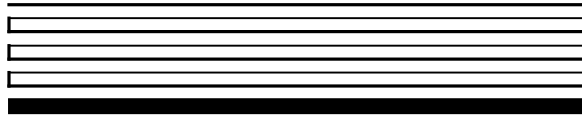


# Gizmo 4 User's Guide

Revision 2



078-0191-01B



Echelon, LON, LonTalk, LonBuilder, NodeBuilder, Neuron, LonManager, LONMARK, LONWORKS, 3120, and 3150 are registered trademarks of Echelon Corporation. LonMaker, ShortStack, and *i.LON* are trademarks of Echelon Corporation.

Other brand and product names are trademarks or registered trademarks of their respective holders.

Neuron Chips, Serial LonTalk Adapters, and other OEM Products were not designed for use in equipment or systems which involve danger to human health or safety or a risk of property damage and Echelon assumes no responsibility or liability for use of these products in such applications.

Parts manufactured by vendors other than Echelon and referenced in this document have been described for illustrative purposes only, and may not have been tested by Echelon. It is the responsibility of the customer to determine the suitability of these parts for each application.

ECHELON MAKES NO REPRESENTATION, WARRANTY, OR CONDITION OF ANY KIND, EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE OR IN ANY COMMUNICATION WITH YOU, INCLUDING, BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, FITNESS FOR ANY PARTICULAR PURPOSE, NONINFRINGEMENT, AND THEIR EQUIVALENTS.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of Echelon Corporation.

Printed in the United States of America.  
Copyright ©2001-2002 by Echelon Corporation.

Echelon Corporation  
[www.echelon.com](http://www.echelon.com)

# Contents

<b>1</b>	<b>Gizmo 4 Overview</b>	<b>1-1</b>
	Introduction	1-2
	Audience	1-4
	Content	1-4
	Related Documentation	1-4
<b>2</b>	<b>Gizmo 4 Hardware and Electrical Specifications</b>	<b>2-1</b>
	Gizmo 4 Component Placement	2-2
	Gizmo 4 Power Supply	2-3
	PIC, Display, Clock, and A/D Converter	2-3
	Push Buttons	2-4
	D/A Converter and Loop Back D/A to A/D Mode	2-4
	LED1 and LED2	2-4
	Quadrature Encoder	2-4
	Gizmo 4 Block Diagram	2-5
	Gizmo 4 Electrical Specifications	2-6
<b>3</b>	<b>Gizmo 4 Jumper and Connector Descriptions</b>	<b>3-1</b>
	Gizmo 4 Connectors	3-2
	J1 I/O Connector	3-2
	J2 Control Module I/O Connector	3-3
	J3 Signal Routing Connector	3-4
	J4 Prototype I/O Connector	3-5
	J5 Control Module Network Connector	3-5
	J6 Analog Output Connector	3-6
	J7 Analog Input Connector	3-6
	J8 Digital I/O Connector	3-7
	J9 Digital I/O Connector	3-7
	J10 and J12 Power Supply Connectors	3-8
	J11 Network Connector	3-8

Gizmo 4 Jumpers	3-8
JP7 D/A 1 to A/D 1 Loop-Back Mode	3-8
JP8 D/A 2 to A/D 2 Loop-Back Mode	3-9
JP12 Gizmo 4 Power Supply Source Selection	3-9
JP13 Piezo Speaker Select	3-9
Optional Gizmo 4 Connectors and Jumpers	3-9
J13 PIC Programming Connector	3-9
J16 Unused PIC I/O Connector	3-10
JP10 PIC Stand Alone Mode Select	3-10
JP11 D/A Stand Alone Mode Select	3-10
JP14 PIC Reset Enable	3-10
JP17 SPI Stand Alone Mode Select	3-10
<b>4 Gizmo 4 Software</b>	<b>4-1</b>
Getting Started	4-2
Macros in Gizmo4.h	4-3
GIZMO_DISPLAY_*	4-3
GIZMOPIC_CS_DELAY	4-3
USE_FREQUENCY	4-3
USE_QUADRATURE	4-4
Types	4-4
tGizmoNote	4-4
tGizmoTone	4-5
tGizmoToneLength	4-5
Functions	4-6
GizmoBuzz()	4-6
GizmoClearDisplay()	4-6
GizmoDefineCharacter()	4-6
GizmoDisplayChar()	4-7
GizmoDisplayDate()	4-7
GizmoDisplayFormattedNumber()	4-8
GizmoDisplayNumber()	4-8
GizmoDisplayString()	4-9
GizmoDisplayTemperature()	4-9
GizmoDisplayTime()	4-10
GizmoGetTime()	4-10
GizmoPlayTune()	4-10
GizmoReadAnalog()	4-11
GizmoReadTemperature()	4-11
GizmoReset()	4-11
GizmoRTC()	4-11
GizmoSetLed()	4-11
GizmoSetTime()	4-12
GizmoWriteAnalog()	4-12

I/O Objects	4-12
LEDs and Quadrature Input	4-12
Neurowire	4-13
Push Buttons	4-13
Error Management	4-13
Communicating Directly with the Gizmo 4	4-14
Communicating with the I/O Wired to the Neuron Chip	4-15
Push Buttons	4-15
LEDs	4-15
Quadrature Encoder	4-15
Piezo Speaker	4-16
I/O Controlled by the PIC	4-16
Register Tables	4-17
<b>Appendix A Creating a Custom Character</b>	<b>A-1</b>
Custom Character Example	A-2
<b>Appendix B Gizmo 4 Character Set</b>	<b>B-1</b>
Character Set	B-2



# 1

## Gizmo 4 Overview

The Gizmo 4 I/O Board is a LONWORKS® product that can be used as a prototyping, testing, and learning tool for the LonBuilder® or NodeBuilder® Development Tool and other LONWORKS products. With minor hardware modification, the Gizmo 4 board can also be used as a test or training platform for the ShortStack Developer's Kit.

---

## Introduction

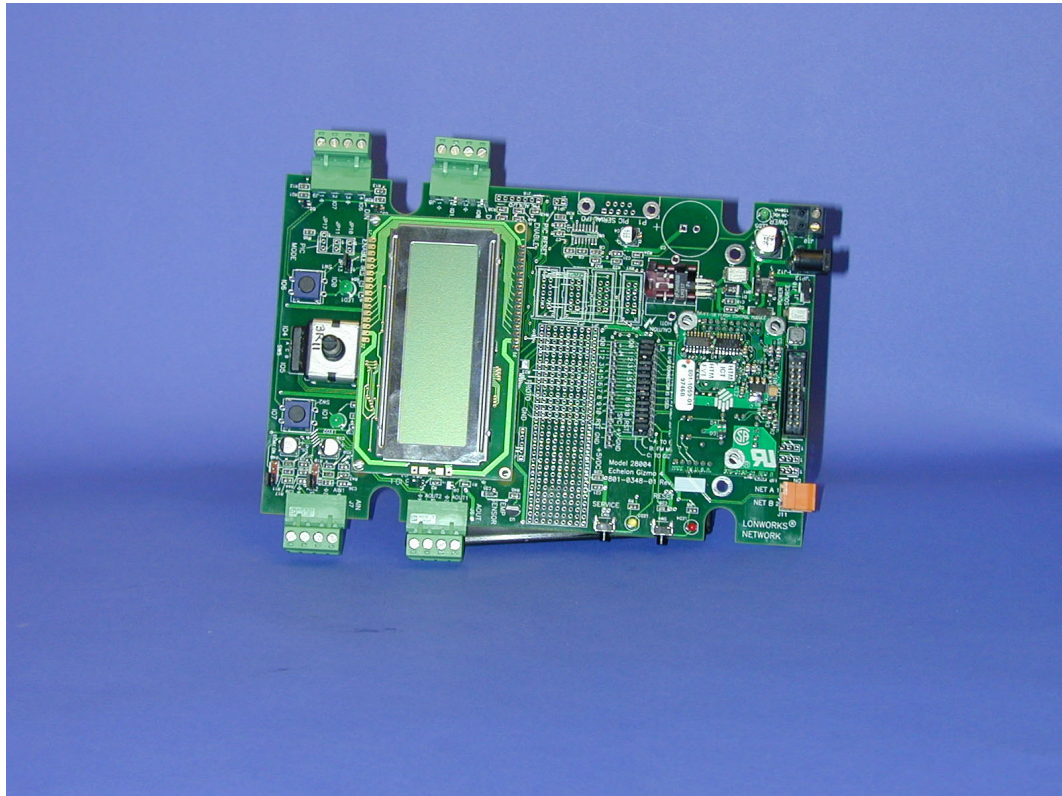
The Gizmo 4 I/O Board is a collection of I/O devices that you can use with the LTM-10A Platform or an embedded control module for developing prototype devices and I/O circuits, developing special-purpose devices for testing, running the NodeBuilder examples, or learning more about the LONWORKS platform. The following figure illustrates the Gizmo 4 plugged into the LTM-10A Platform.



Gizmo 4 Plugged-into LTM-10A Platform

You can also plug a TP/FT-10 or TP/FT-10F Control Module into the Gizmo 4 to create a self-contained LONWORKS device. This requires separate purchase of the TP/FT-10 or TP/FT-10F Control Module. The following figure illustrates this configuration.





