Creating Neuron® C Applications with the Gizmo 3

January 1997 **LONWORKS® Engineering Bulletin**

Introduction

The Neuron C programming language provides 39 I/O objects that support a wide variety of I/O devices. These I/O objects range from simple bit I/O to more complex timer counter objects which precisely generate and measure various square wave signals. The I/O objects are implemented through 11 I/O pins using firmware and hardware support on the Neuron Chip.

The Gizmo 3 enables rapid development of prototype applications that demonstrate several of these I/O objects. The I/O devices supported by the Gizmo kit are used to build examples for distributed sense and control problems. Quick prototypes and feasibility studies are possible without developing any I/O hardware.

This document describes how use the Gizmo 3 and the example application programs provided with it. Multi-node application examples are included to help the user get started.

Getting Started

The Gizmo 3 is included with some versions of the LonBuilder® development system and the NodeBuilder® development tool. Using the Gizmo 3 as described in this bulletin requires the following components:

- Gizmo 3 (Motorola MC143206EVK) - Contains 11 Neuron Chip compatible physical I/O devices.
- Application Interface Board - Provides access to the I/O pins of the Neuron Chip on a LonBuilder processor board through a DB25 connector. This board works with LonBuilder Neuron Emulators.
- Application I/O Interface Cable - Connects the application interface board to the Gizmo 3.
- Programming Examples - These are included with the LonBuilder and NodeBuilder tools, and are also available on the World-Wide Web at URL: `http://www.lonworks.echelon.com/toolbox` under Neuron C Examples.

This archive contains sample source code to demonstrate a variety of I/O objects supported by Neuron C and the Gizmo 3. The examples are designed to support several different distributed sense and control applications.

Hardware Installation

Connect the Gizmo 3 to a LonBuilder Neuron emulator, or to the LTM-10 LonTalk Node, using the supplied ribbon cable.

Software Installation

The Gizmo 3 programming examples provide Neuron C source code for use in example applications (see last section of this document). This source code is compiled and loaded into nodes using the LonBuilder or NodeBuilder software. To install and load the example applications, follow the steps in the appropriate User's Guide.

Gizmo 3 I/O Hardware

The Gizmo 3 is used to prototype LONWORKS applications. It includes the physical I/O devices listed in table 1.

Table 1. Gizmo 3 I/O Devices

Type	***Device***	***Function***
Analog	Analog-to-Digital	4-channel 10-bit MC14053. Neurowire device using IO1 as the select line.
	Digital-to-Analog	4-channel 6-bit MC14411. Neurowire device using IO3 as the select line.
Sound	Piezo-electric transducer	Operated by square wave output from IO0
Time	Real Time Clock	MC68HC68T1. Neurowire device using IO6 as the select line.
Temperature	Temperature Sensor	Connected to channel 0 of the A/D converter
Switches	Left Push-button	When pressed, drives the IO7 input low.
	Right Push-button	When pressed, drives the IO3 input low.
	Quadrature Control	When rotated, drives the quadrature shaft encoder input on IO4 and IO5.
LED Indicators	IO1 Bit Output LED (Red)	Indicates the state of the IO1 output. The LED is on when IO1 is low.
	IO2 Bit Output LED (Green)	Indicates the state of the IO2 output. The LED is on when IO2 is low.
	LED Display	4-digit, 7 segment display controlled by an MC14489 Multi-Character LED Display Driver. Neurowire device using IO2 as the select line.

The block diagram of the Gizmo 3 is shown in figure 1.

