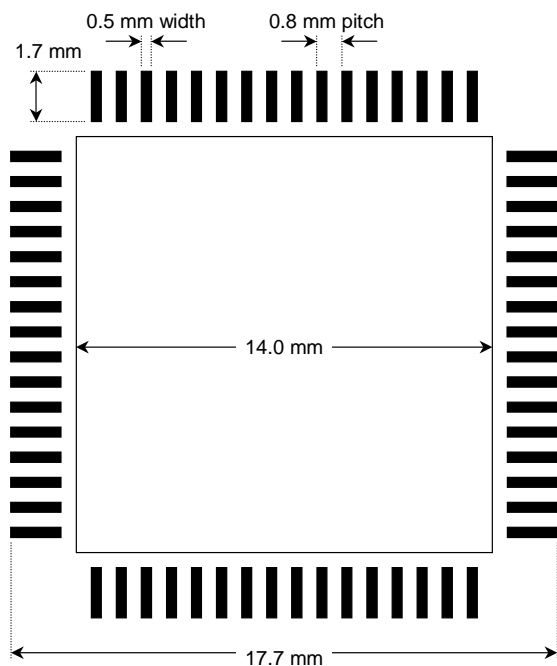


Figure 1.1 Recommended Pad Layout for QFP64-P-1414-0.80C

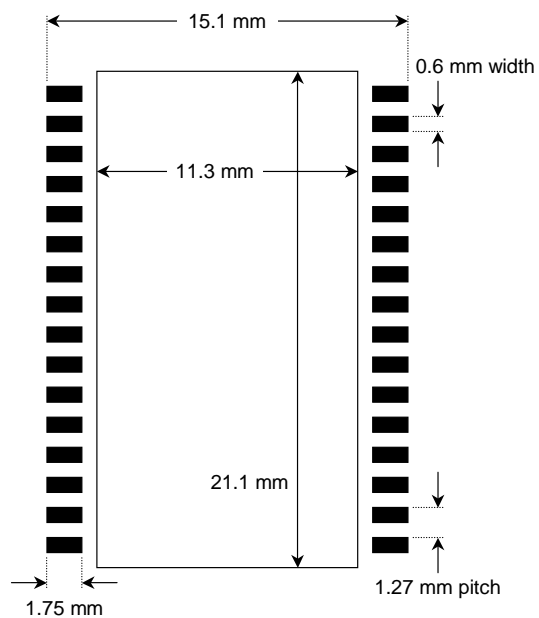


Figure 1.2 Recommended Pad Layout for SOP32-P-525-1.27

2. Socket Information

QFP64-P-1414-0.80C	Manufacture: Yamaichi	Part number: IC51-0644-692
	Manufacturer: Enplas	Part number: FPQ-64-0.8-10A
SOP32-P-525-1.27	Manufacture: Yamaichi	Part number: IC51-0322-667-2

[14] Electrical Characteristics

[14] Electrical Characteristics

1. Communications Port Differential Receiver Electrical Characteristics

Specifications are valid over the entire free-air operating temperature range, unless otherwise noted.

Table 1.1 Miscellaneous Characteristics

Item		TYPE-1	TYPE-2	Unit
Common mode range (with hysteresis)	min	$V_{SS} + 1.2$	$V_{SS} + 0.45$	V
	max	$V_{DD} - 2.2$	$V_{DD} + 0.55$ $V_{(COMM)} < 7.0$	
Common mode range (no hysteresis)	min	$V_{SS} + 0.9$	$V_{SS} - 0.1$	V
	max	$V_{DD} - 1.75$	$V_{DD} + 1.0$ $V_{(COMM)} < 7.0$	
Input offset voltage	min	$-0.05 \times V_h - 35$		mV
	max	$0.05 \times V_h + 35$		
Input resistance (at V_{SS} to V_{DD} level)	min	5 (Note1)	5 (Note2)	MΩ

Table 1.2 Absolute Maximum Ratings ($V_{SS} = 0$ V, V_{SS} reference)

Item	Min	Max	Unit
Maximum Input voltage Pin CPO to CP3 V_{IN} (type-1)	-0.3	$V_{DD} + 0.3$	V
Maximum Input voltage Pin CPO to CP3 V_{IN} (type-2)	$V_{SS} - 0.5$ V	$V_{DD} + 1.3$ V $V_{IN} < 7.3$	V

TYPE-1: TMPN3150B1AF, TMPN3120FE3MG

TYPE-2: TMPN3120FE5MG

Note1: This value applies to CP0-CP3 pins of TMPN3120FE3MG at power down.

Note2: This value applies to CP0-CP3 pins at power on and down. ($V_{SS} \leq V_{IN} \leq 7.0$ V).

Table 1.3 Hysteresis Values Expressed as Differential Peak-to-Peak Voltages in Terms of V_{DD} (V_h)

Hysteresis (H)	Min	Typ.	Max	Unit
0	$0.019 \times V_{DD}$	$0.027 \times V_{DD}$	$0.035 \times V_{DD}$	V
1	$0.040 \times V_{DD}$	$0.054 \times V_{DD}$	$0.068 \times V_{DD}$	V
2	$0.061 \times V_{DD}$	$0.081 \times V_{DD}$	$0.101 \times V_{DD}$	V
3	$0.081 \times V_{DD}$	$0.108 \times V_{DD}$	$0.135 \times V_{DD}$	V
4	$0.101 \times V_{DD}$	$0.135 \times V_{DD}$	$0.169 \times V_{DD}$	V
5	$0.121 \times V_{DD}$	$0.162 \times V_{DD}$	$0.203 \times V_{DD}$	V
6	$0.142 \times V_{DD}$	$0.189 \times V_{DD}$	$0.236 \times V_{DD}$	V
7	$0.162 \times V_{DD}$	$0.216 \times V_{DD}$	$0.270 \times V_{DD}$	V

Table 1.4 Receiver (Note3) (end to end) Filter Values Expressed as Transient Pulse Suppression Times

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

Table 1.5 Receiver (Note3) (end to end) Absolute Asymmetry (worst case across hysteresis)

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

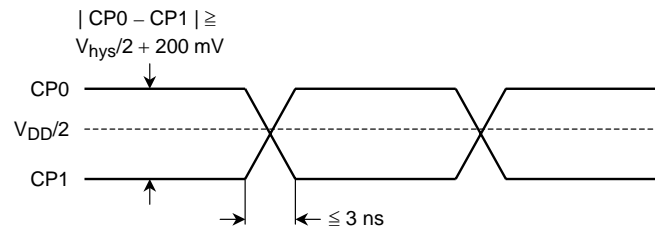


Figure 1.1

Note3: Receiver input, $V_D = V_{CP0} - V_{CP1}$, which must be at least 200 mV greater than hysteresis levels

See Figure 1.1.

Table 1.6 Receiver (Note4) (end-to-end) Absolute Asymmetry

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

Note4: ● CP0 and CP1 each an input 0.6 V_{PP} , 1.25 MHz sine wave 180° out of phase with each other, as shown in Figure 1.2.

● $V_{DD} = 5.00 \text{ V} \pm 5\%$

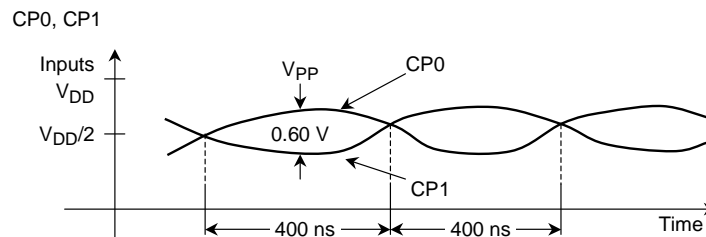


Figure 1.2

2. Memory Interface Timing Specifications

Memory interface timing parameters are defined in Figure 2.6. Figure 2.7 illustrates an example of the TMPN3150 connected to external memory. A [15:0], D [7:0], and R/~W are all measured with 50 pF loads; ~E is measured with a 20 pF load. Timing parameters are measured to TTL levels (see Figure 2.2).

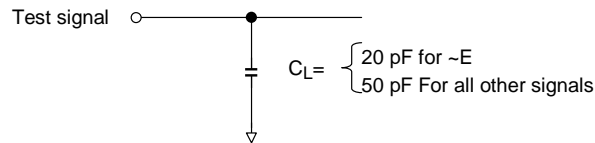


Figure 2.1 Signal Loading for Memory Interface Timing Specifications

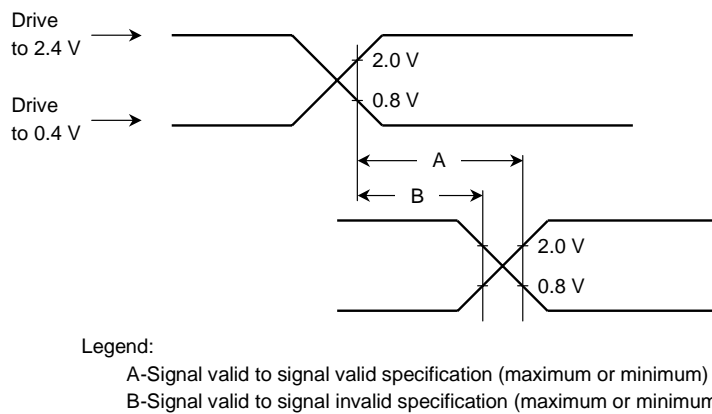


Figure 2.2 Drive Levels and Test Points for Memory Interface Timing Specifications

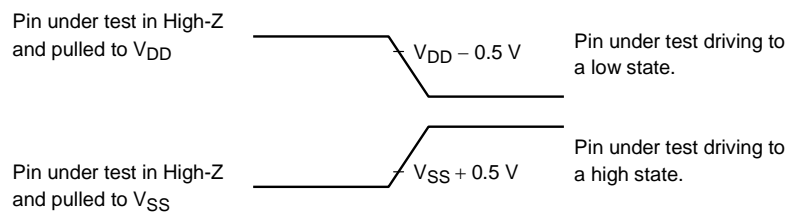


Figure 2.3 Test Point Levels for Three-State to Driven Time Measurements

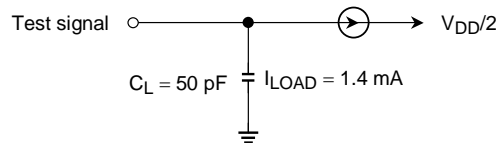


Figure 2.4 Signal Loading for Driven to Three-State Timing Measurements

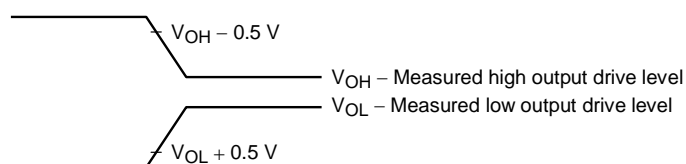


Figure 2.5 Test Point Levels for Driven to Three-State Time Measurements

Table 2.1 TMPN3150B1AFG Chip Memory Timing Variance by Voltage (4.5 to 5.5 V) and Temperature (–40 to + 85°C)

Parameter	Symbol	Min	Max	Unit
Memory cycle time (system clock period)	t_{CYC}	200	3200	ns
Pulse width $\sim E$ high	PW_{EH}	$t_{CYC}/2 - 5$	$t_{CYC}/2 + 5$	ns
Pulse width $\sim E$ low	PW_{EL}	$t_{CYC}/2 - 5$	$t_{CYC}/2 + 5$	ns
Delay, $\sim E$ high to address valid	t_{AD}	—	45	ns
Address hold time	t_{AH}	10	—	ns
Delay, $\sim E$ high to R/ $\sim W$ valid (read operation)	t_{RD}	—	45	ns
R/ $\sim W$ hold time (read operation)	t_{RH}	5	—	ns
Read data setup time	t_{DSR}	25	—	ns
Data hold time (read operation)	t_{DHR}	0	—	ns
Delay, $\sim E$ high to R/ $\sim W$ valid (write operation)	t_{WR}	—	45	ns
R/ $\sim W$ hold time (write operation)	t_{WH}	5	—	ns
Delay, $\sim E$ low to data valid	t_{DDW}	—	60	ns
Data hold time (write operation) (Note4)	t_{DHW}	9	40	ns

Note1: $t_{CYC} = 2 \times 1/F$, where F is the input clock frequency.