Gizmo 4 with a TP/FT-10F Flash Control Module

The Gizmo 4 includes the following I/O devices:

- 4 row x 20 character LCD display
- 2 10-bit resolution analog inputs with screw terminal connector
- 2 8-bit resolution analog outputs with screw terminal connector
- 2 digital inputs with screw terminal connector and pushbutton inputs
- 2 digital outputs with screw terminal connector and LED outputs
- Digital shaft encoder (quadrature encoder)
- Piezoelectric speaker
- Real-time clock
- Temperature sensor

The Gizmo 4 also includes a prototyping area that you can use to add your own hardware prototypes, or to extend the hardware on the Gizmo 4 board.

A Gizmo 4 library is included with the NodeBuilder software that provides easy-to-use high-level functions for accessing the display, analog I/O, piezo speaker, real-time clock, and temperature sensor. The Gizmo 4 library is also available in the Echelon Developer's Toolbox at [www.echelon.com/toolbox](http://www.echelon.com/toolbox).

---

## Audience

This document describes how to use the Gizmo 4 hardware and the example application programs. Multi-device application examples are included to help you get started.

---

## Content

This manual provides detailed hardware specifications as well as a description of the Gizmo 4 library.

The examples contain sample source code to demonstrate a variety of I/O objects supported by Neuron C and the Gizmo 4. The examples are designed to support several different types of LONWORKS applications.

---

## Related Documentation

The following Echelon publications are suggested for additional information:

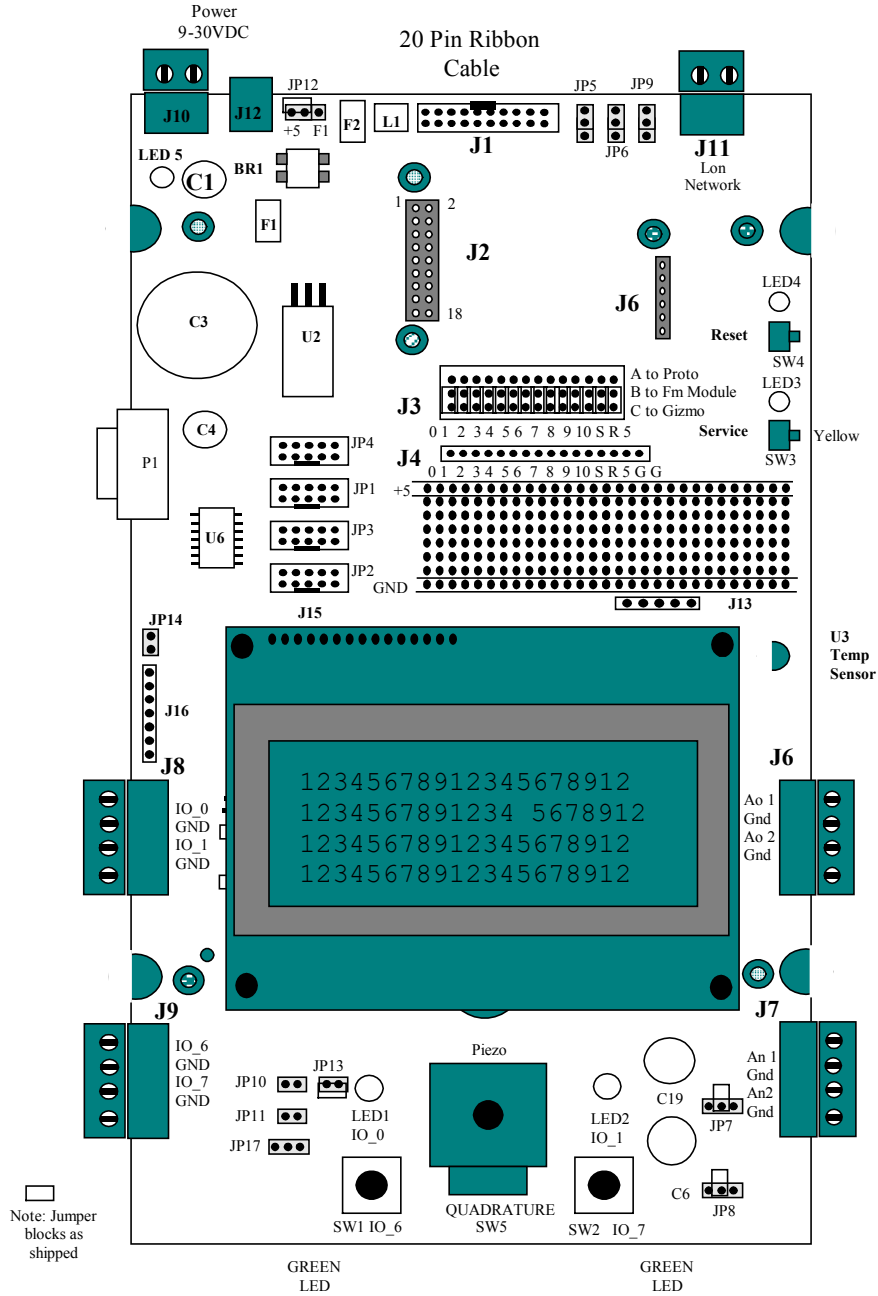
- *LonBuilder User Guide* (078-0001-01)
- *NodeBuilder User Guide* (078-0141-01)
- *Neuron C Programmer's Guide* (078-0002-01)
- *Neuron C Reference Guide* (078-0140-01)

# 2

## **Gizmo 4 Hardware and Electrical Specifications**

This chapter provides schematics and diagrams for the Gizmo 4 hardware and circuitry. It also provides the electrical specifications for the Gizmo 4.

# Gizmo 4 Component Placement



---

## Gizmo 4 Power Supply

You can supply power to the Gizmo 4 board using any of the following methods:

- +5V input to the I/O connector at J1. You will use this method if you are connecting the Gizmo 4 board to an LTM-10A Platform. The JP12 Power Source Jumper must be set to J1.
- 9-30VDC input to the Screw Terminal Power connector at J10. You will use this method if you have installed a TP/FT-10 or TP/FT-10F Control Module on the Gizmo 4 board and you are connecting the Gizmo 4 board to a 9-30VDC power supply with flying-lead output. The JP12 Power Source Jumper must be set to J10.
- 9-30VDC input to the Power connector at J12. You will use this method if you have installed a TP/FT-10 or TP/FT-10F Control Module on the Gizmo 4 board and you are connecting the Gizmo 4 board to a 9-30VDC power supply with 2.1mm connector output. The JP12 Power Source jumper must be set to J10.

The Gizmo 4 requires under 50ma of power. If you are using the Gizmo 4 with a control module, the Power Supply Source jumper needs to be installed in JP12 to select between external 9-30 volts from either J10 or J12. For power from J10 or J12, locate a jumper between center pin and top pin.

If you are using the Gizmo 4 with an LTM-10A or LonBuilder Emulator, they will provide power to the Gizmo 4. First connect the J1 jumper from center pin to bottom pin. Power is supplied through the ribbon cable to header J1, pins 19 and 20.

Not including the control module and any external hardware that you add to the Gizmo 4 board, the power supply current required for the Gizmo 4 board is as follows:

J1	+5VDC @ 12ma
J10 or J12	+9VDC @ 23ma, +30VDC @ 10ma

---

## PIC, Display, Clock, and A/D Converter

The Gizmo 4 board includes a Microchip PIC16F877 microcontroller. This chip manages the LCD display, has an on-board 10 bit A/D, and performs the real-time clock function in software. The Neuron Chip interfaces to the PIC via the SPI port and can use Neurowire I/O to communicate to the PIC. You can reprogram the PIC by downloading new code through the header pins on J13. With minor hardware modification to the Gizmo 4 board, you can install a ShortStack Module on the Gizmo 4 board and reprogram the PIC with a ShortStack application.

The A/D within the PIC is accessed through connector J7, which connects to the PIC An0 and An1 inputs. The PIC An2 input has a National Semiconductor LM34 temperature sensor attached. This sensor changes 1mv per degree C. The PIC performs the calculations and gives the Neuron Chip access to both Celcius and Fahrenheit readings.

---

## Push Buttons

There are two push buttons of the Gizmo 4 that can be connected to the Neuron Chip I/O pins. Button SW1 connects to IO7 and SW2 connects to IO6. In addition, both buttons are connected to input pins RB2 and RB3 of the PIC, which are only used in the test mode.

The other two buttons on the Gizmo 4 are the SW3 Neuron Chip service input and SW4 Neuron Chip reset input. You can connect the SW4 button to the PIC reset input by adding a jumper at JP14. Note that JP14 will reset PIC TRC when SW4 is pressed.

---

## D/A Converter and Loop-back D/A to A/D Mode

There is a Maxim MAX522 dual 8 bit D/A converter that provides two 8-bit analog outputs. The Neuron Chip uses the SPI port to communicate to the D/A converter, with IO3 as the chip enable pin.