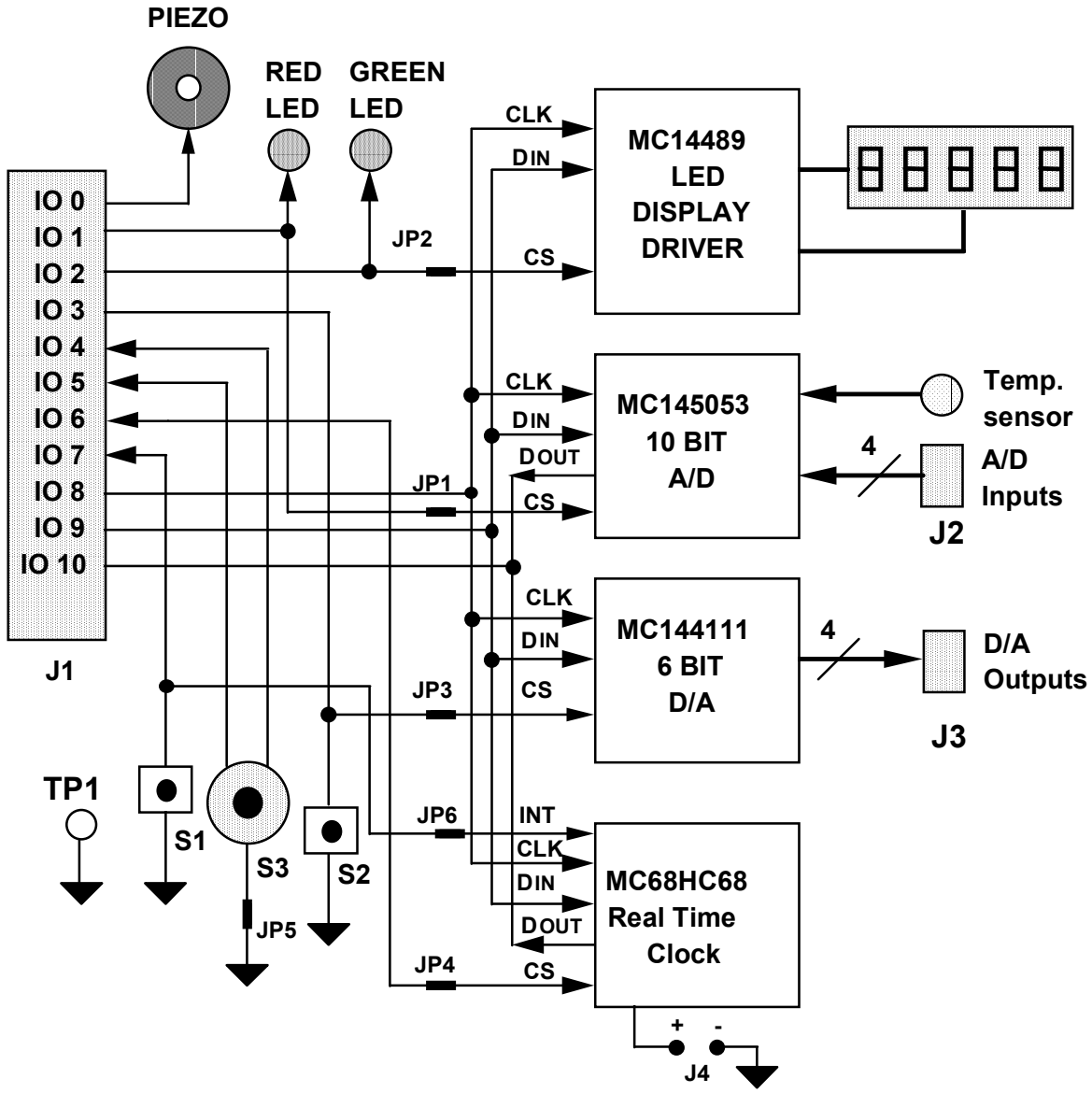


Figure 1. Gizmo 3 Block Diagram

Three of the I/O pins are used both for Neurowire device selection as well as for direct I/O devices. If the application uses the direct I/O devices, the associated Neurowire device may exhibit undesirable behavior. Jumpers are provided to disconnect the chip select pin for each of the affected Neurowire devices, as shown in table 2.

Table 2. Gizmo 3 Neurowire devices

I/O Pin	*Selected device*	*Direct I/O Function*	*Jumper*
IO1	A/D converter	Red LED	JP1
IO2	7-segment display	Green LED	JP2
IO3	D/A converter	Right push-button	JP3
IO6	Real time clock	N/A	JP4

LonBuilder Hardware

The Gizmo 3 attaches to a LonBuilder processor board using the Echelon model 27810 LonBuilder Application Interface Board. This board is fully described in the *LonBuilder Hardware Guide*. The features and function of this board are described here for convenience to the user.

The Application Interface Board provides access to the 11 I/O pins, 5 communications pins, reset pin, and service pin of the Neuron Chip. The board also provides +5V, ground, +12V, and -12V to external hardware. The Gizmo 3 uses the 11 I/O pins, +5V, and GND. Table 3 shows the pin-out for the application I/O interface cable shipped as part of the LonBuilder Gizmo 3 Kit.

Table 3. Gizmo 3 Application I/O Interface Cable Pin-out

Gizmo 3 and LTM-10 20-pin Connector Pin #	Signal Name	LonBuilder AIB DB-25 Connector Pin #
1	IO0	13
2	IO1	25
3	IO2	12
4	IO3	24
5	IO4	11
6	IO5	23
7	IO6	10
8	IO7	22
9	IO8	9
10	IO9	21
11	IO10	8
12	--	20
13	--	7
14	--	19
15	~SERVICE*	6
16	~RESET*	18
17	-12VDC*	5
18	+12VDC*	17
19	+5VDC	4
20	GND	16

*These signals are not used by the Gizmo 3.

IMPORTANT

For the Gizmo 3's quadrature shaft encoder, and left push-button devices to function properly with a LonBuilder emulator, you must **remove** the **JP2** jumper on the emulator board. However, this jumper must be **installed** to use the IO4 push-button on the emulator's front panel.

Also, the Neuron Chip's internal I/O pull-ups must be enabled. All applications using the Gizmo 3 must therefore include the following Neuron C compiler directive:

```
#pragma enable_io_pullups
```

The Application Interface Board provides unbuffered access to the 11 I/O pins (IO0-I10), the 5 communications pins (CP0-4), and the `~Service` pin. Refer to the *Neuron Chip Data Book* for the electrical specification of these signals.

WARNING: These signals are unbuffered. When using the application interface board with hardware other than the Gizmo 3, the Neuron Chip may be easily damaged due to electrostatic discharge or incorrect voltage levels.

The ~Reset signal is a buffered CMOS input. Asserting this signal low causes a hardware reset to occur on the emulator. Refer to the *Neuron Chip Data Book* for the timing specification of the reset signal.

The application interface board provides fuse protected access to the development station power supply (+5V, -12V, and +12V). These fuses are user replaceable. Replacement fuses must not exceed a 0.5A rating. External hardware should not exceed 400 mA for 5V, and 35 mA for +12V and -12V.

There are three jumpers on the application interface board. These jumpers connect ~Reset, ~Service, and Clock to the DB25 connector when installed. The silk screen references on the board document the function of each jumper. The default configuration (jumpers removed) should be used when using the Gizmo 3.

NodeBuilder Hardware

The Gizmo 3 attaches to the LTM-10 LonTalk Node using the supplied 20-conductor ribbon cable. The pin-out of this cable is shown in table 3.

Using the IO4 Push-button and the IO0 LED

Both the LonBuilder Neuron emulator and the LTM-10 LonTalk Node supplied with the NodeBuilder Development Tool include a momentary push-button controlling the Neuron Chip IO4 pin, and a green LED controlled by the IO0 pin.