Table 2.2 TMPN3150B1AFG Chip Memory Timing Variance by Voltage (3.8 to 5.5 V) and Temperature (–40 to + 85°C)

Parameter	Symbol	Min	Max	Unit
Memory cycle time (system clock period)	t_{CYC}	400	3200	ns
Pulse width $\sim E$ high	PW_{EH}	$t_{CYC}/2 - 10$	$t_{CYC}/2 + 10$	ns
Pulse width $\sim E$ low	PW_{EL}	$t_{CYC}/2 - 10$	$t_{CYC}/2 + 10$	ns
Delay, $\sim E$ high to address valid	t_{AD}	—	50	ns
Address hold time	t_{AH}	10	—	ns
Delay, $\sim E$ high to R/ $\sim W$ valid (read operation)	t_{RD}	—	50	ns
R/ $\sim W$ hold time (read operation)	t_{RH}	5	—	ns
Read data setup time	t_{DSR}	30	—	ns
Data hold time (read operation)	t_{DHR}	0	—	ns
Delay, $\sim E$ high to R/ $\sim W$ valid (write operation)	t_{WR}	—	50	ns
R/ $\sim W$ hold time (write operation)	t_{WH}	5	—	ns
Delay, $\sim E$ low to data valid	t_{DDW}	—	65	ns
Data hold time (write operation) (Note4)	t_{DHW}	9	40	ns

Note1: $t_{CYC} = 2 \times 1/F$, where F is the input clock frequency.

Note2: LVD Reset occurs at $V_{DD} = 3.8 \text{ V}$ to 4.4 V .

Note3: This specification covers timing in the range between the normal minimum operating voltage of 4.5 V , and the minimum LVD trip point of 3.8 V .

Note4: Since the data bus goes high-impedance at $\sim E$ high, the effective data hold time is dependent on the current load on the bus. Typically this will be $\gg 50 \text{ ns}$.

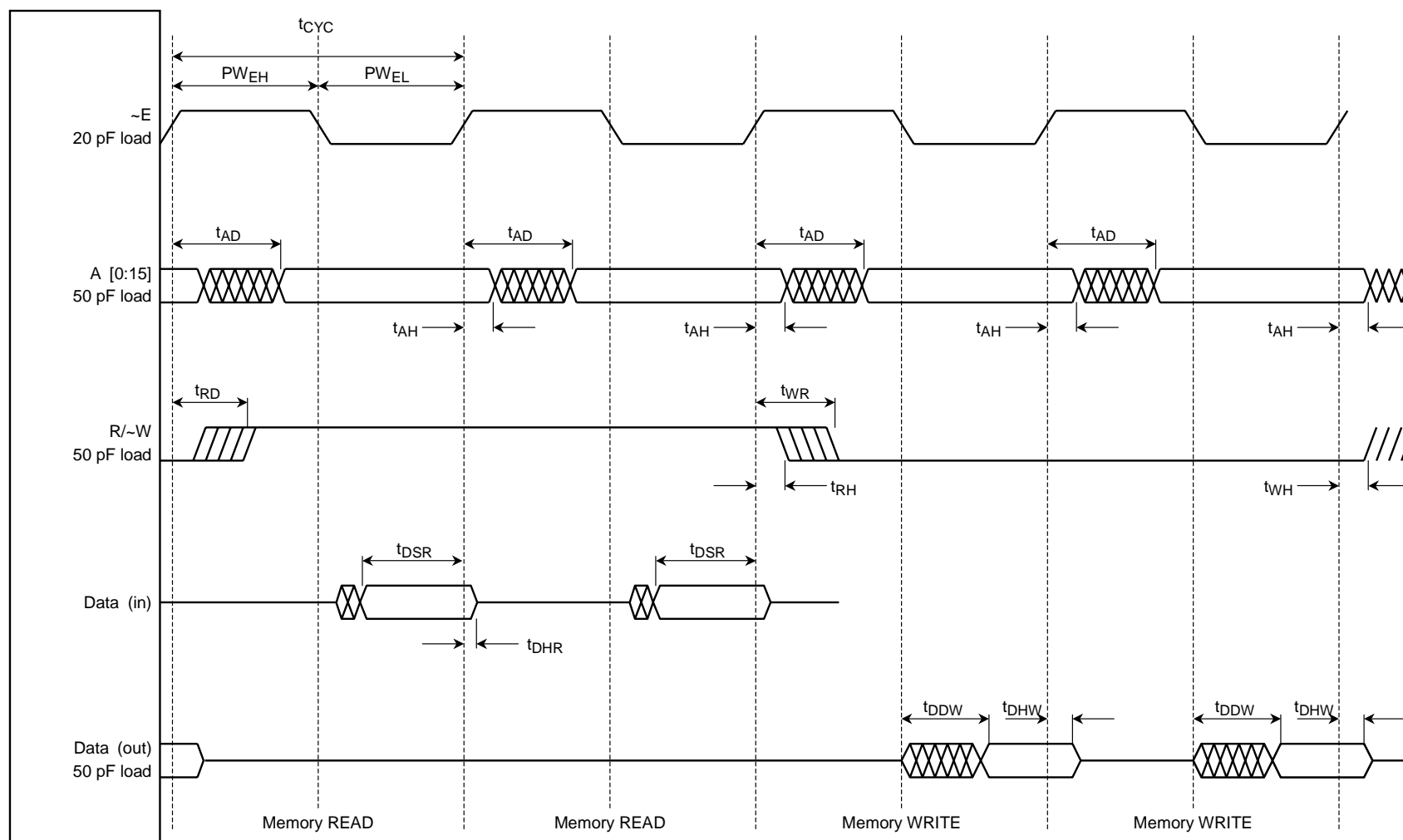


Figure 2.6 TMPN3150B1AF Chip Memory Interface Timing Diagram

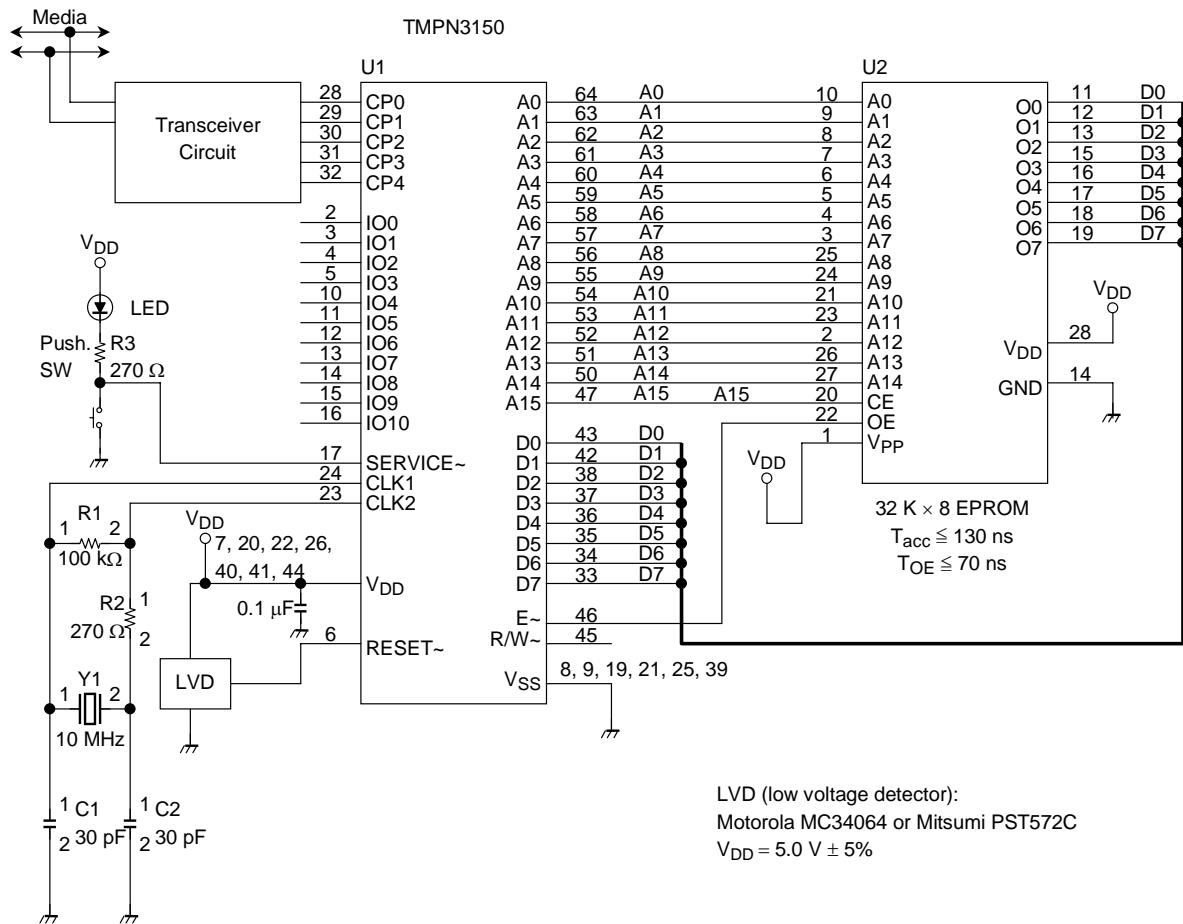


Figure 2.7 TMPN3150 Chip External Memory Interface with 32 Kbyte EPROM

Appendix A:

Appendix A:

Neuron Chip Data Structures

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

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

The System Image

This contains the Neuron Chip firmware that includes the LonTalk protocol, the Neuron C runtime library, and the task scheduler. In the TMPN3120×× Chip, this software is in the on-chip ROM. In the TMPN3150 Chip, this software is in an external ROM. For the TMPN3150 Chip, these items are provided as part of the Development tool. Using Development 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 TMPN3120 Chip, Some Neuron C runtime library functions and I/O objects are not in the on-chip ROM, but may be loaded into EEPROM along with the application program (please refer to section 6).