You can loop the output of the D/A converter back to the input of the A/D converter for testing or running examples. To do this, place jumpers in JP7 and JP8. This is the same as if wires were connected between D/A Analog Output Connector J6 to A/D Analog Input Connector J7. You can insert an analog delay in the loop-back by insert jumpers at JP6 and JP9. This adds a 200 $\mu$ f capacitor to the loop-back connection.

---

## LED1 and LED2

There are two output LEDs. LED1 is connected to Neuron Chip IO0 and LED2 is connected to Neuron Chip IO1. IO0 is also connected to the piezo speaker. If you want to drive the LED, but not the piezo, remove jumper JP13. If you choose to drive the piezo, the LED will always come on.

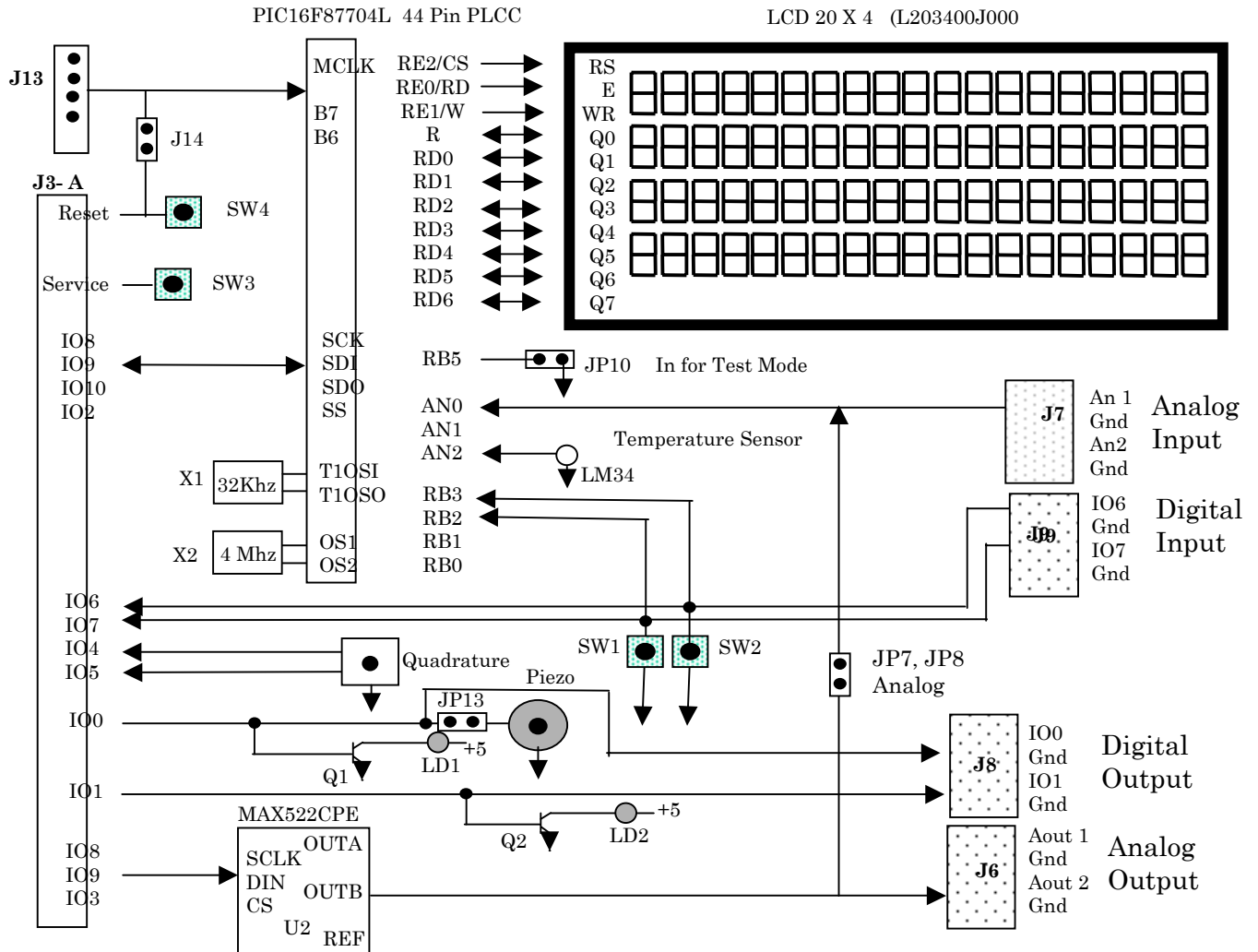
You can access the IO0 and IO1 outputs through the J8 DOut Connector.

---

## Quadrature Encoder

There is a quadrature shaft encoder connected to IO4 and IO5 of the Neuron Chip. This encoder is a detent type and increments in counts of two. The encoder pulls the input pins to ground when turning, so the Neuron Chips internal pull-up resistors must be enabled to use the encoder.

# Gizmo 4 Block Diagram



## Gizmo 4 Electrical Specifications

Con.	Function	Symbol	Pins	Min	Max	Max
J10	Input	Power Supply VDD	1-2	9 VDC	30VDC	25 ma
J12	Input	Power Supply VDD	1-2	9 VDC	30VDC	25 ma
J1	Input	Power Supply VDD	19	4.75	5.5	15 ma
J6	Output	Aout1 - Aout 2	1 - 3	0.2	VDD -.8	1 ma
J7	Input	Ain 1 – Ain 2	1 - 3	0	VDD + .5	20 ua
J8	Input	IO0 – IO1		0	VDD + .5	10 ua
J9	Output	IO6 – IO7		0.2	VDD-.2	20 ma sink
J9	Output	IO6 – IO7		0.2	VDD-.2	1 ma source
J1	Input	IO0 – IO1	1 - 2	0	VDD + .5	10 ua
J1	Input	IO2 – IO3	3 - 4	0	VDD + .5	1 ua
J1	Output	IO4 – IO5	5 - 6	0.2	VDD-.2	10 ua
J1	Output	IO6 – IO7	7 – 8	0.2	VDD-.2	10 ua
J1	Input	IO8 – IO9	9 -10	0	VDD + .5	1 ua
J1	Output	IO10	11	0.2	VDD-.2	1 ma
J1	Input/Output	Service	15	0	VDD + .5	10ma
J1	Input/Output	Reset	16	0	VDD + .5	10ua



# 3

## **Gizmo 4 Jumper and Connector Descriptions**

This chapter provides connector and jumper descriptions for the Gizmo 4 I/O Board.

---

## Gizmo 4 Connectors

This section describes the Gizmo 4 connectors.

---

### J1 I/O Connector

The J1 I/O connector interfaces to an LTM-10A Platform or compatible device. Pin 19 is the optional +5VDC power supply input.

**Table 3.1** J1 2x10 Pin Header for Ribbon Cable

Pin	Name	* Default Function (See note below)
1	IO0	Input to Piezo & LED 1
2	IO1	Input to LED 2
3	IO2	Input to PIC Chip Select
4	IO3	Input to D/A Chip Select
5	IO4	Output from Quadrature encoder
6	IO5	Output from Quadrature encoder
7	IO6	Output Switch SW2 Normally High
8	IO7	Output Switch SW1 Normally High
9	IO8	Input to PIC and D/A SCLK
10	IO9	Input to PIC and D/A Data In
11	IO10	Output from PIC and D/A Data Out
12		
13		
14		
15	Service	Output from Switch SW4
16	Reset	Output from Switch SW3
17		
18		
19	VDD	+ 5VDC Input
20	GND	Ground

*Note: This description works only if jumpers are installed between the B and C rows in connector J3.*

---

## J2 Control Module I/O Connector

The J2 Control Module I/O connector is an 18-pin connector for interfacing a TP/FT-10 or TP/FT-10F Control Module to the Gizmo 4. You cannot use the J1 I/O connector when a control module is inserted.

**Table 3.2** J2 2x18-Pin Control Module I/O Connector

Pin	Name	Function
1	N.C.	
2	IO0	Neuron Chip IO0
3	GND	Ground
4	IO1	Neuron Chip IO1
5	GND	Ground
6	IO2	Neuron Chip IO2
7	GND	Ground
8	IO3	Neuron Chip IO3
9	Reset	Reset Output
10	IO4	Neuron Chip IO4
11	IO5	Neuron Chip IO5
12	VDD	+5 volts Output
13	IO6	Neuron Chip IO6
14	IO9	Neuron Chip IO9
15	IO7	Neuron Chip IO7
16	IO10	Neuron Chip IO10
17	IO8	Neuron chip IO8
18	Service	Service Output

---

## J3 Signal Routing Connector

The J3 Signal Routing connector is used to connect the Neuron Chip I/O pins from the J1 I/O Connector or the J2 Control Module I/O Connector to either the Gizmo 4 I/O components or the Gizmo 4 prototype area. Row A is connected to the prototyping area. Row B is connected to both the J1 I/O Connector and J2 Control Module I/O Connector. Row C is connected to the Gizmo 4 I/O components. Place a jumper across pins B and C to connect a Neuron Chip I/O signal to the on-board circuitry. Place a jumper across pins B and A to connect a Neuron Chip I/O signal to the prototype area on the Gizmo 4 board. There is a separate set of pins for each I/O signal.