All applications that wish to use the IO4 push-button on the LTM-10 LonTalk node must include the following Neuron C compiler directive:

```
#pragma enable_io_pullups
```

Some revisions of the LonBuilder Neuron emulator may require you to *omit* this directive for reliable operation of the IO4 push-button on the emulator. You must **install** the **JP2** jumper on the emulator board to use the IO4 push-button on the emulator. However, this jumper must be **removed** to use the Gizmo 3 with the emulator. You must install the emulator JP1 jumper to use the IO0 LED, but you need not remove it if you wish to use the Gizmo 3. See Chapter 2 of the *LonBuilder Hardware Guide* for more details.

For the LonBuilder Neuron emulator, the IO4 push-button and IO0 LED operate with *non-inverted* logic, so that if the push-button is pressed, the Neuron C application will read a 1, and if the Neuron C application writes a 1, the LED will light.

For the LTM-10 LonTalk node, the IO4 push-button and IO0 LED operate with *inverted* logic, so that if the push-button is pressed, the Neuron C application will read a 0, and if the Neuron C application writes a 0, the LED will light.

Programming Examples

A working knowledge of how to use the LonBuilder or NodeBuilder software to assign Neuron C applications to specific nodes is assumed. The Neuron C source files listed in table 4 are provided in the `EXAMPLES` directory.

Table 4. Neuron C Source Files

Source File	*Program Function*	*Hardware Required*
`ACTUATOR.NC`	LONMARK closed-loop discrete actuator	IO0 LED
`ALARM.NC`	Temperature alarm node	Gizmo 3
`ANLGSNSR.NC`	LONMARK open-loop analog sensor	Gizmo 3
`BLINK.NC`	Blink an LED	IO0 LED
`DIMMER.NC`	AC lamp control	Triac AC dimmer*
`DISPLAY.NC`	Display value of network variable	Gizmo 3
`ENCODER.NC`	Display dial position	Gizmo 3
`LAMP.NC`	Control an LED	IO0 LED
`SCPT_EX.NC`	LONMARK configuration parameter file example	None
`SENSOR.NC`	LONMARK closed-loop discrete sensor	Push-button (IO4 or Gizmo 3)
`SWITCH.NC`	Sense a push-button	Push-button (IO4 or Gizmo 3)
`SWITCH_8.NC`	LONMARK node object, eight open-loop sensor objects	Eight switches*
`TEMP.NC`	Monitors and displays temperature	Gizmo 3
`TEMP_MON.NC`	Monitor temperature with guard-point and alarm	Gizmo 3
`TEST.NC`	Tests Gizmo 3 I/O devices	Gizmo 3
`THERMOS.NC`	Thermostat device	Gizmo 3

(*) Hardware not included with LonBuilder or NodeBuilder tools

A functional specification for each example program (other than those marked with an asterisk in table 4) is provided in Appendix A. In addition, suggestions for combining the example programs to create distributed sense and control applications are described.

The Neuron C include files listed in table 5 contain drivers for the various Gizmo 3 hardware devices. These files are also provided in the EXAMPLES directory. See Appendix B for information on using these drivers in your own applications.

Table 5. Neuron C Include Files

Source File	*Functions Included*	*Hardware Required*
ANALOG.H	Sample analog inputs Update analog outputs	A/D converter D/A converter
DISPLAY.H	Numeric and alphanumeric display	Seven-segment display
GIZMO_IO.H	Miscellaneous I/O functions	Push-buttons, discrete LEDs, quadrature dial, piezo transducer
RTCLOCK.H	Maintain time and date	Real time clock
TEMPERAT.H	Temperature-related functions	Temperature sensor

Using the Examples with the LonBuilder Tool

To prepare your LonBuilder object database to run the example applications with the LonBuilder tool, follow these steps. The first program you will run is TEST.NC. This application exercises a variety of the I/O devices on the Gizmo 3.

To install this program example into a Neuron emulator:

1. Make the directory to which the examples were copied the current working directory. Create a sub-directory named PROJECT if one does not already exist.
2. Invoke the LonBuilder software using the following command: LB PROJECT
3. Using the Navigator screen in the LonBuilder system, select the App Node class of objects from the top-level menu.
4. Select the Node Spec sub-class of objects from the **App Node** window. Create a node specification for the emulator which has the Gizmo 3 attached, and enter TEST in the App Image Name field. Enter the name of the emulator (for example emulator_1) in the Target HW field, and a node name (for example test_node) in the Node Name field. Save the node specification record.
5. Return to the **App Node** window, and request an Automatic Load from the project menu. Review the functional specification that follows for this application to verify proper hardware operation.
6. Experiment with the other examples provided:

   ACTUATOR ALARM ANLGSNSR BLINK DIMMER DISPLAY ENCODER LAMP

```
SCPT_EX SENSOR SWITCH SWITCH_8 TEMP TEMP_MON TEST THERMOS
```

Using the Examples with the NodeBuilder Tool

To prepare your NodeBuilder project to run the example applications with the NodeBuilder tool, follow these steps. The first program you will run is `TEST.NC`. This application exercises a variety of the I/O devices on the Gizmo 3.

To install this program example into the LTM-10 LonTalk Node:

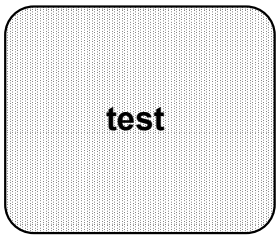
1. Make the directory to which the examples were copied the current working directory.
2. Invoke the NodeBuilder software by clicking on its icon in the Windows Program Manager
3. If a device window is already open, close it. Create a new device by clicking on the Device icon.
4. In the General tab of the Device window, enter TEST into the Application Image edit box. Make sure that the Device Template is set to LTMRAM.DTM.
5. Click on the Save button, and save the device as TEST.DEV.
6. Click on the Build and Load button to compile and load the application into the LTM-10 LonTalk Node.
7. Review the functional specification that follows for this application to verify proper hardware operation.
8. Experiment with the other examples provided.