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 tool. They include:

- Network variable fixed and self-identification 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 TMPN3150 Chip, the application image is typically programmed into an external ROM, or downloaded over the network to EEPROM memory. In the TMPN3120 Chip, the application image is downloaded into the on-chip EEPROM memory. The LonBuilder and NodeBuilder tools support creation of application images.

The application image data structures described here are:

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

The Network Image

This contains the address assignments of the LONWORKS node, the binding information connecting network variables and message tags between the nodes in the network, parameters of the LonTalk protocol that may be set at installation time, and configuration variables of the application program. A network management tool typically downloads the network image over the network into on-chip EEPROM memory when a 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 this data using run-time library calls and declarations for the purpose of node self-installation. In the LonBuilder or NodeBuilder Neuron C development environment, function prototypes and declarations are found in the ACCESS.H and ADDRDEFS.H include files.

These data structures may also be accessed by network management messages received over the network from a network management tool (see appendix B). For network management tools running on PC hosts, the LNS products provides convenient high-level access to these data structures from a host-based application program. For network management tools running on any host, the LonManager NSS-10 Networks Services Server module provides similar service.

The network image data structures described here are:

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

On-Chip EEPROM Layout

For reference, this section defines the relative layout of the various data structures in on-chip EEPROM. Structures in on-chip EEPROM should only be accessed 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 programmer should access only these structures using the built-in access functions because:

1. The locations of these structures vary depending on the configuration of the application;

Structure	Location	
	TMPN3150 (firmware version 4)	TMPN3120FE3MG/FE5MG, TMPN3150 (firmware version 6 or later)
Fixed read-only data structure	0xF000	0xF000
Configuration data structure	0xF024	0xF029
Boot ID	0xF1FE (TMPN3150 only)	0xF1FE (neuron 3150 chip only)
Domain table	0xF03D	0xF042
Address table	0xF03D + 15 (decimal) bytes per domain	0xF042 + 15 (decimal) bytes per domain
Network variable configuration table	(start address of address table) + (15 bytes) × (number of address table)	(start address of address table) + (15 bytes) × (number of address table)

Refer to the Reset Processes and Timing section for more information on the Boot ID.

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.

The network variable fixed table, the SNVT descriptor table, and the various other parts of the application image are located by the Neuron C linker at addresses which depend on the memory map of the node. For a TMPN3150 Chip-based node, it is possible to specify that the off-chip ROM is just 16 Kbytes (64 pages), which will force the whole application image into internal EEPROM.

RAM Layout

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

Systems 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., TMPN3120 or TMPN3150. The available amounts of system base RAM for each of these chips are 454 and 636 bytes respectively. The portions of this memory allocated to the application data and return stacks are 114 and 232 bytes for the 3120 and 3150 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, otherwise there is only one.

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 byte count.

I/O Changes Array. Required space is equal to the number of I/O change events in the program, times 3 bytes per event.

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

The system base RAM, application timers, receive transactions, and transmit transactions must reside in on-chip RAM. The buffers and the 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 span the on-chip/off-chip boundary.**

A.1 Fixed Read-Only Data Structure

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

The following are declarations from ACCESS.H and ADDRDEFS.H:

```
#define NEURON_ID_LEN 6
#define ID_STR_LEN 8
typedef struct {
    unsigned neuron_id[ NEURON_ID_LEN ];           // offset 0x00
    unsigned model_num;                             // offset 0x06
    unsigned                                     : 4;
    unsigned minor_model_num                       : 4;           // offset 0x07
    const nv_fixed_struct *nv_fixed                :           // offset 0x08
    unsigned read_write_protect                    : 1;           // offset 0x0A
    unsigned                                     : 1;
    unsigned nv_count                              : 6;
    const snvt_struct * snvt;                      // offset 0x0B
    unsigned id_string[ ID_STR_LEN ];               // offset 0x0D
    unsigned NV_processing_off                     : 1;           // offset 0x15
    unsigned two_domains                          : 1;
    unsigned explicit_addr                        : 1;
    unsigned                                     : 5;
    unsigned address_count                        : 4;           // offset 0x16
    unsigned                                     : 4;
    unsigned                                     : 4;
    unsigned receive_trans_count                  : 4;           // offset 0x17
    unsigned app_buf_out_size                     : 4;           // offset 0x18
    unsigned app_buf_in_size                     : 4;
    unsigned net_buf_out_size                     : 4;           // offset 0x19
    unsigned net_buf_in_size                     : 4;
    unsigned net_buf_out_priority_count           : 4;           // offset 0x1A
    unsigned app_buf_out_priority_count           : 4;
    unsigned app_buf_out_count                    : 4;           // offset 0x1B
    unsigned app_buf_in_count                    : 4;
    unsigned net_buf_out_count                    : 4;           // offset 0x1C
    unsigned net_buf_in_count                    : 4;
    int reserved1[6];
    unsigned                                     : 6;           // offset 0x23
    unsigned tx_by_address                        : 1;
    unsigned idempotent_duplicate                 : 1;
} read_only_data_struct;
```

```
// The following addendum to the read-only data
// structures is available only in the 3150 firmware
// version 6 and later (3120 version 4)
typedef struct (
    unsigned                                :    2;
    unsigned  alias_count                  :    6;
    unsigned  msq_tag_count                :    4;
    unsigned                                :    4;
    int       reserved2[3];
} read_only_data_struct_2;

const read_only_data_struct read_only_data;
const read_only_data_struct_2 read_only_data_2;
```

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

A.1.1 Read-Only Structures 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: TMPN3150 Chip = 0, TMPN3120B Chip = 8, TMPN3120E1 Chip = 9, TMPN3120FE1 = 9, TMPN3120FE3 = 11 (or 0 × 0B), TMPN3120A20/A20U = 12 (or 0 × 0C), TMPN3120FE5M = 13 (or 0 × 0D)

```

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 if this node is a LONWORKS network interface using host network variable selection, then this pointer is not used.

```

unsigned  read_write_protect      : 1;                // offset 0x0A

```

This bit specifies that of the Neuron Chip memory may not be read or written over the network with the Read Memory and Write Memory network management messages (section B.1.5). The application program may set this bit with the Neuron C compiler directive `#pragma read_write_protect`. If this bit is set, only the Read-only Structure (section A.1), the SNVT Structures (section A.5), the Configuration Structures (section A.6), and the application data area may be read, and only the Configuration Structures 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 to 62). Each element of a network variable array is counted separately. If this node is a LONWORKS network interface using host network variable selection, 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 a LONWORKS network interface using host network variable selection, this pointer is 0xFFFF. The self-identification information may be suppressed by 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 these Neuron C compiler directives:

```
#pragma set_id_string "sssssss"
```

or

```
#pragma set_std_prog_id fm:mm:mm:cc:cc:ss:ss:nn
```

The second format is reserved for nodes that conform to the LONMARK interoperability guidelines. A program ID conforming to this format can also be generated using the program ID field in the NodeBuilder device definition window. The bit assignment for this format is as follows:

- The first 4 bits are the format code (0x8 to 0xF). Format 8 is reserved for devices passing the LONMARK conformance review, and format 9 should be used by devices with standard program IDs that have not yet passed the conformance review. Formats 0xA to 0xF are reserved for future use.
- The next 20 bits are the manufacturer code. Unique identifiers are assigned to manufacturers when they become members of the LONMARK program.
- The next 16 bits are the device class. It is drawn from a registry of pre-defined class definitions defined by the LONMARK program.
- The next 16 bits are the device subclass. It is drawn from a registry of pre-defined subclass definitions defined by the LONMARK program.
- 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 directive, by the NodeBuilder device definition window, 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 selection is being performed off-chip in a host-based node using a LONWORKS network interface.

```
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 transaction, 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_but_out_priority_count: 4;           // offset 0x1A
unsigned app_but_out_priority_count: 4;
unsigned app_but_out_count          : 4;           // offset 0x1B
unsigned app_but_in_count           : 4;
unsigned net_but_out_count          : 4;           // offset 0x1C
unsigned net_but_in_count           : 4;

```

These fields specify the number of application and network buffers, as defined by the corresponding Neuron C compiler directives (from 0 to 191 buffers). **Note that if one of the priority output buffer counts is zero, then both of them must be zero.** The fields are encoded as follows:

Field Value	Buffer Count
0	0
2	1
3	2
4	3
5	5
6	7
7	11
8	15
9	23
10	31
11	47
12	63
13	95
14	127
15	191

```

unsigned tx_by_addr          : 1;

```

This bit specifies that the node is maintaining a separate outgoing transaction space for each unique destination address in the address table.

```

unsigned idempotent_duplicate : 1;

```

This bit specifies that the Neuron Chip will set the idempotent retry bit in the application buffer when a request retry is sent up to the application.

```

unsigned alias_count         : 6 ;

```

The field specifies the number of entries in the network variable alias table. If this node is a LONWORKS network interface using host network variable selection, this field is zero.

The maximum value for this field is 62.