**Table 3.3** J3 3 x 14 Pin Header to Select Neuron Chip Source

Pin	Pin	Pin	Name	Function
A*	B**	C***		
1	1	1	IO0	Input to Piezo & LED 1
2	2	2	IO1	Input to LED 2
3	3	3	IO2	Input to PIC Chip Select
4	4	4	IO3	Input to D/A Chip Select
5	5	5	IO4	Output from Quadrature Encoder
6	6	6	IO5	Output from Quadrature Encoder
7	7	7	IO6	Output Switch SW2 Normally High
8	8	8	IO7	Output Switch SW1 Normally High
9	9	9	IO8	Input to PIC and D/A SCLK
10	10	10	IO9	Input to PIC and D/A Data In
11	11	11	IO10	Output from PIC and D/A Data Out
12	12	12	Service	Service Input
13	13	13	Reset	Reset Input
14	14	14	VDD	+5VDC Output

Notes:

- \* Column A connects to the prototype area
- \*\* Column B connects to both the J1 I/O Connector and the J2 Control Module I/O Connector
- \*\*\* Column C connects to the Gizmo 4 I/O components

---

## J4 Prototype I/O Connector

The J4 Prototype I/O connector connects to the A-side of the J3 connector and can be used for connecting components in the prototype area to the Neuron Chip I/O pins.

**Table 3.4** J4 1x16-Pin Header for Prototype Area

Pin	Name	Function
1	IO0	Input to Piezo & LED 1
2	IO1	Input to LED 2
3	IO2	Input to PIC Chip Select
4	IO3	Input to D/A Chip Select
5	IO4	Output from Quadrature Encoder
6	IO5	Output from Quadrature Encoder
7	IO6	Output Switch SW2 Normally High
8	IO7	Output Switch SW1 Normally High
9	IO8	Input to PIC and D/A SCLK
10	IO9	Input to PIC and D/A Data In
11	IO10	Output from PIC and D/A Data Out
12	Service	Service Input
13	Reset	Reset Input
14	VDD	+5VDC Output
15	GND	Ground
16	GND	Ground

---

## J5 Control Module Network Connector

The J5 Control Module Network connector is a 6-pin connector for interfacing a TP/FT-10 or TP/FT-10F Control Module to the Gizmo 4. You cannot use the J1 I/O Connector with a control module inserted.

**Table 3.5** J5 1x6-Pin Control Module Network Connector

Pin	Name	Function
1	N.C	No Connection
2	N.C	No Connection
3	Net A	Connects to J11 LONWORKS Network Connector
4	Net B	Connects to J11 LONWORKS Network Connector
5	N.C	No Connection
6	N.C	No Connection

---

## J6 Analog Output Connector

The J6 Analog Output connector provides the Aout1 analog output of the 8-bit Maxim MAX522 D/A converter. Send an 8-bit value to the D/A converter to produce a 0 to 5 volt output in 20mv increments. The serial interface to the MAX522 is compatible with Microwire, SPI, or Neuron Neurowire.

To send an output to the D/A converter, send two 8-bit values, where the first is the command value and the second is the data value. Part of the command value is the address of the specific D/A converter. The two address registers are as follows:

Analog Out 1	0x21
Analog Out 2	0x22

**Table 3.6** J6 1x4-Pin Connector for Analog Output

Pin	Name	Function	Specifications
1	Aout 1	Output from MAX 522 D/A Output A	0 to 4.99 Volts / 256Bits
2	GND	Ground	
3	Aout 2	Output from MAX 522 D/A Output B	0 to 4.99 Volts / 256Bits
4	GND	Ground	

---

## J7 Analog Input Connector

The J7 Analog Input connector provides the Ain1 analog input to the 10-bit A/D converter of the 16F877 PIC processor. The input voltage range is determined by the reference voltage, which is set at 5 volts. The A/D resolution is 10-bits or 1024 divided by the total reference voltage. The A/D samples at a 200µs rate. The resulting 2-byte value is stored at a PIC memory location. The serial interface to the 16F877 PIC is compatible with Microwire, SPI, or Neuron Neurowire. Read the following locations to access to the resulting A/D samples:

Analog In 1	0x07	LSB
	0x08	MSB
Analog Out 2	0x09	LSB
	0x10	MSB

**Table 3.7** J7 1x4-Pin Connector for Analog Input

Pin	Name	Function	Specifications
1	AIn 1	Analog Input	0 to 5 volts
2	GND	Ground	
3	AIn 2	Analog Input	0 to 5 volts
4	GND	Ground	

---

## J8 Digital I/O Connector

The J8 Digital I/O connector provides ESD-protected Neuron IO0 and IO1 signals. These signals are typically configured as digital outputs, but they can be configured as either output or input signals. PNP transistors are connected to IO0 and IO1 to drive LED1 and LED2. The base of the transistors is referenced to +5 through a 5K $\Omega$  resistor with a 10K $\Omega$  pull-up, pulling the base low turns on the respective transistor.

**Table 3.8** J8 1x4-Pin Connector for Digital Output

Pin	Name	Function	Specifications
1	IO0	Digital Output from Neuron Chip	0 to 5 volts
2	GND	Ground	
3	IO1	Digital Output from Neuron Chip	0 to 5 volts
4	GND	Ground	

---

## J9 Digital I/O Connector

The J9 Digital I/O connector provides ESD-protected Neuron IO6 and IO7 signals. These I/O signals are typically configured as digital inputs and each has an on-board switch connected to ground. IO6 connects with switch SW1 and IO7 connects with switch SW2. Each signal line has a 10K $\Omega$  pull-up resistor for reference. The Neuron Chip can configure IO6 and IO7 as outputs; however, in this mode, the 100 $\Omega$  series protection resistance must be taken into consideration. Also, remember that pressing either SW1 or SW2 will force the output low.

**Table 3.9** J9 1x4 Pin Connector for Digital Input

Pin	Name		Function	Specification
1	IO6	SW1	Digital Input from Neuron Chip	0 – 5 volts, 100K $\Omega$ impedance
2	GND		Ground	
3	IO7	SW2	Digital Input from Neuron Chips	0 – 5 volts, 100K $\Omega$ impedance
4	GND		Ground	

---

## J10 and J12 Power Supply Connectors

The J10 and J12 Power Supply connectors are connected in parallel and either can be used to provide standalone power to the Gizmo 4. These connectors are not used when the Gizmo 4 is powered from the J1 I/O Connector. J10 is a screw-terminal connector and J12 is a plug connector. The board contains a bridge rectifier allowing for positive and negative polarity to be connected to either post. Jumper JP12 allows for selection of power sources between J10 and J12 or the +5 volt source coming through the ribbon cable to J1.

**Table 3.10** J10 – J12 1x2-Pin Power Supply Connectors

Pin	Name	Function	Specification
1	ConnA	Input Power Source	9-30 Volt DC
2	ConnB	Input Power Source	9-30 Volts DC

*Note: If your prototype area draws more than 100ma, reduce the DC input voltage to 9 volts. This reduces the heat from the linear regulator if the Gizmo 4 is powered via J10 or J12.*

---

## J11 Network Connector

The J11 Network connector is the LONWORKS Network Connector for a TP/FT-10 or TP/FT-10F Control Module, if one is inserted in the Gizmo 4. J11 is routed from the J5 Control Module Network Connector. The Gizmo 4 does not contain any additional protection circuitry for the network.

**Table 3.11** J11 1x2-Pin Network Connector

Pin	Name	Function	Specification
1	Net A	Network A interface	Limited to the Transceiver Type
2	Net B	Network B interface	Limited to the Transceiver Type

---

## Gizmo 4 Jumpers

The four jumpers on the Gizmo 4 are described in the following sections.

---

### JP7 D/A 1 to A/D 1 Loop-Back Mode

The JP7 D/A 1 to A/D 1 Loop-Back Mode jumper enables a loop-back connection between the D/A 1 analog output and the A/D 1 analog input with optional delay. Insert a jumper between pins 1 and 2 to connect the Aout1 analog output to the Ain1 analog input with no delay. Insert a jumper between pins 2 and 3 to insert RC-delay circuit in the loop-back path using a 1M $\Omega$  series resistor and 220 $\mu$ F capacitor to ground.



---

## ***JP8 D/A 2 to A/D 2 Loop-Back Mode***

The JP8 D/A 1 to A/D 1 Loop-Back Mode jumper enables a loop-back connection between the D/A 2 analog output and the A/D 2 analog input with optional delay. Insert a jumper between pins 1 and 2 to connect the Aout1 analog output to the Ain1 analog input with no delay. Insert a jumper between pins 2 and 3 to insert RC-delay circuit in the loop-back path using a 1M $\Omega$  series resistor and 220 $\mu$ F capacitor to ground.

---

## ***JP12 Gizmo 4 Power Supply Source Selection***

The JP12 Gizmo 4 Power Supply Source Selection selects the Gizmo 4 power supply source selection. You can choose between the +5VDC input on J1 I/O Connector or the 9 – 30VDC input from the J10 and J12 Power Supply Connectors. Select the J1 input if you are attaching the Gizmo 4 to an LTM-10 Platform. Select the J10/J12 input if you have inserted a TP/FT-10 or TP/FT-10F Control Module.