   ```
   ACTUATOR ALARM ANLGSNSR BLINK DIMMER DISPLAY ENCODER LAMP
   SCPT_EX SENSOR SWITCH SWITCH_8 TEMP TEMP_MON TEST THERMOS
   ```

   You cannot use the NodeBuilder tool to bind nodes together in order to create the multiple node examples described below. You can use a network management tool such as LonMaker™ for this purpose.

Appendix A - Sample Applications

Program - `test.nc`



Input Network Variables

None

Output Network Variables

None

Input I/O Objects

`IO_4..5`	`ioDial`	`quadrature`
`IO_7`	`ioLeftPB`	`bit`
`IO_3`	`ioRightPB`	`bit`

Output I/O Objects

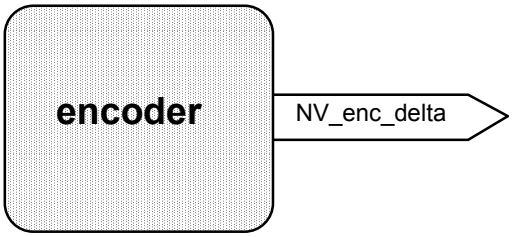
`IO_0`	`ioPiezo`	`frequency`
`IO_1`	`ioRedLED`	`bit`
`IO_2`	`ioGreenLED`	`bit`
`IO_8..9, IO_2`	`ioSevenSeg`	`neurowire`

Functional Specification

This program tests the LED, push-button, piezo-electric and quadrature devices on the Gizmo 3. The following operation is expected for a working module:

1. All segments of the display are lit when the Neuron Chip is reset.
2. The piezo transducer plays a tune when the right push-button is pressed.
3. The encoder position is displayed when changed.
4. The red LED lights up when the left push-button is pressed.
5. The green LED flickers as the display is updated.

Program - `encoder.nc`



Input Network Variables

None

Output Network Variables

```
NV_enc_delta        long
```

Input I/O Objects

```
IO_4..5             ioDial          quadrature
```

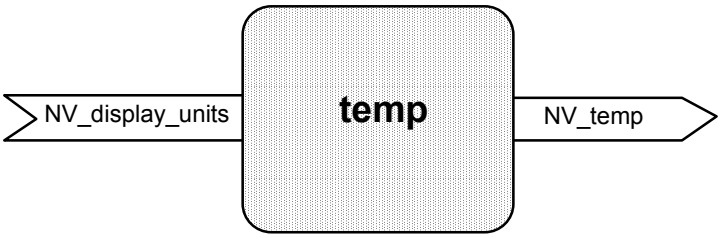
Output I/O Objects

None

Functional Specification

When the dial is turned, the node attached to the Gizmo 3 sends out a network variable, `NV_enc_delta`, indicating the incremental change in value of the encoder.

Program - `temp.nc`



Input Network Variables

```
NV_display_units    config          TempUnits
```

Output Network Variables

```
NV_temp             SNVT_temp
```

Input I/O Objects

```
IO_8..9, IO_1       ioA2D           neurowire master
```

Output I/O Objects

```
IO_8..9, IO_2       ioSevenSeg      neurowire master
```

Functional Specification

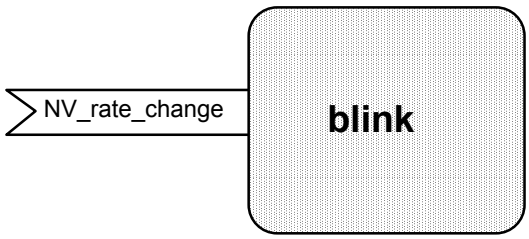
The temperature sensor is read and displayed every 0.5 seconds. If a change in temperature has occurred, the updated value is sent out as the `NV_temp` network variable. The `NV_display_units` network variable is declared as a `config` network variable and can therefore be modified by a network management node. The value of this variable determines whether the temperature is shown in degrees Celsius or Fahrenheit. The resolution of the value displayed is either 1 degree Fahrenheit or 0.5 degrees Celsius. Modify the default value by changing the

```
#define DEFAULT_TEMP_UNIT
```

statement in the file `TEMPERAT.H`. Choose either of the following:

```
#define DEFAULT_TEMP_UNIT CELSIUS
#define DEFAULT_TEMP_UNIT FAHRENHEIT
```

Program - `blink.nc`



Input Network Variables

```
NV_rate_change      long
```

Output Network Variables

None

Input I/O Objects

None

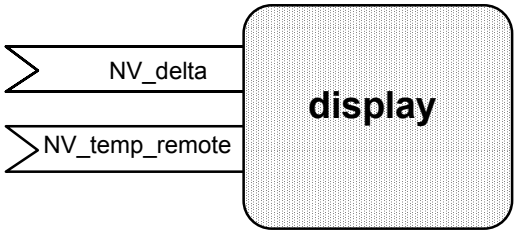
Output I/O Objects

```
IO_0                IO_emulator_led oneshot
```

Functional Specification

The LED is flashed at a frequency changed in proportion to the integrated change in the value of the `NV_rate_change` input network variable. The input network variable for this application may be attached to the output network variable of the `ENCODER.NC` application.