```

unsigned msg_tag_count       : 4;

```

This field species the number of bindable message tags used by the node. The allowable range is 0 to 15.

A.2 The Domain Table

This table defines the domains to which this node belongs. It is located in EEPROM, and is part of the network image written during node installation. During development, the contents of this table are downloaded when the node is loaded.

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( const 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-installed 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 by subnet/node addressing mode in that domain. If `update_clone_domain()` is used, the node can only receive messages addressed by 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, using 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 6 bytes, it is left-justified in this field.

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 from 1 to 255 for each domain.

unsigned node;

This field specifies the ID of the node within this subnet (1 to 127). When using the Lonbuilder tool, this ID is assigned automatically when the node specification object is created. Zero is an invalid node ID.

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. 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 in part of the network image written during node installation.

The following are declarations from ACCESS.H and ADDRDEFS.H:

```
typedef enum { UNBOUND, SUBNET_NODE, NEURON_ID, BROADCAST } addr_type;
typedef union {
    group_struct      gp;
    snode_struct      sn;
    bcast_struct      bc;
    turnaround_struct ta;
} address_struct;

const address_struct * access_address( int index );
update_address( const address_struct * address_entry, int index );
int addr_table_index( message_tag_name );
```

The application program may read and write any entry in this table using the access routines `access_address()` and `update_address()`. The index of the address table entry corresponding to any message tag declared in the program may be determined with the function `addr_table_index()`. The address table consists of up to 15 entries, each 5 bytes in length. The default number of entries is 15, which may be overridden by the Neuron C compiler directive `#pragma num_addr_table_entries nn`.

Each entry may be in one of five formats: group address, subnet/node address, broadcast address, turnaround address, or not in use. A group address is used for multicast addressing, when a network variable or message tag is used in a connection having more than two members. A subnet/node address is used for unicast addressing, when an output network variable or message tag is used in a connection with one other node. Broadcast addressing is not used by the LNS binders when they create network variable or message tag connections. However, broadcast addressing is supported in the protocol, and may be used with explicit addressing. A turnaround address is used for network variables that are only bound to other network variables in the same node, and not to any network variables on other nodes. Destination addresses in the unique 48-bit Neuron 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`.

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 to 255	group format

Any bindable message tags declared in the application program are assigned to the first entries in the address table in order of declaration. This is followed by address table entries used for network variables—in the development environment, the binder assigns these entries.

The repeat timer, retry count, receive timer, and transaction timer are common to several of these address formats, and are described below in Section A.3.11.

A.3.1 Declaration of Group Address Format

This is done as follows:

```
typedef struct {
    unsigned    type        : 1;           // offset 0x00
    unsigned    size        : 7;
    unsigned    domain      : 1;           // offset 0x01
    unsigned    member      : 7;
    unsigned    rpt_timer   : 4;           // offset 0x02
    unsigned    retry       : 4;
    unsigned    rcv_timer   : 4;           // offset 0x03
    unsigned    tx_timer    : 4;
    unsigned    group       : 8;           // offset 0x04
} group_struct;
```


A.3.2 Group Address Field Descriptions

unsigned type : 1;

This bit is 1 for a group address, 0 for any of the other formats.

unsigned size : 7;

This field specifies the size of the group (2-64). The size of a group includes the sender of the message. If this field is 0, then the group is of unlimited size, and Unacknowledged or Repeated service must be used.

unsigned domain : 1;

The value in this field (0 or 1) specifies domain table 0 or 1.

unsigned member : 7;

This field specifies the member ID of this node within its group (0 to 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 the group within its domain. A group ID may be in the range from 0 to 255. In the LonBuilder environment, the group ID is typically allocated by a network variable binder.

A.3.3 Declaration of Subnet/Node Address Format

This declaration is made as follows:

```
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     : 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;

The value in this field (0 or 1) specifies domain table 0 or 1.

unsigned node : 7;

This field specifies the node ID (1 to 127) within the specified subnet and domain. Zero is not a valid node ID.

unsigned subnet : 8;

This field specifies the subnet ID (1 to 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;
    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;

The value in this field (0 or 1) specifies domain table 0 or 1.

unsigned backlog : 6;

This field specifies an estimate of the channel backlog that would be created by an acknowledged message or a request/response message broadcast using this address. It should be set to the expected number of acknowledgements or response. For example, this might be the worst-case number of nodes expected to respond to a *Query ID* message on a channel. If that 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 to 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 acknowledged is received. Subsequent responses or acknowledgments are discarded.

A.3.7 Declaration of Turnaround Address Format

This declaration is made as follows:

```
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;
```

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

```
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
} nnnid_struct;
```

This field specifies the unique 48-bit Neuron ID of the destination Neuron Chip.

A.3.11 Timer Field Descriptions

unsigned rpt_timer : 4;

When Repeated service is used, this field specifies the time interval between repetitions of an outgoing message. The encoding of this field is specified in Table A.1.

unsigned retry : 4;

This field specifies the maximum number of retries for Acknowledged, Request/Response, or Repeated service (0 to 15). The maximum number of messages sent is one more than this number.

unsigned rcv_timer : 4;

When the node receives a multicast (group) message, the receive timer is set to the time interval specified by this field. If a message with the same transaction ID is received before the receive timer expires, it is considered to be a retry of the previous message. The encoding of this field is specified in Table A.1.

unsigned tx_timer : 4;

When Acknowledged or Request/Response service is used, this field specifies the time interval between retries. The transaction retry timer is restarted when each attempt is made, and also when any acknowledgment or response (except for the last one) is received. For Request/Response service, when setting the transaction timer the requesting node should take into account the delay necessary for the application to respond. This is important, for example, for network management messages that write into EEPROM. The encoding of the transaction timer field is specified in Table A.1.

See Section A.6.1 for a description of the non_group_timer field defined in Table A.1.

Table A.1 Encoding of Timer Field Values in msec.

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 three tables associated with network variables: the network variable configuration table, the network variable alias table, and the network variable fixed table. The network variable configuration table defines the configurable attributes of the network variables in the node. It is located in EEPROM so that it can be modified during node installation, and is part of the network image written during node installation. The network variable alias table defines the configurable attributes of the alias network variables in the node, and is located in EEPROM, immediately following the network variable configuration table. The network variable fixed table defines the compile and link-time attributes of the network variables in the same node. It may be located in read-only memory, and is part of the application image written during application download.

For a node using a LONWORKS network interface, the network variable fixed table and the network variables themselves are located in the memory of the host microprocessor. The network variable configuration and alias tables may be in either the memory of the Neuron Chip, or the memory of the host microprocessor. In the latter case, the on-chip limits of 62 bound network variables and 62 alias network variables per node both increase to 4096, and the LonTalk protocol firmware on the Neuron Chip does not process network variable update messages, but instead passes them to the host microprocessor.

Declarations from ACCESS.H

```
#define MAX_NV 62
```

```
typedef struct {
```

```
    unsigned    nv_priority           : 1;           // offset 0x00
    unsigned    nv_direction          : 1;
    unsigned    nv_selector_hi        : 6;
    unsigned    nv_selector_lo        : 8;           // offset 0x01
    unsigned    nv_turnaround          : 1;           // offset 0x02
    unsigned    nv_service             : 2;
    unsigned    nv_auth               : 1;
    unsigned    nv_addr_index         : 4;
```

```
} nv_struct;
```

```
typedef struct {
```

```
    nv_struct    nv_cnfg;                // offset 0x00
    unsigned     primary;                 // offset 0x03
    unsigned     long host_primary;       // offset 0x04
```

```
} alias_struct;
```

```
const nv_struct * access_nv( int index);
```

```
void update_nv(const nv_struct * nv_entry, int index );
```

```
const alias_struct * access_alias (int index);
```

```
void update_alias ( const alias_struct * alias_entry, int index );
```

```
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 from and write to any entry in the network variable configuration and alias tables, using the access routines `access_nv()`, `update_nv()`, `access_alias()`, and `update_alias()`.

The index of the network variable configuration table entry corresponding to any network variable declared in the program may be determined with the function `nv_table_index()`. For the alias table functions, the index is that of the alias entry. The base address of the network variable fixed table may be retrieved from the read-only data field `read_only_data.nv_fixed` (for non-host-based nodes only).

For the network variable configuration and fixed tables on a Neuron Chip-hosted node, each of the tables consists of up to 62 entries, each 3 bytes in length. The number of entries is determined by the number of network variables declared in the application program—each element of a network variable array counts separately. The index written 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 the *Query/Update Net Variable Configuration* network management messages. The network variable fixed table may not be written, except when downloading the application image.

The network variable alias table can have up to 62 entries, each either 4 bytes (neuron chip or host-based node) or 6 bytes (host-based node only) in length. The actual number of entries for a Neuron Chip-hosted node is set through the use of the compiler directive, `#pragma num_alias_table_entries`.

A.4.1 Network Variable Configuration Table Field Descriptions

unsigned nv_priority : 1;

This bit is set to 1 if the network variable uses priority massaging. The choice of states is specified by bind_info (priority | nonpriority) in the Neuron C declaration of the network variable.

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 by any *Update Net Variable Configuration* network management message. This bit is technically not configuration data; it resides in this table only 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 from 0 to 0x3FFF. Selector values 0x3000 to 0x3FFF are reserved for unbound network variables, with the selector value equal to 0x3FFF minus the network variable index. Selector values 0 to 0x2FFF are available for bound network variables. The input network variables on any one node must all have different selectors. The output network variables on any one node also must all have different selectors. The LNS binders ensure this by assigning the same network variable selector to the union of all the connection sets in which a network variable participates.

unsigned nv_turnaround : 1;

This bit is set to one if this is a turnaround network variable, that is, if it 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. The service type is specified by bind_info (ackd | unackd_rpt | unackd) in the Neuron C declaration of the network variable.

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. This choice is specified by bind_info (authenticated | nonauthenticated) in the Neuron C declaration of the network variable.

unsigned nv_addr_index : 4;

The value in this field (0 to 14) specifies an address table from 0 to 14. The value 15 (0x0F) is used if the network variable is not associated with an address table entry. Multiple network variables may use the same address table index.

A.4.2 Network Variable Alias Table Field Description

`nv_struct nv_cnfg;`

This has the same definition as in the network variable configuration table above.

`unsigned primary;`

This is the index of the network variable configuration table. For host-based nodes, a value of 0xFF indicates that the following two bytes should be used for the index instead.

`unsigned long host_primary;`

Network variable configuration table index. This field is present only if the `primary` field is set to 0xFF. This is valid for host-based nodes. It is only necessary to use these bytes if the network variable index used is larger than 254.

A.4.3 Network Variable Fixed Table Field Descriptions

`unsigned nv_sync : 1;`

This bit is set to one if this is a synchronous network variable. The bit is specified with the modifier `sync` in the Neuron C declaration of the network variable.

`unsigned nv_length : 5;`

This field specifies the number of bytes in the network variable (1 to 31).

`Void * nv_address;`

This field is a pointer to the location of the network variable's data in RAM or EEPROM.

A.5 The Standard Network Variable Type (SNVT) Structures

These are five 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
- self-identification data for binding and status information

This information forms part of the application image written during node manufacture. If the Neuron C compiler directive `#pragma disable_snvt_si` is specified in the application program, none of this information is present. In a TMPN3150 Chip, 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
    union {
        struct {
            unsigned msb_num_netvars;       // offset 0x04
            unsigned mtag_count;            // offset 0x05
        } ver1;
        struct {
            unsigned mtag_count;            // offset 0x04
        } ver0;
    } variable_part;
} snvt_struct;

typedef struct {
    unsigned    ext_rec          : 1;       // offset 0x00
    unsigned    nv_sync          : 1;
    unsigned    nv_polled        : 1;
    unsigned    nv_offline       : 1;
    unsigned    nv_service_type_config : 1;
    unsigned    nv_priority_config : 1;
    unsigned    nv_auth_config    : 1;
    unsigned    nv_config_class   : 1;
    unsigned    snvt_type_index;   // offset 0x01
} SNVT_desc_struct;

typedef struct {
    unsigned    binding_ll      : 1;
    unsigned    query_stats     : 1;
    unsigned    alias_count     : 6;
    unsigned long    host_alias; // Host node only
} alias_field
```

For a node running a Neuron Chip-based application program, the base address of the SNVT structure may be retrieved from the `read_only_data.snvt` read-only data field. For a node using a LONWORKS network interface, the SNVT Structures are located in the memory of the host processor and may be read with the *Query SNVT* network management message (the maximum size of the four SNVT structures is 65,535 bytes). The SNVT header structure is five bytes long. The SNVT descriptor table follows immediately afterwards, and it has one entry for every network variable declared in the application program. Version 0 format has a separate entry for each element of a network variable array. Version 1 format allows different elements of a network variable array to share a single 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. For Neuron Chip-based nodes, the self-identification and self-documentation information may be read over the network with the Read Memory network management command, using `address_mode=0`.