---

## ***JP13 Piezo Speaker Select***

The JP 13 Piezo Speaker Select jumper enables the Gizmo 4 Piezo speaker.

---

## **Optional Gizmo 4 Connectors and Jumpers**

There are optional connectors and jumpers included on the Gizmo 4. These are not factory installed, but you can install headers or jumpers at these locations, or solder in wire connections for the jumpers.

---

## ***J13 PIC Programming Connector***

The J13 PIC Programming Connector is used to reprogram the PIC processor. Pin 1 of J13 is the programming control pin and is also the PIC reset input. If a jumper is installed in JP14, the PIC reset is also connected to the Neuron Chip reset pin. When the programmer is downloading or reading the PIC, this pin will go to 13 volts. To protect the Neuron Chip's reset pin from damage, the JP14 jumper must be removed when programming. If J13 is not populated you need to solder a 5-pin header in place to use the programming option.

**Table 3.12** J13 PIC Programming

<b>Pin</b>	<b>Name</b>	<b>Function</b>	<b>Specification</b>
1	VPP/Reset	+13VDC programming input or Active Low Reset	0 to 13VDC
2	VDD	+5 Supply Input	5 volts
3	GND	Ground	
4	B7/PGD	Programmer Data In or IOB7	
5	B6/PGC	Programmer Clock or IOB6	

---

## ***J16 Unused PIC I/O Connector***

The J16 Unused PIC I/O Connector provides access to PIC I/O pins that are not used by the I/O components on the Gizmo 4. You will have to reprogram the PIC to use these pins. Additional I/O and programming information is available from the PIC16F877 data sheet. If J16 is not populated you need to solder a 7-pin header in place to use the additional PIC I/O.

**Table 3.13** J16 PIC Programming Pin

<b>Pin</b>	<b>Name</b>	<b>Function</b>
1	GND	Ground
2	IOB1	
3	IOB0	
4	IOC2	
5	IOA4	
6	IOA2	
7	VDD	+5VDC

---

## ***JP10 PIC Stand Alone Mode Select***

The JP10 PIC Stand Alone Mode Select jumper enables the PIC processor to run stand alone. The jumper is connected to the PIC's IOB5 input pins. This jumper is used for factory testing of the Gizmo 4 board.

---

## ***JP11 D/A Stand Alone Mode Select***

The JP11 D/A Stand Alone Mode Select jumper enables the PIC processor to take control of the D/A chip select line. This jumper is used for factory testing of the Gizmo 4 board.

---

## ***JP14 PIC Reset Enable***

The JP14 PIC Reset Enable jumper connects the SW4 Reset button to the PIC processor reset pin. The SW4 button is also connected to the Neuron Chip reset pin. The JP14 jumper must be removed when you are programming the PIC. A 13VDC input is applied to the PIC reset pin when programming. This 13VDC input will damage the Neuron Chip. To prevent this, be sure to disconnect the JP14 jumper when programming the PIC processor. JP14 causes PIC RTC to reset.

---

## ***JP17 SPI Stand Alone Mode Select***

The JP17 SPI Stand Alone Mode Select jumper routes the SPI signal to either the PIC or D/A converter. This jumper is used for factory testing of the Gizmo 4 board.

# 4

## Gizmo 4 Software

This chapter describes the Gizmo 4 library. This library provides utility functions for using the I/O components on the Gizmo 4 board. It is included with the NodeBuilder Development Tool (starting with Release 3), and is also available in the Echelon Developer's Toolbox at [www.echelon.com/toolbox](http://www.echelon.com/toolbox). The NodeBuilder Development Tool includes additional programming examples for the Gizmo 4 that use the Gizmo 4 library.

---

## Getting Started

The Gizmo 4 library consists of source code for functions that access the I/O components on the Gizmo 4 board. The library is provided as a C header file that you must include in the source code for any Neuron C application that uses the library. You will compile this file, called `gizmo4.h`, as part of your application using the NodeBuilder or LonBuilder software.

To use the Gizmo 4 library in a Neuron C application using the NodeBuilder tool, follow these steps.

1. Create a new NodeBuilder device template.
2. Open the template's settings and add the following to the include search path field:

```
$LonWorks$\NodeBuilder\Gizmo4
```

This is the default location. If it was changed at installation, specify the exact location of the Gizmo 4 library files.

You also can specify the path in the NodeBuilder Project Preferences window. Select **Use project include path** in the device template to use the project path.

If you have additional entries in the include search path, separate the multiple entries with a semicolon.

3. Open the Neuron C code for your application and insert the following statement:

```
#include "gizmo4.h"
```

4. Find the when (reset) task, or create one. Add the following statement to the task:

```
GizmoReset();
```

5. To clear the display on start-up add the following statement after the `GizmoReset()` call:

```
GizmoClearDisplay();
```

---

## Macros in Gizmo4.h

The following macros are defined by the Gizmo 4 library:

```
GIZMO_DISPLAY_*
GIZMOPIC_CS_DELAY
USE_FREQUENCY
USE_QUADRATURE
```

These macros are defined in the following sections.

---

### *GIZMO\_DISPLAY\_\**

This macro displays dimension and related properties, as specified.

```
#define GIZMO_DISPLAY_ROWS          4
#define GIZMO_DISPLAY_COLUMNS      20
#define GIZMO_DISPLAY_TABWIDTH     5
#define GIZMO_DISPLAY_SIZE         (GIZMO_DISPLAY_ROWS *
                                   GIZMO_DISPLAY_COLUMNS)
```

---

### *GIZMOPIC\_CS\_DELAY*

The PIC processor on the Gizmo 4 requires a delay between activation of the chip select line and the transmission of the first data bit. The `GIZMOPIC_CS_DELAY` macro is used by the library functions to implement this pause.

By default, the Gizmo 4 library does not implement a delay because a delay is not required when running on a Neuron Chip with an input clock of 10MHz or less. If your target hardware platform needs the delay, remove the `//` from the following statement to enable the delay.

```
#define GIZMOPIC_CS_DELAY //scaled_delay(25ul)
```

---

### *USE\_FREQUENCY*

The `USE_FREQUENCY` macro allows you enable the piezo speaker in order to play simple tunes.

LED1 and the piezo speaker share the IO0 I/O pin. Because the speaker requires a frequency output, the LED also must be driven by a frequency output when using the speaker. LED1 will always be on when using frequency output because the LED is driven with an active low signal and the frequency output model always returns to low when shut off.

To enable use of LED1, the default is to use the pulsewidth model at 610 Hz. This allows the piezo speaker to make noise at this frequency, with a duty cycle of 50%.

This also allows the LED to be turned on (pulsewidth at 0%) and off (pulsewidth at 100%). However, you cannot play tunes on the speaker.

By default, `USE_FREQUENCY` is not defined. This gives you full control over LED1, but you cannot play tunes on the piezo speaker.

To enable tunes and disable use of LED1, uncomment the following statement in `Gizmo4.h`:

```
#define USE_FREQUENCY
```

---

## *USE\_QUADRATURE*

The second LED uses the IO1 pin and uses a Neuron C bit output. This causes LED2 to be brighter than LED1 which uses a frequency or pulsewidth output model. If you want LED2 to be the same brightness as LED1, you can change LED2 to use the same pulsewidth or frequency model as LED1. However, you cannot use a pulsewidth output on LED2 if you are using the quadrature input on IO4 and IO5 since both types of I/O require the same timer/counter.

By default, a `USE_QUADRATURE` macro is defined that enables quadrature input from the shaft encoder, and enables bit output to LED2. This will, by default, make LED2 slightly brighter than LED1.

To disable quadrature input and enable either pulsewidth or frequency output to LED2, comment out the following macro in `Gizmo4.h`:

```
#define USE_QUADRATURE
```

Use the `GizmoSetLed()` function to control the LEDs since it takes into account the output model defined for the selected LED.

---

## Types

The following types are defined by the Gizmo 4 library if `USE_FREQUENCY` is defined:

```
tGizmoNote  
tGizmoTone  
tGizmoToneLength
```

These types are defined in the following sections.

---

### *tGizmoNote*

The `tGizmoNote` type is used to specify a note to the `GizmoPlayNote()` function. A note consists of tone and length values defined in a `tGizmoNote` structure. See `GizmoPlayNote()` for more information.

---

## *tGizmoTone*

The `tGizmoTone` type is used to specify a tone (a note of sound) for the piezo speaker, or to specify a control value for the `GizmoPlayTune()` function. The following macros can be used to assign any of the tones in an octave to a `tGizmoTone` value:

```
#   define GizmoToneC      9579ul
#   define GizmoToneCs    9058ul
#   define GizmoToneD     8532ul
#   define GizmoToneDs    8065ul
#   define GizmoToneE     7599ul
#   define GizmoToneF     7184ul
#   define GizmoToneFs    6775ul
#   define GizmoToneG     6394ul
#   define GizmoToneGs    6039ul
#   define GizmoToneA     5682ul
#   define GizmoToneAs    5376ul
#   define GizmoToneB     5071ul
```

The constants are defined assuming a clock modifier of 0. You can change the octave by multiplying or dividing the values by 2.