Program - `display.nc`



Input Network Variables

```
NV_delta            long
NV_temp_remote      SNVT_temp
```

Output Network Variables

None

Input I/O Objects

None

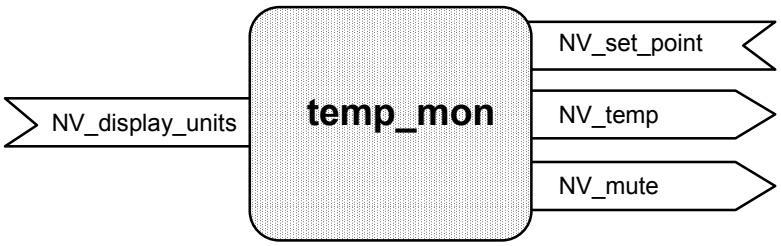
Output I/O Objects

```
IO_1                ioRedLED        bit
IO_8..9, IO_2       ioSevenSeg      neurowire master
```

Functional Specification

A slave seven-segment display that either shows temperature or an incrementing value. While showing a temperature, the red LED is lit. The input network variables to this application support connection to the output network variables of the `TEMP.NC`, and `ENCODER.NC` applications.

Program - `temp_mon.nc`



Input Network Variables

```
NV_set_point        SNVT_temp
NV_display_units    config TempUnits
```

Output Network Variables

```
NV_temp             SNVT_temp
NV_mute             SNVT_lev_disc
```

Input I/O Objects

```
IO_8..9, IO_1       ioA2D           neurowire master
IO_7                ioLeftPB        bit
```

Output I/O Objects

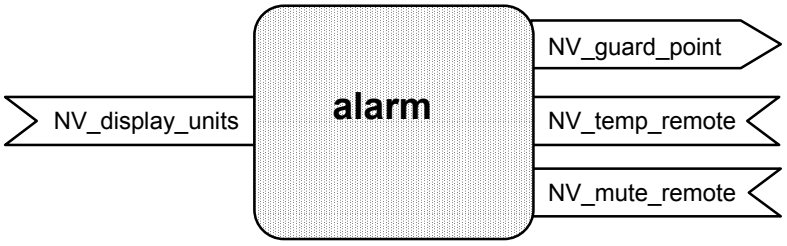
```
IO_1                ioRedLED        bit
IO_8..9, IO_2       ioSevenSeg      neurowire master
```

Functional Specification

The current temperature of the sensor is sent out as the `NV_temp` network variable. The `NV_set_point` network variable provides a temperature setpoint. The left push button toggles the seven-segment display between displaying the setpoint and displaying the current temperature. Whenever the setpoint is displayed, the red

LED is lit. The `NV_display_units` configuration network variable determines whether the temperature is shown in degrees Celsius or Fahrenheit.

Program - `alarm.nc`



Input Network Variables

```
NV_temp_remote      SNVT_temp
NV_mute_remote      SNVT_lev_disc
NV_display_units    config TempUnits
```

Output Network Variables

```
NV_guard_point      SNVT_temp
```

Input I/O Objects

```
IO_3                ioRightPB       bit
IO_4                ioDial          quadrature
IO_7                ioLeftPB        bit
```

Output I/O Objects

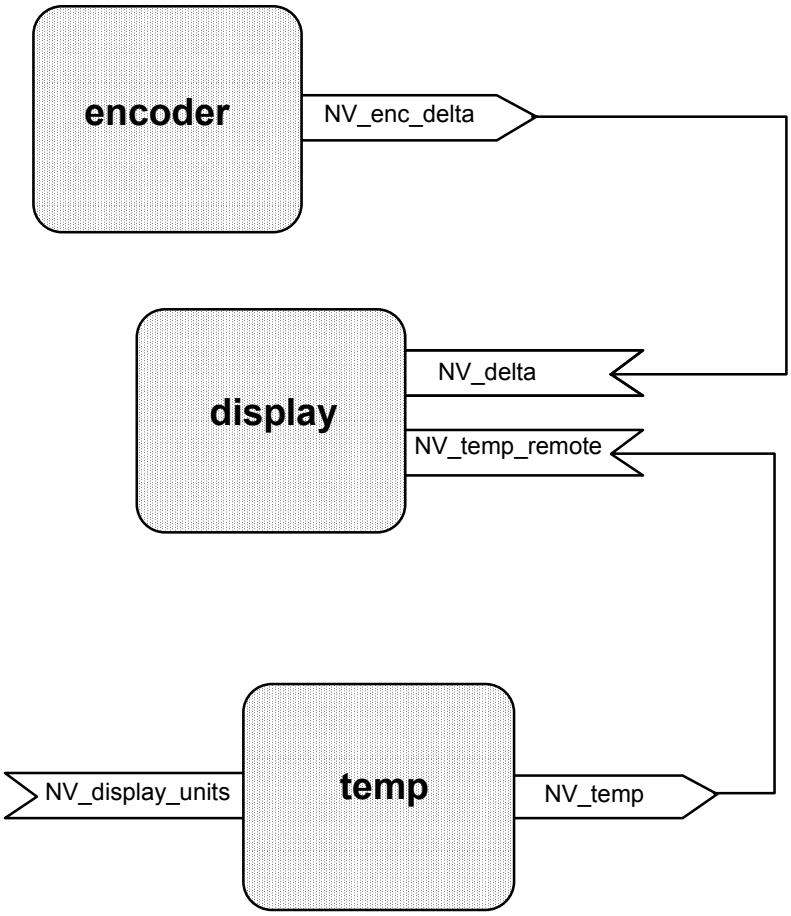
```
IO_0                ioPiezo         frequency
IO_1                ioRedLED        bit
IO_8..9, IO_2       ioSevenSeg      neurowire master
```

Functional Specification

The dial is used to set a temperature guard point which is sent out as the `NV_guard_point` network variable. The left push button toggles the seven-segment display between displaying the guard point and displaying the current temperature. The guard point is automatically displayed for 2 seconds when its value is updated. Whenever the guard point is displayed, the red LED is lit. The `NV_temp_remote` network variable is compared with the guard point temperature and the audible alarm is sounded if the temperature equals or exceeds the limits.