Following the extension records for the network variables are the self-identification bits which reflect the state of the various binding and query status conditions.

A.5.1 SNVT Structure Field Descriptions

unsigned long length;

This field specifies the total number of bytes in the self-identification and self-documentation data structures described here.

unsigned num_net_vars;

This field specifies the number of network variables declared in this node. Each element of a network variable array counts separately. In version 1 format, this byte is the least significant byte of the number of network variables declared in this node, and in this node an NV array counts as only one item.

unsigned version;

This field specifies the format of the SNVT structures to follow. Versions 0 and 1 are defined.

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. An NV array counts as only item 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 form the first entries in the address table (see section A.3).

```
unsigned    ext_rec           : 1;
```

```
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;
```

A-34

A.5.3 SNVT Table Extension Records

Extension records may be present for any of the network variables. If an extension record is present, the field `ext_rec` is set to 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 from 0 to 127, representing the rate in the range from 0 to 1878.0 which is specified by `bind_info(max_rate_est(nnn))` in the Neuron C declaration of the network variable. This 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 from 0 to 127, representing the rate in the range from 0 to 1878.0 which is 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 22 bytes if it is a network variable array element. Each array element has its own extension record containing the name of the array, a left square bracket, one, two, or three decimal digits denoting the index of the element, and a right square bracket.

unsigned `sd` : 1;

If this bit is set to 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.5.4 SNVT Alias Field Descriptions

unsigned binding_ll :1;

This bit is one if the node is using new binding constraints. Nodes supporting these binding constraints do not require that a unique selector be assigned to output network variables in non-polling connections. For example, in a node supporting these constraints, two different output network variables could be connected to the same input network variable in any type of node, as long as that input network variable does not initiate polls.

Note that polling of such output network variables by a network management of monitor node would have to be done using Query Network Variable network management messages rather than network variable poll messages.

unsigned query_stats :1;

This bit is one if the Query Statistics Addressing Mode of the *read memory* network management command can be used to extract the extended statistics information.

unsigned alias_count :6;

This field specifies the number of alias network variables used by the node (0 to 62). A value of 63 indicates that the next two bytes contain the actual number of alias network variables.

unsigned long host_alias;

This field specifies the number of alias network variables used by the host node (0 to 4095). This field is only present if *alias_count* is 63.

A.6 The Configuration Structure

This structure, located in EEPROM, defines the hardware and transceiver properties of this node. Parts of it belong to the application image written during node manufacture, and other parts are associated with 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      preamble_length;       // offset 0x0A
    unsigned      packed_cycle;         // offset 0x0B
    unsigned      beta2_control;        // offset 0x0C
    unsigned      xmit_interpacket;     // offset 0x0D
    unsigned      recv_interpacket;     // offset 0x0E
    unsigned      node_priority;        // offset 0x0F
    unsigned      channel_priorities;   // offset 0x10
    union {
        unsigned  xcvr_params[ NUM_COMM_PARAMS ];
        direct_param_struct  dir_params;    // offset 0x11
    } params
    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      cd_to_end_packet   : 6;   // offset 0x12
    unsigned      cd_tail            : 1;
    unsigned      cd_preamble        : 1;
} direct_param_struct;

const config_data_struct config_data;

void update_config_data( const config_data_struct * config_data );
```

The application program may read this structure using the global declaration `config_data`, and may write it using the function `update_config_data()`. The structure is 25 bytes long, and it may be read and written over the network using the *Read Memory and Write Memory* network management messages with `address_mode=2`. The 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 so that the changes can take effect.

A.6.1 Configuration Structure Field Descriptions

unsigned long `channel_id`;

This field specifies the ID of the channel to which this node is assigned. In the LonBuilder environment, this is assigned automatically when the channel object is created. The NodeBuilder tool uses a fixed-channel ID. The Neuron Chip firmware does not reference this field, but it is available to assist in recovering the network topology by interrogating the nodes.

char `location[LOCATION_LEN]`;

The location field is used to pass a 6-byte ASCII string which describes the physical location of the node, to the network management tool.

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 transceiver, it specifies the rate of the bit clock transmitted from the Neuron Chip to the transceiver. Table A.2 shows the transceiver bit rate as a function of the *input_clock* field and the *comm_clock* field.

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

comm_clock	ratio	6 (*1)	5	4	3	2	1
		20 MHz	10 MHz	5 MHz	2.5 MHz	1.25 MHz	625 kHz
0	8:1	2,500	1,250	625	312.5	156.3	78.1
1	16:1	1,250	625	312.5	156.3	78.1	39.1
2	32:1	625	312.5	156.3	78.1	39.1	19.5
3	64:1	312.5	156.3	78.1	39.1	19.5	9.8
4	128:1	156.3	78.1	39.1	19.5	9.8	4.9
5	256:1	78.1	39.1	19.5	9.8	4.9	2.4
6	512:1	39.1	19.5	9.8	4.9	2.4	1.2
7	1,024:1	19.5	9.8	4.9	2.4	1.2	0.6

unsigned input_clock : 3;

This field specifies the Neuron Chip input clock (oscillator frequency). The encoding is as follows:

6 (*2)	20 MHz
5	10.0 MHz
4	5.0 MHz
3	2.5 MHz
2	1.25 MHz
1	625 KHz
0	not used

*1: Only a product supporting an input clock of 20 MHz

*2: This value apply in Single-End mode only to products for which a 20 MHz input clock is available.

unsigned comm_type : 3;

This field specifies the type of transceiver. The encoding is as follows:

0	Blank Neuron Chip
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:

0x00	Blank Neuron Chip
0x0E	Direct mode-single-ended
0x0C	Direct mode-differential
0x1E	Special-purpose-wake-up pin is an output
0x16	Special-purpose-wake-up pin is an input

The following five fields specify the raw transceiver parameters used by the Media Access Control processor. 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 formulas are given in terms of 'x', the value in the control field, which may be 0 to 255 unless otherwise specified. A Neutron Chip CPU cycle is 0.6 μ s at 10 MHz input clock frequency, 1.2 μ s at 5 MHz and so on. See Figure 6.3 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 = $\{(209 \text{ to } 223) + 32 x\}$ cycles ($x = 1 \text{ to } 253$). For Special-Purpose mode transceivers, the value should be zero.
- `packet_cycle` This field determines the packet cycle duration used 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 beta 2 slot width.
Time = $(40 + 20 x)$ cycles.
- `xmit_interpacket` This field determines the interpacket padding after transmitting.
Time = $(41 x)$ cycles for $x < 128$,
 $\{145 (x - 128)\}$ cycles for $x \geq 128$.
- `recv_interpacket` This field determines the interpacket padding after receiving.
Time = $(41 x)$ cycles for $x < 128$,
 $\{145 (x - 128)\}$ cycles 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 to 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.
- `unsigned channel_priorities;` This field specifies the number of priority slots on the channel (0 to 255). The slots are numbered starting at one.

```
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. The most significant bit of the first transceiver parameter is defined to be the alternate channel bit. The alternate channel is used for the last two transmission attempts when using Acknowledged, Request/Response, or Repeated service. 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 or broadcast (non-group) message requiring a response or acknowledgment, the receive timer is set to the time interval specified by this field. If a message with the same transaction ID, and priority, as well as the same source and destination addresses is received before the receive timer expires, it is considered to be a retry of the previous message. The encoding of this field is shown in Table A.1. When network management messages using Neuron ID addressing are received, a receive time of 8.192 seconds is used instead of the value of this field.

```
unsigned  nm_auth                  : 1;
```

This field specifies that network management messages are to be authenticated. Setting this bit prevents the node from being configured by an unauthorized network management tool. However, network management messages received when the node is unconfigured 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.

```
unsigned  preemption_timeout
```

This field specifies the maximum time the node will wait in Preemption mode for a free buffer. A Preemption mode timeout logs an error and resets the Neuron Chip. The encoding is as follows:

0	forever
1	2 s
2	4 s
3	6 s
4	8 s
5	10 s
6	12 s
7	14 s

A.6.2 Direct Mode Transceiver Parameters Field Descriptions

For direct (single-ended or differential) mode transceivers, these parameters are used to control the operation of the transceiver port. In the LonBuilder environment, these parameters are specified in the channel screen.

unsigned collision_detect : 1;

This field specifies the Neuron Chip will monitor 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 to 3). See the electrical specifications of the Neuron Chip (Table 12.6) for details of the available glitch filter settings.

unsigned hysteresis : 3;

For Differential mode transceivers, this field specifies the setting of the receive hysteresis filter (0 to 7). See the electrical specifications of the Neuron Chip (Table 12.5) for details of the available hysteresis filter settings.

unsigned cd_to_end_packet : 6;

This field controls how close to the end of a packet the collision detect signal is checked in a transmitting Neuron Chip. It is set as a function of the bit rate and the input clock.

unsigned cd_tail : 1;

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

Appendix B:

Appendix B:

Network Management and Diagnostic Services

In addition to application message service, the LonTalk protocol provides network management services for installation and configuration of nodes, downloading of software, and diagnosis of the network. These messages are based on using it from network management tools, such as LonMaker. Therefore, about these messages, it becomes the outside for support. Message codes used by the LonTalk protocol are as follows:

Message Type	Hexadecimal Message Codes
Application messages	0x00 to 0x3E
Foreign Messages	0x40 to 0x4E
Network Diagnostics Messages	0x50 to 0x5F
Network Management Messages	0x60 to 0x7D
Router Configuration Messages	0x74 to 0x7E
Service Pin Message	0x7F
Network Variable Messages	0x80 to 0xFF

Response codes used by the LonTalk protocol are as follows:

Response Type	Hexadecimal Message Codes
Application Responses	0x00 to 0x3E
Response if node is off-line	0x3F
Foreign Responses	0x40 to 0x4E
Response if node is off-line	0x4F
Network Diagnostic Success	0x31 to 0x3F
Network Diagnostic Failure	0x11 to 0x1F
Network Management Success	0x21 to 0x3D
Network Management Failure	0x01 to 0x1D
Router Configuration Success	0x34 to 0x3E
Router Configuration Failure	0x14 to 0x1E
Network Variable Poll Response	0x80 to 0xFF

Notice that response codes are not unique. They must be chosen according to the original request.