The following control values are used to control the `GizmoPlayTune()` function:

```
#   define GizmoTonePause 0ul    // an important 'tone', too
#   define GizmoToneEnd   1ul    // end of melody marker
#   define GizmoToneMarker 2ul   // use this to set a repeat
                                   // marker in the melody
#   define GizmoToneRewind 3ul   // go to the last known
                                   // GizmoToneMarker - this causes
                                   // an endless loop
#   define GizmoToneDaCapo 4ul   // go to the last known marker
                                   // position, but only do so once
```

---

## *tGizmoToneLength*

The `tGizmoToneLength` type is used to define the duration of a tone in hundreds of a second. This is an enumeration type that defines the following enumeration constants:

```
GizmoNoteEight = 2,
GizmoNoteQuarter = 4,
GizmoNoteHalf = 8,
GizmoNoteFull = 16
```

---

## Functions

The following functions are used by the Gizmo 4.

```
GizmoBuzz()  
GizmoClearDisplay()  
GizmoDefineCharacter()  
GizmoDisplayCharacter()  
GizmoDisplayChar()  
GizmoDisplayDate()  
GizmoDisplayFormattedNumber()  
GizmoDisplayString()  
GizmoDisplayTemperature()  
GizmoDisplayTime()  
GizmoGetTune()  
GizmoPlayTune()  
GizmoReadAnalog()  
GizmoReadTemperature()  
GizmoReset()  
GizmoRTC()  
GizmoSetLed()  
GizmoSetTime()  
GizmoWriteAnalog()
```

---

### *GizmoBuzz()*

```
void GizmoBuzz(boolean bOn);
```

**bOn**                Turns the buzzer on if TRUE..

The `GizmoBuzz()` function controls the piezo speaker. The frequency of the sound produced will be about 610Hz if the pulsewidth model is in use, or 840 Hz (`GizmoToneA`) if the frequency model is in use. See also `GizmoPlayNote()`.

---

### *GizmoClearDisplay()*

```
void GizmoClearDisplay(void);
```

The `GizmoClearDisplay()` clears the LCD display. This function does not disable automatic time, data, or temperature display. You must disable those display functions prior to calling `GizmoClearDisplay()` to completely clear the display.

---

### *GizmoDefineCharacter()*

```
void GizmoDefineCharacter(unsigned code,  
                          unsigned char *const pData);
```

**code**                Character to define, between 0 and 7..

**pData**               Pointer to an array of 8 bytes defining the character.



The `GizmoDefineCharacter()` function defines up to eight (8) graphic characters, using ASCII codes 0x00 to 0x07. Each call to `GizmoDefineCharacter()` redefines one character, indicated by code. The character generator uses a 5x8 dot matrix font where the first byte shown is the one at the top right-hand corner of the display. The least significant bit is shown at the right edge of the display. For textual data, the lowest line for each character should remain blank (0x00) or reserved for the tail of a character. Semigraphical characters are free to use the entire area. For example, character matrix data { 0x1F, 0x1F, 0x1F, 0x1F, 0x1F, 0x1F, 0x1F, 0x1F } represents a 'black' rectangle, and matrix data { 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01 } produces a character with a thin line at the rightmost edge. A re-defined character with code 0x00 can not be displayed using the `GizmoDisplayString()` function, use `GizmoDisplayCharacter()` instead.

---

### *GizmoDisplayChar()*

```
void GizmoDisplayChar(unsigned xPos, unsigned yPos, char cChar);
```

xPos            Column number, between 0 and 19.

yPos            Row number, between 0 and 3.

cChar           The character to be displayed.

The `GizmoDisplayChar()` function displays a single character at the position specified by `xPos`, `yPos`. See Appendix B for a list of the characters that can be displayed with the `GizmoDisplayChar()` function. You can also create up to eight custom characters using the `GizmoDefineChar()` function.

---

### *GizmoDisplayDate()*

```
void GizmoDisplayDate(unsigned xPos, unsigned yPos, boolean  
                          bEnable);
```

xPos            Column number, between 0 and 19.

yPos            Row number, between 0 and 3.

bEnable        Enables automatic display.

The `GizmoDisplayDate()` functions starts or stops continuous display of the current date. To enable continuous display, set `bEnable` to `TRUE` and specify a starting position using `xPos` and `yPos`. There can only be one field for this display, you cannot display the date in two locations simultaneously. To disable continuous display, set `bEnable` to `FALSE`. The `xPos` and `yPos` arguments are ignored when disabling the display.

---

## *GizmoDisplayFormattedNumber()*

```
Void GizmoDisplayFormattedNumber(unsigned xPos, unsigned yPos,  
                                long lNumber, unsigned uBase,  
                                unsigned uField, unsigned  
                                uPoint char cNeutral);
```

*(Note: the lNumber entry is a lower case “L” – it is not a number one.)*

xPos	Column number, between 0 and 19.
yPos	Row number, between 0 and 3.
lNumber	The number to be displayed. (This is a lowercase “L” and not a one.)
uBase	The displayed number base, from base 2 to 16.
uField	The total number of characters to display
uPoint	The position of the decimal point within the field, from the left (0 for none)
cNeutral	Fill character for unused characters.

The `GizmoDisplayFormattedNumber()` function displays `lNumber` at the position specified by `xPos`, `yPos`. The formatting is for the following parameters:

The number is positioned on the rightmost edge of the field, the field is being filled up with ‘`cNeutral`’ to the left, if needed.

The display number is considered unsigned, and no decimal point will be inserted if the base is different than 10. A series of asterisks is shown if the field is too small for the formatted data. Note a prefix or postfix to indicate the base is not inserted.

---

## *GizmoDisplayNumber()*

```
void GizmoDisplayNumber(unsigned xPos, unsigned yPos, long  
                        lNumber);
```

xPos	Column number, between 0 and 19.
yPos	Row number, between 0 and 3.
lNumber	The number to be displayed.

The `GizmoDisplayNumber()` function displays `lNumber` at the position specified by `xPos`, `yPos`, using decimal formatting. See also `GizmoDisplayFormattedNumber()`.

---

## *GizmoDisplayString()*

```
void GizmoDisplayString (unsigned xPos, unsigned yPos, const
                        char *pString);
```

xPos            Column number, between 0 and 19.  
yPos            Row number, between 0 and 3.  
pString        Pointer to a string to be displayed.

The `GizmoDisplayString()` function displays the zero-terminated string specified by `pString` at the position specified by `xPos`, `yPos`, using decimal formatting. See Appendix B for a list of the characters that can be displayed with the `GizmoDisplayString()` function. `GizmoDisplayString()` handles the following non-printable characters:

`\t`     tab (see `GizmoDisplayTabwidth`)  
`\r`     go to beginning of current line  
`\n`     newline

For example, to start a new line from the leftmost column, enter the following sequence `\n\r` or `\r\n`.

---

## *GizmoDisplayTemperature()*

```
void GizmoDisplayTemperature(unsigned xPos, unsigned yPos,
                             boolean bEnable);
```

xPos            Column number, between 0 and 19.  
yPos            Row number, between 0 and 3.  
bEnable        Enables automatic display.

The `GizmoDisplayTemperature()` functions starts or stops continuous display of the current temperature. To enable continuous display, set `bEnable` to `TRUE` and specify a starting position using `xPos` and `yPos`. There can only be one field for this display, you cannot display the temperature in two locations simultaneously. To disable continuous display, set `bEnable` to `FALSE`. The `xPos` and `yPos` arguments are ignored when disabling the display.

The temperature is always displayed in Fahrenheit. Use the `GizmoReadTemperature()` and `GizmoDisplay*()` functions for alternative units and formatting.

---

## *GizmoDisplayTime()*

```
void GizmoDisplayTime(unsigned xPos, unsigned yPos, boolean
                      bEnable);
```

xPos            Column number, between 0 and 19.

yPos            Row number, between 0 and 3.

bEnable        Enables automatic display.

The `GizmoDisplayTime()` functions starts or stops continuous display of the current time. To enable continuous display, set `bEnable` to `TRUE` and specify a starting position using `xPos` and `yPos`. There can only be one field for this display, you cannot display the time in two locations simultaneously. To disable continuous display, set `bEnable` to `FALSE`. The `xPos` and `yPos` arguments are ignored when disabling the display.

---

## *GizmoGetTime()*

```
void GizmoGetTime (SNVT_time_stamp *pTime);
```

pTime            Pointer to a time value.

The `GizmoGetTime()` function reads the current time from the Gizmo 4.

---

## *GizmoPlayTune()*

```
void GizmoPlayTune(const tGizmoNote *pNote);
```

pNote            A pointer to an array of notes. See the definitions of `tGizmoNote`, `tGizmoTone`, `tGizmoToneLength` in the *Types* section.

The `GizmoPlayTune()` function plays a composition consisting of a series of notes, sounds, and durations. The `USE_FREQUENCY` macro must be defined. Be sure to include a `GizmoToneEnd` or `GizmoToneRewind` marker in your melody. The entire tune memory must be valid for the duration of the tune. Because there is background processing required to play your tune, the `GizmoPlayTune()` function cannot be used from within a `when(offline)` task. You can use this function from within a `when(reset)` or `when(wink)` task, but only if you have configured the device and it is online.