The alarm can be muted based on the `NV_mute_remote` network variable. The `NV_display_units` network variable determines whether the temperature is shown in degrees Celsius or Fahrenheit. The default value is specified in the file `TEMPERAT.H`.

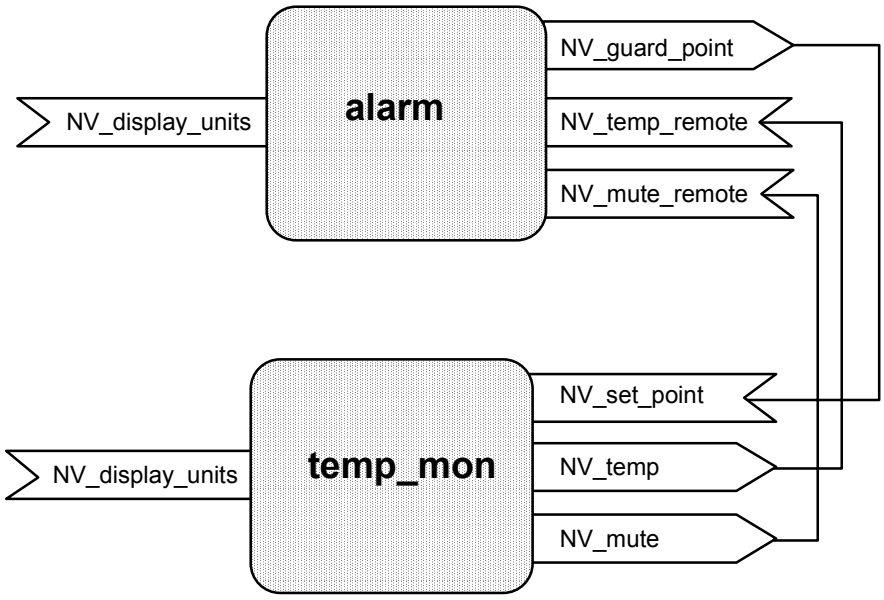
Application Example - Encoder/Temperature Display



Description

This example illustrates an encoder or temperature display. The display node can receive either a temperature value from the temperature sensor or an incremental value from the encoder for display on the seven-segment display.

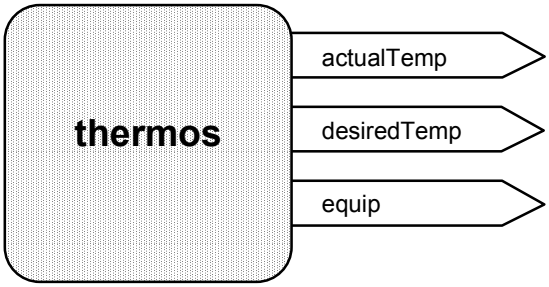
Application Example - Remote Temperature Alarm



Description

This example illustrates a remote temperature alarm. One node acts as a temperature sensor and the other node as a remote alarm. The temperature guard point is set on the alarm node. The current temperature is monitored at the sensor. Whenever its value changes, the updated value is communicated to the remote alarm. The alarm checks the current temperature against the guard point. The alarm is sounded when the guard point is exceeded and remains on until the temperature goes below the guard point. If the `NV_mute_remote` network variable changes state to `ST_OFF`, the alarm is silenced.

Program - `thermos.nc`



Input Network Variables

None

Output Network Variables

```
actualTemp          SNVT_temp
desiredTemp         SNVT_temp
equip               enum { OFF, HEATING, COOLING }
```

Input I/O Objects

```
IO_8..9, IO_1       ioA2D           neurowire master
```

Output I/O Objects

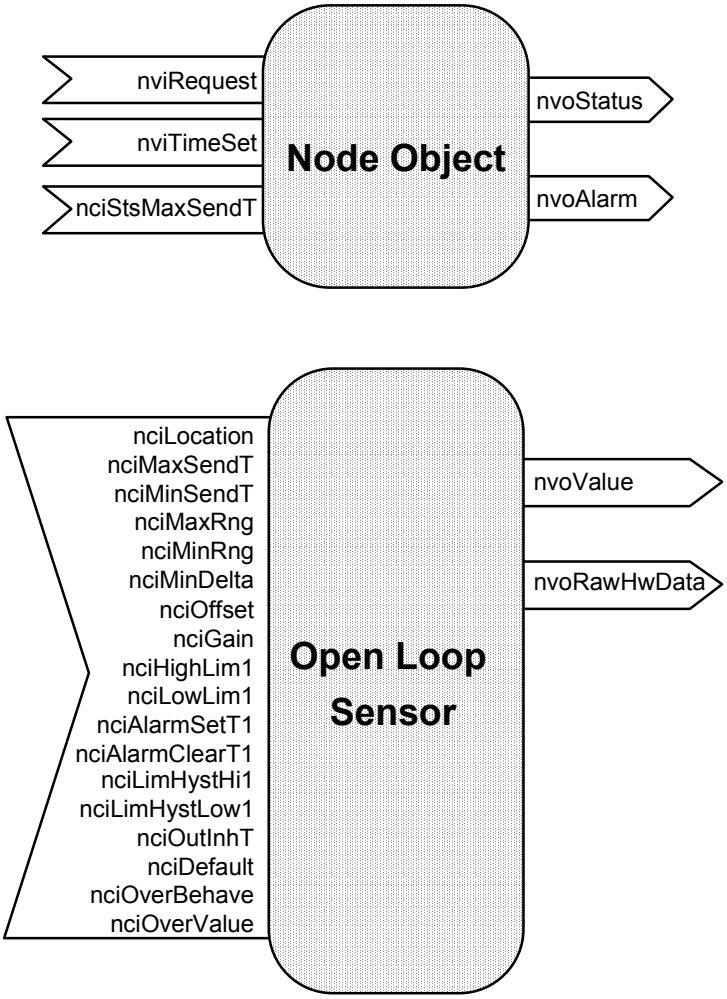
```
IO_1                ioRedLED        bit
IO_2                ioGreenLED      bit
```