Section 3 describes the use of application messages. A foreign message is a packet of another protocol embedded in the LonTalk protocol packet. The application programs designates message and response codes as being either application or foreign.

Table B.1 Network Diagnostic Messages

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

Table B.2 Network Management Messages

Network Management Messages	Request Code	Success Response	Failed Response
Query ID	0x61	0x21	0x01
Respond to Query	0x62	0x22	0x02
Update Domain	0x63	0x23	0x03
Leave Domain	0x64	0x24	0x04
Update Key	0x65	0x25	0x05
Update Address	0x66	0x26	0x06
Query Address	0x67	0x27	0x07
Query Net Variable Config	0x68	0x28	0x08
Update Group Address Data	0x69	0x29	0x09
Query Domain	0x6A	0x2A	0x0A
Update Net Variable Config	0x6B	0x2B	0x0B
Set Node Mode	0x6C	0x2C	0x0C
Read Memory	0x6D	0x2D	0x0D
Write Memory	0x6E	0x2E	0x0E
Checksum Recalculate	0x6F	0x2F	0x0F
Wink	0x70	0x30 (*)	0x10 (*)
Memory Refresh	0x71	0x31	0x11
Query SNVT	0x72	0x32	0x12
Network Variable Fetch	0x73	0x33	0x13
Device Escape Code	0x7D	0x3D	0x1D

*: Only for request/response messaging service

Router messages are used by network management tools to configure nodes that run the special router system image. They are not useful for application nodes, which will return a failed response.

Network variable messages cannot be received by an application program using explicit messaging syntax. They are sent by updating output network variables in the application program, and are implicitly received by input network variables in the same connection. For a discussion of network variable messages, see Section B.3.

Network management and network diagnostic messages may be delivered using Request/Response service (except for *mode on-line*, *mode off-line* and *wink*, which are delivered to the application processor and do not have response data associated with them). Network management messages that do not have response data associated with them may also be delivered with the other classes of service, namely Acknowledged, Unacknowledged and 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 (section. B.1.5) to retrieve the 6-byte Neuron ID from another Neuron Chip at offset 0x0000, into the Read-Only Structure:

```
#include <ADDRDEFS.H>
#include <ACCESS.H>
#include <NETMGMT.H>
NM_read_memory_request 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, NEURON_ID_LEN );
    // copy the response data to a local variable
}
```

The failed response is returned whenever the destination Neuron Chip cannot process the message. For example, this may happen when a table index is out of range, when there is a memory failure while writing to EEPROM, or when 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. A few are limited to acknowledged service. Throughout the messages descriptions that follows, 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 which do not require authentication in order 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 which 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 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 into a domain of which the node is not a member (either because the node is unconfigured or because it is simply not in the domain) are termed as being received on a flexible domain. Some commands are not permitted under these circumstances, and they are noted below.

One advantage of using Neuron ID addressing for network management commands is that if a node were to accidentally go unconfigured (eg. due to a checksum error resulting from a power cycle which changing configuration), the network management tool 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 should be used for network management activity. Neuron ID addressing should only be used for communicating with nodes which are not in the configured state.

Configuration Changes

The paradigm for making configuration changes is as follows:

1. After the node state or condition (optional).
2. Perform the change or changes. Most messages automatically update the configuration checksum.
The only 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 OK 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 the node state is corrupted (e.g. due to a power cycle which the state is being changed), the node will come up applicationless. If the network management tool 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.

B.1 Network Management Messages

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

B.1.1 Node Identification Messages

Query ID (request/response only)

This message requests selected nodes to respond with a message containing their 48-bit Neuron ID and program ID. This message is normally broadcast during network installation, to find specific nodes in the domain. It can be used to find unconfigured nodes or explicitly selected nodes, or nodes with specified memory contents at a specified address. This address may be specified absolutely, relative to the Read-only Structure (section A.1), or relative to the Configuration Structure (section A.6). Only data within the Read-only Structure, the SNVT Structures (section A.5) or the Configuration Structure, may be matched. For example, this can be used 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 from NETMGMT.H:

```
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, and if so, they specify an address mode, a byte count and an array of bytes which must match a part of the destination node's memory. For interoperability, the length of the matched region can be up to 11 bytes long. The matched region may be in the read-only structure, the configuration structure, the SNVT structure or the application RAM data variable area. The address mode specifies whether the address of the matched memory is an absolute address in the memory space of the Neuron Chip, an address relative to the read-only data structure (see section A.1), or an address relative to the configuration data structure (see section A.6). If the memory matching feature is not required, the optional fields must not be present in the message; the length of the data part of the message must be 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 from `NETMGMT.H` is as follows:

```
typedef enum {  
    MODE_OFF      = 0,  
    MODE_ON       = 1,  
} NM_respond_to_query_request;
```

The request message consists of a single byte specifying whether the node should be selected or deselected. Authentication is never used with this message.

Service Pin Message (unsolicited)

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

Message declaration from `NETMGMT.H` is as follows:

```
typedef struct {  
    unsigned neuron_id[ NEURON_ID_LEN ];  
    unsigned id_string[ ID_STR_LEN ];  
} NM_service_pin_msg;
```

The message data contains the 6-byte unique Neuron ID assigned by the manufacturer of the Neuron Chip, followed by the 8-byte ID of the application program in the Neuron Chip. See Section A.1.1 for details of these values. The service pin message is sent as a domain-wide broadcast on the domain, whose length is zero. The source subnet and node IDs are both zero. Routers retransmit service pin messages on the domains in which they are configured.

B.1.2 Domain Table Messages

Update Domain

This message overwrites a domain table entry with a new value, and recomputes the configuration checksum. This assigns a domain, subnet, and node identifier to the node, as well as an authentication key for that domain. The authentication key is transmitted in the clear, so that this message should not be used on an open network if authentication protection is desired. The node does not enter the configured state until a Set Node Mode message is sent to change its state to configured. The Update Domain message is honored even if the node is read/write protected. For a description of the domain table, see Section A.2. If the Update Domain message is received by a node on the domain which is being updated, the response is returned to the new domain.

A node that receives this message can take up to 330 ms to execute the function.

Message declaration from `NETMGMT.H` is as follows:

```
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_KYE_LEN ];
} NM_update_domain_request;
```

The request message consists of an index of 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 the `ACCESS.H` include file. The most significant bit of the byte containing the node ID must be set for normal operation. If this bit is clear, the node is treated as a “cloned” node. That is, the node will not be able to receive messages addressed to it using subnet/node address format. It will, however, be able to receive messages from another node in that domain that has the same subnet and node IDs.

Query Domain (request/response only)

This message retrieves an entry in the domain table. This message is honored even if the node is read/write protected. For a description of the domain table, see Section A.2.

Message declarations from `NETMGMT.H` are as follows:

```
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 of 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 the `ACCESS.H` include file.

Leave Domain

This message deletes a domain table entry, and recomputes the configuration checksum. After the message is processed, if the node does not belong to any domain, it becomes unconfigured and is reset. This message is honored even if the node is read/write protected. For a description of the domain table, see Section A.2. If the Leave Domain message is received by a node on the domain which is to be left, no response is returned. If the message causes the node to leave the last domain where it has been configured, its state becomes unconfigured. A node that receives this message can take up to 330 ms to execute the function

Message declaration from `NETMGMT.H`:

```
typedef unsigned /* domain_index */ NM_leave_domain_request;
```

The request message consists of an index into the domain table (0 or 1).

Update Key

This message adds an increment to the current encryption key in a domain table entry, to form a new key, and recomputes the configuration checksum. This allows re-keying of a node without having to transmit the new key in the clear on the network. This message is honored even if the node is read/write protected. For a description of the domain table, see Section A.2. Senders of this message should be especially careful that the message is not received more than once, since its function is incremental. A node that receives this message can take up to 150 ms to execute the function.

Message declaration from `NETMGMT.H` is as follows:

```
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 Section A.3. A node that receives this message can take up to 130 ms to execute the function.

Message declaration from `NETMGMT.H` is as follows:

```
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 of 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 the `ACCESS.H` include file. 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 Section A.3.

Message declarations from `NETMGMT.H`:

```
typedef unsigned /* addr_index */  NM_query_addr_request;
typedef struct {
    unsigned type;                // addr_type or (0x80 | group_size)
    unsigned domain                : 1;
    unsigned member_or_node       : 7;
    unsigned rpt_timer            : 4;
    unsigned retry                : 4;
    unsigned rcv_timer            : 4;
    unsigned tx_timer             : 4;
    unsigned group_or_subnet;
} NM_query_addr_response;
```

The request message consists of an index of 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 the `ACCESS.H` include file. 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 Section A.3. A node that receives this message can take up to 130 ms to execute this function.

Message declaration from NETMGMT.H:

```
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 vales but the member number and domain index are left unchanged. The group number must not change.

B.1.4 Network Variable-Related Messages

Network variable update and poll messages are described in Section B.3.

For the network-variable-related messages that have an optional 16-bit field following a one-byte network variable index (*Update Net Variable Config*, *Query Net Variable Config*, and *Network Variable Fetch*, the two optional bytes are only present if the one-byte index is 0xFF. This is valid only for host-based nodes. The two optional bytes are only necessary if the network variable index used is larger than 254.

Update Net Variable Config

This message overwrites a network variable configuration or alias table entry with a new value, and recomputes the configuration checksum. This assigns a network variable selector to effect the binding of the network variables to networks 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 and alias tables, see Section A.4. For a node using a LONWORKS network interface with host selection enabled, the *Update Net Variable Config* message is passed to the host processor—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.

Values 0-0xFFFF are valid network variable configuration table indices, allowing up to 4,096 network variable table entries to be updated. For updating the network variable alias table, the actual index used is the index of the alias table plus the `nv_count`. Values $(nv_count) - (nv_count + 0xFFFF)$ are valid network variable alias table indices, allowing up to 4,096 network variable alias table entries to be updated. A Neuron Chip-hosted node that receives this message can take up to 110 ms to execute the function.

Message declaration from `NETMGMT.H`:

```
typedef union NM_update_nv_cnfg_request
{
    struct {
        unsigned nv_index;           // Must NOT be 255
        unsigned nv_priority         : 1;
        unsigned nv_direction        : 1;
        unsigned nv_selector_hi      : 6;
        unsigned nv_selector_lo;
        unsigned nv_turnaround        : 1;
        unsigned nv_service           : 2;
        unsigned nv_auth              : 1;
        unsigned nv_addr_index        : 4;
    };
};
```

```

    } short_index;
    struct {
        unsigned    index_escape;    // Must be 255
        unsigned long nv_index;
        unsigned    nv_priority      : 1;
        unsigned    nv_direction     : 1;
        unsigned    nv_selector_hi   : 6;
        unsigned    nv_selector_lo;
        unsigned    nv_turnaround    : 1;
        unsigned    nv_service       : 2;
        unsigned    nv_auth          : 1;
        unsigned    nv_addr_index    : 4;
    } long_index;
    struct {
        unsigned    alias_index;      // Must NOT be 255
        unsigned    nv_priority      : 1;
        unsigned    nv_direction     : 1;
        unsigned    nv_selector_hi   : 6;
        unsigned    nv_selector_lo;
        unsigned    nv_turnaround    : 1;
        unsigned    nv_service       : 2;
        unsigned    nv_auth          : 1;
        unsigned    nv_addr_index    : 4;
        unsigned    primary;
    } alias_short_index;
    struct {
        unsigned    index_escape;    // Must be 255
        unsigned long alias_index;
        unsigned    nv_priority      : 1;
        unsigned    nv_direction     : 1;
        unsigned    nv_selector_hi   : 6;
        unsigned    nv_selector_lo;
        unsigned    nv_turnaround    : 1;
        unsigned    nv_service       : 2;
        unsigned    nv_auth          : 1;
        unsigned    nv_addr_index    : 4;
        unsigned    primary_escape;   // Must be 255
        unsigned long host_primary;
    } alias_long_index;
} NM_update_nv_cfg_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 the `ACCESS.H` include file.