---

## ***GizmoReadAnalog()***

```
unsigned long GizmoReadAnalog(unsigned channel);
```

channel        The A/D converter channel; either 0 or 1.

The `GizmoReadAnalog()` function reads analog data. The function returns a value of 0..1000 (for 100%).

---

## ***GizmoReadTemperature()***

```
signed long GizmoReadTemperature(Boolean bFahrenheit,  
                                  Boolean bUseFixed);
```

bFahrenheit    Returns a Fahrenheit temperature if TRUE; Celcius temperature if FALSE..

bUseFixed      Returns a scaled fixed point value if TRUE; or integer temperature if FALSE.

The `GizmoReadTemperature()` function reads the current temperature. The PIC controller returns the temperature with a resolution of 0.5 C/F. This function can return two possible data formats, an integer temperature (in either Fahrenheit or Celcius), or a fixed point value with a precision of 0.5 C/F. The fixed point format is chosen by the second, Boolean, parameter (`bUseFixed`), otherwise the value is rounded to the nearest integer. Note that `GizmoReadTemperature(FALSE, TRUE)` returns data that is compatible with `SNVT_temp`.

---

## ***GizmoReset()***

```
void GizmoReset(void);
```

The `GizmoReset()` function initialize the Gizmo 4 library and the Gizmo 4 I/O interface. This function should be called from within your when task (reset) .

---

## ***GizmoRTC()***

```
void GizmoRtc(Boolean bStart);
```

bStart        Starts the real-time clock if TRUE.

The `GizmoRTC()` function starts or stops the real-time clock.

---

## ***GizmoSetLed()***

```
void GizmoSetLed(unsigned channel, boolean bOn);
```

channel        The LED number; 0 for LED1 or 1.for LED2

bOn            The value to set. TRUE turns the LED on.

The `GizmoSetLed()` function turns LEDs on or off. If a tune is playing when you attempt to turn the LED off, the function will not complete and the frequency output will be stopped.

---

## *GizmoSetTime()*

```
void GizmoSetTime(const SNVT_time_stamp *pTime);
```

`pTime`            Pointer to a time value.

The `GizmoSetTime()` sets the clock in the Gizmo 4 to the time specified by `pTime`.

---

## *GizmoWriteAnalog()*

```
void GizmoWriteAnalog(unsigned channel, unsigned long Data);
```

`channel`            The A/D converter channel; either 0 or 1.

`Data`              The value to write, between 0 and 1000 (for 0..100%).

The `GizmoWriteAnalog()` function writes a data value to the D/A converter, producing an analog output.

---

## **I/O Objects**

This section describes the Gizmo 4 library I/O objects..

---

### *LEDs and Quadrature Input*

There are two LEDs: LED1 and LED2. They may be declared as pulsewidth, frequency, or bit outputs depending on the settings specified by the `USE_FREQUENCY` and `USE_QUADRATURE` macros. There is an optional quadrature input for the rotary shaft encoder. These are defined as follows:

```
#ifdef USE_FREQUENCY
    IO_0 output frequency clock(0) ioPiezzo = 1;
#else
    IO_0 output pulsewidth short invert clock(5) ioPiezzo = 1;
#endif      // use_frequency

#define ioLed1 ioPiezzo

#ifdef USE_QUADRATURE
    IO_1 output bit ioLed2 = 1;

    // quadrature input
    IO_4 input  quadrature  ioWheel;
```

```

#else
#   ifdef USE_FREQUENCY
        IO_1 output frequency clock(0) ioLed2 = 1;
#   else
        IO_1 output pulsewidth short invert clock(5) ioLed2 = 1;
#   endif // use_frequency
#endif // use_quadrature

```

---

## Neurowire

Neurowire I/O is used for the A/D converter, D/A converter, real-time clock, and display drivers.

Neurowire requires pins IO8 through IO10, plus chip selects on IO2 and IO3.

```

// Neurowire (SPI) declarations
IO_8 neurowire master select(IO_2) ioPIC;
IO_2 output bit ioPICCS = 1; // ~CS for PIC, initially unselected

IO_8 neurowire master select (IO_3) ioDAC;
IO_3 output bit ioDACCS = 1; // ~CS for DAC, initially unselected

```

---

## Push Buttons

The two push buttons are connected to IO6 and IO7. SW1 connects to IO6 and SW2 connects to IO7.

```

// pushbuttons:
IO_6 input bit ioButton1; // the one next to LED1
IO_7 input bit ioButton2; // the one next to LED2

```

---

## Error Management

The Gizmo 4 library returns errors using the `GizmoLastError` global variable which is declared with the `tGizmoError` type. You can watch this variable in the NodeBuilder debugger.

Error reporting is enabled by the `_DEBUG` macro, which is defined for NodeBuilder development targets. If you create a NodeBuilder release target, the `_DEBUG` macro is not defined, and the Gizmo 4 error reporting mechanism is disabled.

Use the `GIZMOSETERROR()` macro to set an error code, and use `GIZMOSETERROR(geNone)` to reset the `GizmoLastError` variable once an error condition has been resolved. If your code evaluates `GizmoLastError`, you should compile this code conditionally based on the existence of the `_DEBUG` macro.

See also `GzCommon.nc` for a definition of the `GIZMOASSERT` macros, which are used by the `Gizmo*()` functions for parameter validation in the debug targets.

### **tGizmoError:**

```
typedef enum
{
    geNone          =      0,          // invalid channel number, etc
    geBadParameter,          // invalid channel number, etc
    geBadPointer,          // NULL pointer given but not acceptable
    geBadCoordinates,      // x/y display positioning values invalid
    geClipped,             // data was clipped at display edge
    geOutOfRange,         // data was out of range (WriteAnalog?)
} tGizmoError;
```

### **tGizmoLastError:**

```
#ifdef _DEBUG
    extern      tGizmoError GizmoLastError;
#endif // def. _debug
```

### **GIZMOSETERROR:**

```
#ifdef _DEBUG
#   define      GIZMOSETERROR(code)      GizmoLastError = code
#else
#   define      GIZMOSETERROR(code)
#endif // def. _debug
```

---

## **Communicating Directly with the Gizmo 4**

You can communicate directly with the Gizmo 4 library. This is typically not required if you are using the Gizmo 4 library. This section describes the hardware interface to the Gizmo 4 if you are maintaining or enhancing the Gizmo 4 library.

The Gizmo 4 contains two types of I/O components for use with the Neuron Chip; the I/O components directly available to the Neuron Chip's I/O pins, and the I/O components available via the PIC processor on the Gizmo 4.

The I/O components directly wired to the Neuron Chip include the following:

- Two push buttons
- Two LEDs
- Quadrature encoder
- Piezzo speaker
- One D/A converter with two D/A Channels

The I/Os available through the PIC include:

- 4x20 character LCD
- One A/D converter with two A/D channels



- One temperature A/D channel
- One real-time clock

---

## *Communicating with the I/O Wired to the Neuron Chip*

The following code is provided for example purposes only. You may need to alter variables or names in your own environment.

### **Push Buttons**

The push buttons are connected directly to IO6 and IO7 of the Neuron Chip. They are low asserting switches.

```
IO_6 input bit switch1;

when(io_changes(switch1))
{
    if (input_value == 0)
        // Switch released
    else
        // Switch pushed
}
```

### **LEDs**

The LEDs are low asserting and are connected to IO0 and IO1.

```
IO0 output bit LED1;

io_out(LED1, 0); //On
io_out(LED1, 1); //Off
```

### **Quadrature Encoder**

The quadrature encoder generates pairs of pulses on two I/O lines. This happens as it is turned. The quadrature input determines positive and negative direction based on the ordering of the pulses. The quadrature encoder is on IO4 and IO5.

```
IO_4 input quadrature ioQuad;

when(io_update_occurs(ioQuad))
{
    static int position;
    position += input_value;
}
```

## Piezo Speaker

The piezo speaker shares its I/O connections with LED1. It is connected to IO0.

```
IO0 output frequency ioBuzz;  
...  
io_out(ioBuzz, 2500); //Output a 1KHz sound
```

## I/O Controlled by the PIC

The PIC directly handles control of the I/O that is connected to it. It provides an SPI interface with which the Neuron Chip can read or write to these I/O components. Within the PIC is a series of 128 registers corresponding to the various I/O components connected to the PIC.

To read the value of a register, use the Neurowire IO model to exchange a series of bits with the PIC. The Neurowire interface uses IO8 through IO10 of the Neuron Chip. The chip-select line for the PIC is connected to IO2.

The first bit (MSB of the register byte) tells the PIC that you are reading or writing to the register. The following 7 bits specify the register to read or write. The remaining bits are a buffer to hold the data that is being read or written.