Functional Specification

The current temperature of the sensor is measured every five minutes and compared to the desired temperature. If it is too hot, the cooling equipment is activated (indicated by the green LED). If it is too cold, the heating equipment is activated (indicated by the red LED). If the measured temperature is in a dead band of 1°C around the desired temperature, all equipment is inactivated. The desired temperature is set using the quadrature dial encoder. The output network variables `actualTemp` and `desiredTemp` contain the measured and desired temperature respectively. The output network variable `equip` contains the state of the heating and cooling equipment.

Program - `anlgsnsr.nc`

This example complies with the LONMARK® Application Layer Interoperability Guidelines, Revision 3.0.



Input Network Variables

```
nviRequest          SNVT_obj_request
nviTimeSet          SNVT_time_stamp
nciStsMaxSendT      config SNVT_elapsed_tm
nciLocation         config SNVT_str_asc
nciMaxSendT         config SNVT_elapsed_tm
nciMinSendT         config SNVT_elapsed_tm
nciMaxRng           config SNVT_temp
nciMinRng           config SNVT_temp
```

```
nciMinDelta         config SNVT_temp
nciOffset           config SNVT_temp
nciGain             config SNVT_muldiv
nciHighLim1         config SNVT_temp
nciLowLim1          config SNVT_temp
nciAlarmSetT1       config SNVT_elapsed_tm
nciAlarmClearT1     config SNVT_elapsed_tm
nciLimHystHi1       config SNVT_temp
nciLimHystLow1      config SNVT_temp
nciOutInhT          config SNVT_elapsed_tm
nciDefault          config SNVT_temp
nciOverBehave       config SNVT_override
nciOverValue        config SNVT_temp
```

Output Network Variables

```
nvoStatus           SNVT_obj_status
nvoAlarm            SNVT_alarm
nvoValue            SNVT_temp
nvoRawHwData        SNVT_count
```

Input I/O Objects

```
IO_3                ioRightPB       bit
IO_7                ioLeftPB        bit
IO_8..9, IO_1       ioA2D           neurowire master
```

Output I/O Objects

```
IO_8..9, IO_2       ioSevenSeg      neurowire master
```

Functional Specification

The application reads the temperature sensor using the A/D converter, calibrates the raw data, and displays the result. When the right push-button is pressed, the result is displayed in degrees Celsius. When the left push-button is pressed, the result is displayed in degrees Fahrenheit. See the *LONMARK Application Layer*

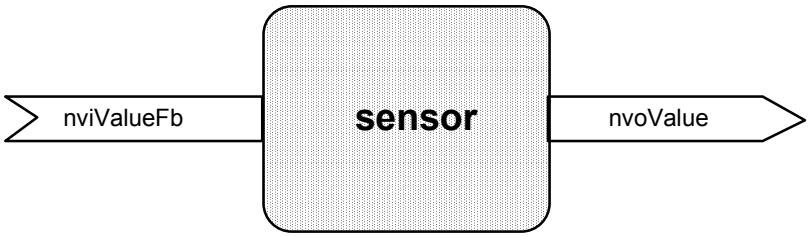
Interoperability Guidelines for a complete description of the functionality of the Node Object and the Open Loop Sensor Object.

This example must be compiled with version 6.02 or later of the `SNVT.TYP` file.

Push-button and LED Application Examples

Program - `sensor.nc`

This example complies with the LONMARK® Application Layer Interoperability Guidelines, Revision 3.0.



Input Network Variables

```
nviValueFb          SNVT_switch
```

Output Network Variables

```
nvoValue            SNVT_switch
```

Input I/O Objects

```
IO_4                ioButton        bit
IO_7                ioLeftPB        bit
IO_3                ioRightPB       bit
```

Output I/O Objects

```
IO_0                ioLED           bit
```

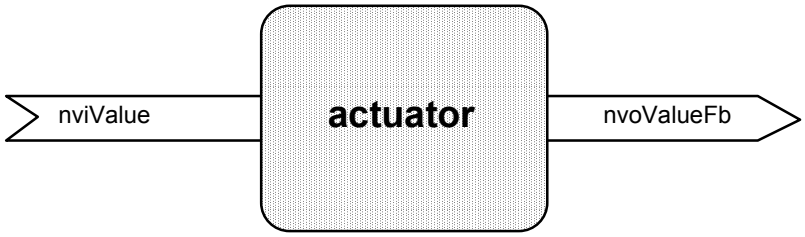
Functional Specification

When any push-button (IO4 or Gizmo 3) is pressed, the node updates its output network variable `nvoValue`. If the feedback input network variable nviValueFb indicates OFF, `nvoValue` is set to the ON state with a full-scale level. If the feedback input indicates ON, `nvoValue` is set to the OFF state with a full-scale level. When the feedback input network variable nviValueFb is updated, the IO0 LED is

set to the state indicated by the feedback variable. The node is implemented as a LONMARK-compliant closed loop sensor.

Program - `actuator.nc`

This example complies with the LONMARK® Application Layer Interoperability Guidelines, Revision 3.0.