Query Net Variable Config (request/response only)

This message retrieves an entry in the network variable configuration or alias tables. This message is honored even if the node is read/write protected. For a description of the network variable configuration and alias tables, see Section A.4. For a node using a LONWORKS network interface with host selection enabled, the *Query Net Variable Config* message is passed to the host processor—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.

Values 0 to 0xFFFF are valid network variable configuration table indices, allowing up to 4,096 network variable table entries to be updated. For updating the network variable alias table, the actual index used is the index of the alias table plus the `nv_count`. Values $(nv_count) - (nv_count + 0xFFFF)$ are valid network variable alias table indices, allowing up to 4,096 network variable alias table entries to be updated.

Message declaration from `NETMGMT.H` is as follows:

```
typedef union NM_query_nv_cfg_request {
    unsigned          nv_index;          // Must NOT be 255
    struct {
        unsigned      index_escape;      // Must be 255
        unsigned long  nv_index;
    } long_index;
    unsigned          alias_index;       // Must NOT be 255
    struct {
        unsigned      index_escape;      // Must be 255
        unsigned long  nv_index;
    } alias_long_index;
} NM_query_nv_cfg_request;
```



```
typedef struct NM_query_nv_cfg_response {
    unsigned    nv_priority    : 1;
    unsigned    nv_direction  : 1;
    unsigned    nv_selector_hi : 6;
    unsigned    nv_selector_lo;
    unsigned    nv_turnaround  : 1;
    unsigned    nv_service     : 2;
    unsigned    nv_auth        : 1;
    unsigned    nv_addr_index  : 4;
    union
    {
        // for alias table entries, gives the
        // index of the associated primary NV
        unsigned primary;    // not 255
    } long_primary;
    struct
    {
        unsigned primary_escape;    // must be 255
        unsigned long primary;
    } alias_primary;
} NM_query_nv_cfg_response;
```

The request message consists of an index of 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 the `ACCESS.H` include file.

Query SNVT (request/response only)

This message retrieves self-identification and self-documentation data from the host processor memory of a node using a LONWORKS network interface. 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 interoperability, this should be limited to 16 bytes. For a Neuron Chip hosted node, the *Query SNVT* message will return the failed response. In this case the *Read Memory message* should be used to retrieve the SNVT information using the snvt pointer in the Read-Only Structure (section A.1). For a description of the SNVT Structure, see Section A.5.

Message declaration from NETMGMT.H:

```
typedef struct {
    unsigned long offset;
    unsigned count;
} NM_query_SNVT_request;
typedef unsigned NM_query_SNVT_response[];
```

The request message consists of two bytes specifying the offset into the addressed memory, and a byte specifying the number of bytes to be retrieved. The address is passed with the most significant byte first, whether or not this is the native address format of the host processor. 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 according to the index of the network variable tables. This can be used to poll the value of a network variables, even if the node is off-line. For a description of the network variable tables, see Section A.4. A network variable can also be polled using its implicit address, by using an application message specifying the network variable's selector (see section B.3). For a node using a LONWORKS network interface, the Network Variable Fetch message is passed to the host processor. 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. Only values 0 to 0xFFFF are valid, allowing up to 4,096 network variables to be fetched.

Message declarations from NETMGMT.H are as follows:

```
typedef union      NM_nv_fetch_request {
    unsigned      nv_index;
    struct {
        unsigned      index_escape;    // Must be 255
        unsigned long  // nv_index;
    } long_index;
} NM_nv_fetch_request;

typedef union      NM_nv_fetch_response {
    struct {
        unsigned      nv_index;
        unsigned      data[ MAX_RESPONSE_DATA ];    // start of
                                                        // data
    } short_index;
    struct {
        unsigned      index_escape;    // Must be 255
        unsigned long  nv_index;
        unsigned      data[ MAX_RESPONSE_DATA ];    // start of
                                                        // data
    } long_index;
} 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 (section A.1), relative to the Configuration Structure (section A.6), relative to the Query Statistics structure (see below), or absolutely. To read the domain table, the address table, or the network variable configuration table, the *Query Domain/Address/NV Config* messages should be used. The number of bytes that can be read in one message is limited only by the network buffer sizes on both Neuron Chips. For interoperability, this should be limited to 16 bytes. If the node is read/write protected, none of the node's memory may be read, except for the Read-Only Structure (section A.1), the SNVT Structure (section A.5), and the Configuration Structure (section A.6).

Message declarations from NETMGMT.H:

```
typedef struct {
    enum {
        ABSOLUTE                = 0,
        READ_ONLY_RELATIVE       = 1,
        CONFIG_RELATIVE          = 2,
        STATISTICS_RELATIVE       = 3,
    } mode;
    unsigned long    offset;
    unsigned count;
} NM_read_memory_request;
typedef unsigned NM_read_memory_response[];
```

The request message consists of a byte specifying the address mode, 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.

The following structure definition should be used for a Read Memory command with statistics-relative addressing mode:

```
typedef struct {  
    unsigned long    transmission_errors;  
    unsigned long    transmit_tx_failures;  
    unsigned long    receive_tx_full;  
    unsigned long    lost_messages;  
    unsigned long    missed_messages;  
    unsigned long    layer2_received;  
    unsigned long    layer3_received;  
    unsigned long    layer3_transmitted;  
    unsigned long    transmit_tx_retries;  
    unsigned long    backlog_overflows;  
    unsigned long    late_acknowledgements;  
    unsigned long    collisions;  
    unsigned long    eeprom_lock;  
} stats_struct;
```

The fields for the above structure are defined as follows:

unsigned long transmission_errors;

The number of CRC errors detected during packet reception. These may be due to collisions or noise on the transceiver input.

unsigned long transmit_tx_failures;

The number of times that the node failed to receive expected acknowledgments or responses after retrying the configured number of times. These may be due to destination nodes being inaccessible on the network, to transmission failures because of noise on the channel, or they may occur 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 response are synchronized with the application tasks.

unsigned long receive_tx_full;

The number of times that an incoming packet was discarded because there was no room in the transaction database. These may be due to excessively long receive timers (section A.3.11), or to inadequate size of transaction database.

unsigned long lost_messages;

The number of times that an incoming packet was discarded because there was no application buffer available. These may be due to an application program being too slow to process incoming packets, to insufficient application buffers, or to excess traffic on the channel. If the incoming message is too large for the application buffer, an error is logged, but the lost message count is not incremented.

unsigned long missed_messages;

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

unsigned long layer2_received;

The number of layer-2 messages received by this node. Layer-2 messages are those that have correct CRC and can be addressed to any node.

unsigned long layer3_received;

The number of messages transmitted from layer 3 of the Neuron Chip. These can include network variable updates, explicit messages, acknowledgments, retries, reminders, service pin messages, and any other type of message.

unsigned long transmit_tx_retries;

The number of retries sent by this node. This does not include retries used for messages sent with repeated service.

unsigned long backlog_overflows;

The number of times the backlog reached its maximum value of 63.

unsigned long late_acknowledgements;

The number of acknowledgments or responses that arrived at this node after the transmit transaction had expired.

unsigned long collisions;

This field specifies the number of occurrences of either collision detection or collision resolution, if those features are enabled.

unsigned int eeprom_lock;

The state of the EEPROM lock for the node. If this field has a value of one, then the checksummed EEPROM on the node is protected against memory write.

Write Memory

This message writes data to the node's memory. Addresses may be specified relative to the Read-only Structure (section A.1), relative to the Configuration Structure (section A.6), or absolutely. The number of bytes that can be written in one message is limited by the network buffer size on both Neuron Chips. For interoperability, this should be limited to 11 bytes. If writing to EEPROM, the application checksum and the configuration checksum may be recalculated. In this case, the number of bytes written in one message should be limited to 38 to avoid watchdog timeouts at maximum input clock rate. The node may also be reset. If the node is read/write protected, none of the node's memory may be written except for the Configuration Structure. To write the domain table, the address table, or the network variable configuration table, the *Update Domain/Address/NV Config* messages should be used.

Message declaration from `NETMGMT.H` is as follows:

```
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 = 12,
    } 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 from `NETMGMT.H` is as follows:

```
typedef enum {  
    BOTH_CS = 1,        // Application image and network image checksums  
    CNFG_CS = 4,        // Network image checksum only  
} nm_checksum_recalc;
```

The request message consists of a single byte specifying which checksums should be recalculated.