**Example 1:** Read 16-bits of data from register 0x05 (temperature in degrees Fahrenheit):

```
IO8 neurowire select(IO_2) ioPIC;

byte buff[3];
long temperature;

buff[0] = 0x05 | 0x80; //0x05 is the register, 0x80 sets the read bit

io_in(ioPIC, buff, 24);

temperature = buff[1] << 8; //bits 8-9 of the temp
temperature |= buff[2]; //bits 0-7 of the temp
```

**Example 2:** Write a line of text to the LCD Display:

```
byte buff[21];

buff[0] = 0x10; // Register 0x10 is the first char of the first line of
the display

memcpy(buff[1], "====GIZMO4====", 20); //Copy the string into the
buffer

io_in(ioPIC, buff, 168); //io_in and io_out perform the same for the
Neurowire IO model
```

## Register Tables

Table 4.1 includes a list of registers that can be written in the PIC.

**Table 4.1** Register Descriptions to Write in the PIC

ADDRESS (HEX)	REGISTER DESCRIPTION
01	Control Register
0B	Analog Out A
0C	Analog Out B
0D	Enable User Graphic Character Mode
0E	Write User Character
10 – 23	LCD Line 1 Character 1-20 (ASCII codes)
24 – 37	LCD Line 2 Character 1-20 (ASCII codes)
38 – 4B	LCD Line 3 Character 1-20 (ASCII codes)
4C – 5F	LCD Line 4 Character 1-20 (ASCII codes)
60	Time Auto Position
61	Date Auto Position
62	Temperature Auto Position
63	Analog In A Auto Position
64	Analog In B Auto Position

**Table 4.1** (Continued)

ADDRESS (HEX)	REGISTER DESCRIPTION
65	Analog Out A Auto Position
66	Analog Out B Auto Position
68	Seconds register
69	Minutes register
6A	Hours register
6B	Days register
6C	Weeks register
6D	Years LSB register
6E	Years MSB register

Table 4.2 includes a list of registers that can be read in the PIC.

**Table 4.2** Register Descriptions to Read in the PIC

ADDRESS (HEX)	REGISTER DESCRIPTION
81	Control Register
83	TemperatureC LSB (bits 0-7)
84	TemperatureC MSB (bits 8-9 => 0-1)
85	TemperatureF LSB (bits 0-7)
86	TemperatureF MSB (bits 8-9 => 0-1)
87	Analog In A LSB (bits 0-7)
88	Analog In A MSB (bits 8-9 => 0-1)
89	Analog In B LSB (bits 8-9 => 0-1)
8A	Analog In B MSB (bits 8-9 => 0-1)
90 – A3	LCD Line 1 Character 1-20 (ASCII codes)
A4 – B7	LCD Line 2 Character 1-20 (ASCII codes)
B8 – CB	LCD Line 3 Character 1-20 (ASCII codes)
CC – DF	LCD Line 4 Character 1-20 (ASCII codes)
E8	Seconds register
E9	Minutes register
EA	Hours register
EB	Days register
EC	Weeks register
ED	Years LSB register
EE	Years MSB register

Table 4.3 includes a list of registers that allow certain functions within the PIC to be enabled or disabled.

**Table 4.3** Registers to Allow PIC Functions to be Enabled/Disabled

ADDRESS (HEX)	REGISTER DESCRIPTION
81	Control Register
83	TemperatureC LSB (bits 0-7)
84	TemperatureC MSB (bits 8-9 => 0-1)
85	TemperatureF LSB (bits 0-7)
86	TemperatureF MSB (bits 8-9 => 0-1)
87	Analog In A LSB (bits 0-7)
88	Analog In A MSB (bits 8-9 => 0-1)
89	Analog In B LSB (bits 8-9 => 0-1)

**Table 4.3** (Continued)

ADDRESS (HEX)	REGISTER DESCRIPTION
8A	Analog In B MSB (bits 8-9 => 0-1)
90 – A3	LCD Line 1 Character 1-20 (ASCII codes)
A4 – B7	LCD Line 2 Character 1-20 (ASCII codes)
B8 – CB	LCD Line 3 Character 1-20 (ASCII codes)
CC – DF	LCD Line 4 Character 1-20 (ASCII codes)
E8	Seconds register
E9	Minutes register
EA	Hours register
EB	Days register
EC	Weeks register
ED	Years LSB register
EE	Years MSB register

The following table provides notes about values being read or written from the PIC.

**Table 4.4** Values Read or Written from the PIC

Control Register	Always set bit 0 to high or the PIC will halt and require the Gizmo's power to be cycled.
Analog Outputs	Analog output range is ~0-5VDC, scaled from values of 0-256, each value step is ~19-20mVdc.
Analog Inputs	Analog input range is ~0-5VDC, scaled from values of 0-1024, each value step is ~5mVdc.
LCD registers	The LCD displays standard ASCII characters and allows for user-defined characters to be added (see Appendix A).
RTC	<p>The RTC's base time is 00:00:00, January 1, 2001. All dates are in reference to that point. For example, 10/31/2001 at 5:04:28pm would be:</p> <pre> Year    = 0; //still 2001 Months  = 10; Days    = 31; Hours   = 17; Minutes = 4; Seconds = 28; </pre>
AutoPosition	<p>The AutoPosition registers are for instructing the PIC to automatically display the given I/O value on the LCD display. Following the register in the buffer, one or two bytes should determine where to display the data.</p> <p>Byte 1: 0 to turn off display, 1-4 for the row to display on  Byte 2: the character (1-20) where the display will start</p> <p>The second bytes is not specified when turning the display off.</p>



# Appendix A

## Creating a Custom Character

This appendix provides you with information to create custom characters. This information assumes that you are not using the functions in the Gizmo 4 (`GizmoDefineCharacter`).

---

## Custom Character Example

With the Gizmo 4 library, you can build up to eight new graphical characters and store them in the PIC with a display address from 00 to 07. The graphical display is in a 5 x 7 dot matrix with a bottom row available for an underline or cursor. To create the new character, build it in an array. The following example shows the creation of the letter "A."

```
// Test for User Character Graphic Mode.
// User can develop up to 8 user characters
unsigned char userchar0[8] =          // Assigned address 0
    {0x0E,                            // ....XXX.
    0x11,                              // ...X...X
    0x11,                              // ...X...X
    0x11,                              // ...X...X
    0x1F,                              // ...XXXXX
    0x11,                              // ...X...X
    0x11,                              // ...X...X
    0x00                              // ..... (underline)
    };
```

To send the character to the PIC, use the following code.

```
int i;
byte buff[2];
int charnum;

charnum = 0;

for( i = 0; i < 8; i++)
{
    buff[0] = 0x0D;
    buff[1] = (charnum << 3) ; //Tell it which char to edit
    buff[1] += i; // Tell it which line we are editing
    io_out( ioPIC, buff, 16); //send it to the PIC

    buff[0] = 0x0E;
    buff[1] = userchar0[i]; //Send the raw data to the PIC
    io_out( ioPIC, buff, 16);
}
```



# **Appendix B**

## **Gizmo 4 Character Set**

This appendix lists the standard character set included with the Gizmo 4.

# Character Set

The following table lists the characters that you can display using the GizmoDisplayChar() and GizmoDisplayString() functions. You can redefine the first eight characters using the GizmoDefineChar() function.

CHARACTER FONT CODES (5 x 7 DOT MATRIX)

Upper 4 Bit Hexadecimal

Upper 4 bits Lower 4 bits	0000 (0)	0010 (2)	0011 (3)	0100 (4)	0101 (5)	0110 (6)	0111 (7)	1010 (A)	1011 (B)	1100 (C)	1101 (D)	1110 (E)	1111 (F)
x x x x 0000 (0)	CG RAM (1)		0	a	P	`	P	—	タ	三	α	p	
x x x x 0001 (1)	(2)	!	1	A	Q	a	q	μ	ア	チ	△	â	q
x x x x 0010 (2)	(3)	"	2	B	R	b	r	Γ	イ	ウ	×	β	θ
x x x x 0011 (3)	(4)	#	3	C	S	c	s	┘	ウ	テ	ε	ε	ω
x x x x 0100 (4)	(5)	\$	4	D	T	d	t	、	エ	ト	ト	μ	α
x x x x 0101 (5)	(6)	%	5	E	U	e	u	・	オ	ナ	1	ε	ü
x x x x 0110 (6)	(7)	&	6	F	V	f	v	ヲ	カ	ニ	ヨ	ρ	Σ
x x x x 0111 (7)	(8)	'	7	G	W	g	w	フ	チ	ヌ	ウ	g	π
x x x x 1000 (8)	(1)	<	8	H	X	h	x	イ	ウ	ホ	リ	フ	Σ
x x x x 1001 (9)	(2)	>	9	I	Y	i	y	ヨ	ク	リ	ル	´	y
x x x x 1010 (A)	(3)	*	:	J	Z	j	z	エ	コ	ン	レ	j	¥
x x x x 1011 (B)	(4)	+	:	K	C	k	c	<	オ	サ	ヒ	□	¥
x x x x 1100 (C)	(5)	,	<	L	¥	l	l	ト	シ	フ	ワ	φ	円
x x x x 1101 (D)	(6)	—	=	M	I	m	>	ユ	ズ	へ	ン	±	÷
x x x x 1110 (E)	(7)	,	>	N	^	n	÷	ヨ	セ	ホ	´	ñ	
x x x x 1111 (F)	(8)	/	?	O	_	o	+	ウ	リ	マ	°	ö	■

Lower 4 Bit Hexadecimal