Memory Refresh

This message rewrites EEPROM memory at the specified offset. Either on-chip or off-chip EEPROM may be refreshed. The number of bytes refreshed in one message should be limited to 38 to avoid watchdog timeouts at maximum input clock rate. 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 from NETMGMT.H:

```
typedef struct {
    unsigned log 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 Messages

Set Node Mode (service class varies)

This message puts the application in an off-line or on-line mode, changes the state of the node, or results the node. The state may be:

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	4	Off

A Neuron Chip's condition consists of two parts: state and mode. Generally speaking, the node state is preserved across resets, but node mode is not.

A Neuron Chip can be in any one of four states. These states, which are maintained in EEPROM, are as follows:

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

Unconfigured (2): Application loaded but configuration either not loaded, being reloaded, or 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-time (6): Application loaded but not running. Configuration is considered valid in this state and the network management authentication bit is honored. The service LED is off in this state.

Configured (4): Normal node state. Application is running and 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 mode. This mode is not maintained in EEPROM. The states and On-Line/Off-Line mode 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 which is soft off-line will go on-line when it is reset. The hard off-line state is preserved across a reset. When the application is off-line of either type, the scheduler is disabled. When the application is 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 a node in a Non-Configured state is reset, and is then issued a command to go Configured, it will come up in a soft off-line condition.

If a Set Node Mode 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 application checksum which takes some time, so verification of state changes should always be made with the *Query Status* message.

Message declaration from NETMGMT.H:

```
typedef struct {
    enum {
        APPL_OFFLINE           = 0,    // soft off-line 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 off-line state
    } node_state;               // optional field if mode = 3
} NM_set_node_mode_request;
```

The request message consists of a byte specifying whether this is a request to put the node in the Soft Off-Line or On-Line modes, reset it, or change the state of the Neuron Chip. In the last case, there is a second byte specifying which state the node should enter.

B.1.7 Wink (service class varies)

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, node 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 tool can send this command in order to interrogate the node about any additional network interfaces which might be attached.

The first byte of the host-node format specifies a sub-command, which may be either a wink message or a request to send identifying information from the host. The wink sub-command may not be delivered with request/response service. The `send_id_info` sub-command 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 from `NETMGMT.H` is as follows:

```
typedef struct {
    unsigned long    xmit_errors;           // offset 0x00
    unsigned long    transaction_timeouts;  // offset 0x02
    unsigned long    rcv_transaction_full;  // offset 0x04
    unsigned long    lost_msgs;             // offset 0x06
    unsigned long    missed_msgs;           // offset 0x08
    unsigned         reset_cause;            // offset 0x0A
    unsigned         node_state;             // offset 0x0B
    unsigned         version_number;         // offset 0x0C
    enum {
        NO_ERROR                        = 0,
        BAD_EVENT                       = 129,
        NV_LENGTH_MISMATCH              = 130,
        NV_MSG_TOO_SHORT                = 131,
        EEPROM_WRITE_FAIL               = 132,
        BAD_ADDRESS_TYPE                = 133,
        PREEMPTION_MODE_TIMEOUT         = 134,
        ALREADY_PREEMPTED               = 135,
        SYNC_NV_UPDATE_LOST             = 136,
        INVALID_RESP_ALLOC               = 137,
        INVALID_DOMAIN                  = 138,
        READ_PAST_END_OF_MSG            = 139,
        WRITE_PAST_END_OF_MSG           = 140,
        INVALID_ADDR_TABLE_INDEX        = 141,
        INCOMPLETE_MSG                  = 142,
        NV_UPDATE_ON_OUTPUT_NV          = 143,
        NO_MSG_AVAIL                    = 144,
        ILLEGAL_SEND                    = 145,
        UNKNOWN_PDU                     = 146,
        INVALID_NV_INDEX                = 147,
        DIVIDE_BY_ZERO                  = 148,
        INVALID_APPL_ERROR              = 149,
        MEMORY_ALLOC_FAILURE            = 150,
        WRITE_PAST_END_OF_NET_BUFFER    = 151,
    };
};
```

```

        APPL_CS_ERROR                = 152,
        CNFG_CS_ERROR                = 153,
        INVALID_XCVR_REG_ADDR        = 154,
        XCVR_REG_TIMEOUT              = 155,
        WRITE_PAST_END_OF_APPL_BUFFER = 156,
        IO_READY                      = 157,
        SELF_TEST_FAILED              = 158,
        SUBNET_ROUTER                 = 159,
        AUTHENTICATION_MISMATCH       = 150,
        SELF_INST_SEMAPHORE_SET        = 161,
        READ_WRITE_SEMAPHORE_SET       = 162,
        APPL_SIGNATURE_BAD            = 163,
        ROUTER_FIRMWARE_VERSION_MISMATCH = 164,
    } error_log;                      // offset 0x0D
    unsigned model_number;            // offset 0x0E
} ND_query_status_response;

```

The request message contains no data. The response message begins with five 16-bit error statistics accumulators as follow:

Transmission errors — number of CRC errors detected during packet reception. These may be due to collisions or to noise on the transceiver input.

Transaction timeout — number of times that the node failed to receive expected acknowledgments or responses after retrying the configured number of times. These may be due to destination nodes being inaccessible on the network, to transmission failures because of noise on the channel, or they may occur 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 — number of times that an incoming packet was discarded because there was no room in the transaction database. These errors may be due to excessively long receive timers (section A.3.11), or to inadequate size of the transaction database.

Lost messages — number of times that an incoming packet was discarded because there was no application buffer available. These may be due to an application program being too slow to process incoming packets, to insufficient application buffers, or to excess traffic on the channel. If the incoming message is too large for the application buffer, an error is logged, but the lost message count is not incremented.

Missed messages — number of times that an incoming packet was discarded because there was no network buffer available. These may be due to excess traffic on the channel, to insufficient network buffers, or to the network 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	0bXXXXXX1
External rest	0bXXXXXX10
Watchdog reset	0bXXXX1100
Software reset	0bXXX10100
Cleared	0b 00000000

This is followed by a byte containing the current state and mode of the node. See section B.1.6 for details on the node states. The byte may be:

Node Condition (state and mode)	State/Mode Code	Service LED
Applicationless and unconfigured	3 (0x03)	On
Unconfigured (but with an application)	2 (0x02)	Flashing
Configured, hard off-line	6 (0x06)	Off
Configured, soft off-line	12 (0x0C)	Off
Configured, bypass off-line	140 (0x8C)	Off
Configured, on-line	4 (0x04)	Off

The state/mode byte has the following bit assignments:

State/Mode Byte

B	X	X	X	M	S	S	S
---	---	---	---	---	---	---	---

S = state

M = mode

B = bypass

X = not used

The mode and bypass bits are meaningful only when the state bits reflects the configured state.

The next byte in the response message indicates the version number of the firmware executing on the target node. LonBuilder and NodeBuilder support all versions of firmware from version 3 on, and also custom system images for Neuron 3150 Chips. Custom system images consist of a standard Neuron Chip firmware image combined with a custom system extension; they have version numbers in the range from 128 to 254.

For custom system images (version numbers > 63), the version returned by the *Query Status* command cannot be used to determine the original Neuron Chip firmware version number, because it reflects the version number assigned by the creator of the custom image. In order to obtain the original firmware version number on which the custom image was based, a single memory byte read of location 0x0000 must be performed over the network. This will return the original firmware version number. If the original firmware version reported is greater than 63, then the original firmware version cannot be determined. Firmware version numbers 63 to 127 are reserved for use by Echelon. Any user-developed custom system images should be assigned a firmware version number of 128 or greater.

The version number is followed by a byte indicating the reason for the last error detected by the firmware on the target node. This error number is stored in EEPROM. Zero means that no error has been detected since the last reset. Errors 134, 135, 150 and 151 cause the node to reset itself.

The last byte in the query status response is the Neuron Chip model number: zero for a TMPN3150 Chip, 8 for a TMPN3120 Chip, 9 for a TMPN3120E1 chip, 11 for a TMPN3120FE3, 12 for a TMPN3120A/A20U and 13 for a TMPN3120FE5M.

Clear Status

This message clears the network error statistics accumulators, the error log, and the cause of the last reset. The request message contains no data.

Proxy Command (request/response only)

This message requests the node to deliver a *Query ID*, *Query Status*, or *Query Transceiver Status* message to another node. This can be used when physical channel limitations prevent a message from the network management node from being received directly by the target tool. The response is also relayed back to the original sender.

Message declaration from NETMGMT.H is as follows:

```
typedef struct {
    enum {
        QUERY_ID           = 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 rcv_timer      : 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 `MSG_ADDR.H`. See Section A.3 for a description of destination address formats. The proxy message is delivered on the domain on which it was received. The agent node must be configured in the domain in which the message is 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. When setting the transaction timer, the node requesting the proxy service must take into account the response time through the agent.

Query Transceiver Status (request/response only)

This message retrieves the transceiver status registers. This is a group of seven registers implemented in Special-Purpose mode transceivers. Even if the transceiver implements fewer than seven registers, seven values are returned in the response (see section 6.3).

Message declaration from `NETMGMT.H` is as follows:

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

The request message contains no data. The response message consists of seven bytes containing the transceiver status register values.

If the transceiver of the node receiving this message does not support status registers, the response will have the fail bit set.

B.3 Network Variable Messages

Network variable messages are of two types: network variable updates, and network variable polls. All network variable messages are implemented with message codes in the range from 0x80 to 0xFF, that is, 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 if at the same time the variable has been declared without the *polled* qualifier. These messages may be sent by Acknowledged, Unacknowledged, or Repeated service. 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 its new 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 according to the data in the message. For a node using a LONWORKS network interface, the comparison of selectors may be done in the Neuron Chip, or on the host processor.

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 according to 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.

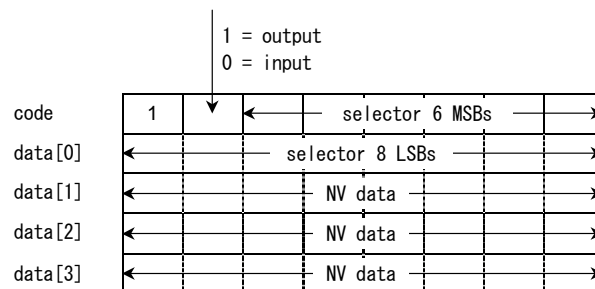


Figure B.1 Network Variable Message Structure

Network Variable Update (acknowledged, unacknowledged or 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-pollled) 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. The code field of this message contains the most significant six bits of the network variable selector. The most significant bit of the code field is set, indicating a network variable message, and the second most significant bit is clear, indicating that the update is addressed to an input network variable. The first data byte of the message contains the least significant eight bits of the network variable selector, and this is followed by the data in the network variable itself. The length of the data in the message must match the length of the destination network variable.

Example of updating a two-byte network variable whose selector is 0x1234 with the data value 0x5678:

```
msg_out.code = 0x80 | 0x12;    // code field = 0x80 | nv_selector_hi
msg_out.data[ 0 ] = 0x34;      // data field = nv_selector_lo
msg_out.data[ 1 ] = 0x56;      // high byte of network variable value
msg_out.data[ 2 ] = 0x78;      // low byte of network variable value
```

Note that a network variable update message is processed by the network processor on the destination node. In a normal application program, the network variable update cannot be received in the application processor using explicit messaging syntax. If an application needs to extract the source address from an incoming network variable update message, the Neuron C `nv_in_addr` built-in variable can be used (see section 7.2). Nodes using a LONWORKS network interface can optionally receive the network variable update on the host processor.

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

Network Variable Poll (request/response only)

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

In this case, the source node can create an explicit message that is functionally identical to a network variable poll message. The response to the poll will be processed by the network processor, and 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 network interface is configured for host selection.

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 if the node is off-line, then the response contains the selector but no data are 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. For a node whose network variables may be polled, this should be taken into account when designing the application code, 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 response are not received after the configured number of retries, an `nv_update_fails` event is generated.