Input Network Variables

```
nviValue            SNVT_switch
```

Output Network Variables

```
nvoValueFb          SNVT_switch
```

Input I/O Objects

None

Output I/O Objects

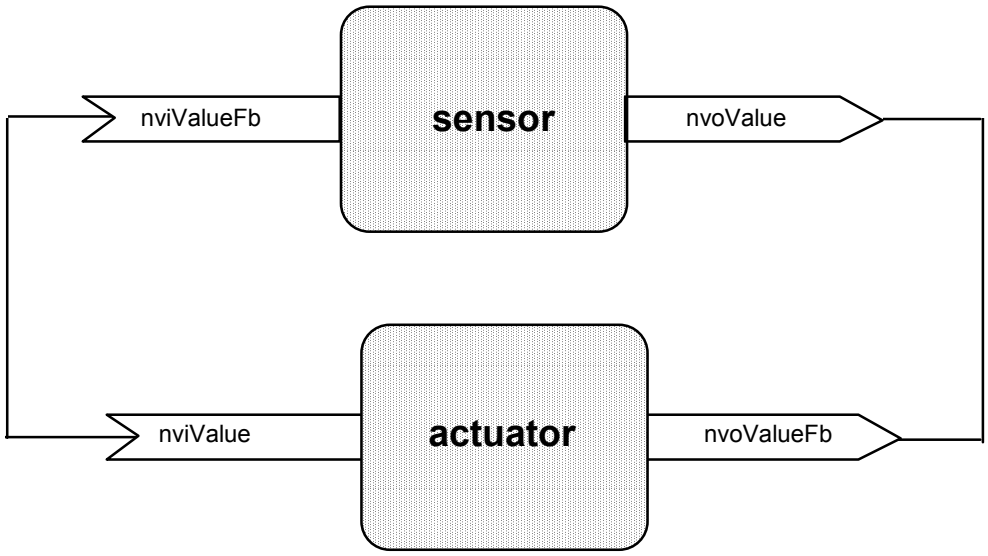
```
IO_0                ioLED           bit
```

Functional Specification

When the input network variable `nviValue` is updated, the IO0 LED is set to the state indicated by the input variable. The feedback output network variable `nvoValueFb` is updated to reflect the value of the input network variable. The node is implemented as a LONMARK-compliant closed loop actuator.

Application Example - Closed Loop Sensor and Actuator

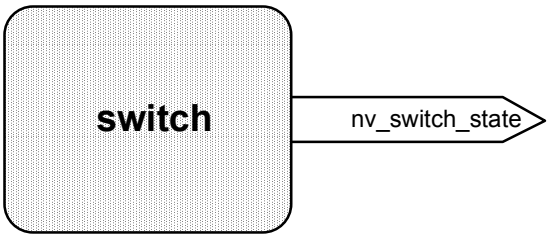
This example complies with the LONMARK® Application Layer Interoperability Guidelines, Revision 3.0.



Description

This example illustrates a closed-loop sensor and actuator system. When the push-button on the sensor node is pressed, the state of the actuator is toggled, and then fed back to the sensor node. Thus each time the push-button is pressed, the LED on the actuator alternates between off and on. If more than one sensor is used for the same actuator, the feedback keeps them all synchronized.

Program - `switch.nc`



Input Network Variables

None

Output Network Variables

```
nv_switch_state     SNVT_lev_disc
```

Input I/O Objects

```
IO_4                ioButton        bit
IO_7                ioLeftPB        bit
IO_3                ioRightPB       bit
```

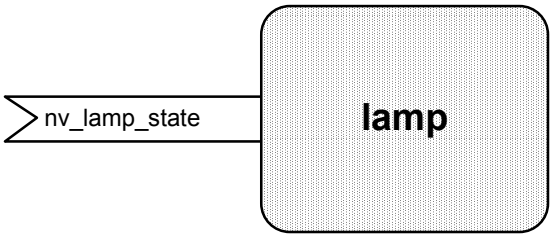
Output I/O Objects

None

Functional Specification

When any push-button (IO4 or Gizmo 3) is pressed, the node updates its output network variable `nv_switch_state` to the value `ST_ON`. When the push-button is released, the output network variable is set to `ST_OFF`.

Program - `lamp.nc`



Input Network Variables

```
nv_lamp_state       SNVT_lev_disc
```

Output Network Variables

None

Input I/O Objects

None

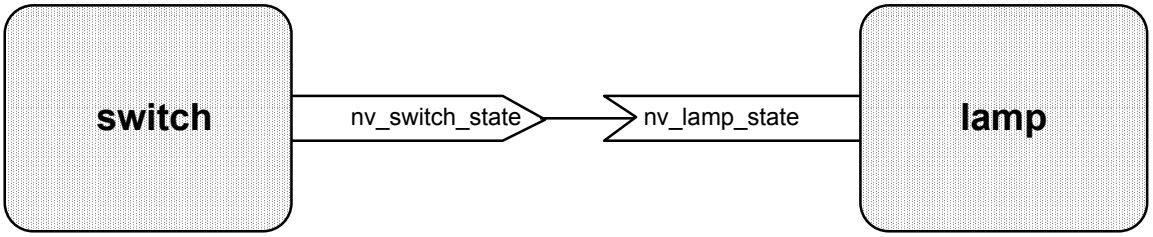
Output I/O Objects

```
IO_0                ioLED           bit
```

Functional Specification

When the input network variable `nviValue` is updated, the LED is set to the state indicated by the input variable.

Application Example - Switch and Lamp



Description

This example illustrates a simple switch and lamp system. When any push-button on the sensor node is pressed, the IO0 LED on the lamp node is turned on. When the push-button on the sensor node is released, the IO0 LED on the lamp node is turned off.

Appendix B - Device Driver Software

The Gizmo 3 example software comes with four Neuron C include files that contain useful definitions and functions for driving the Gizmo 3 hardware devices. This section describes how you can use these functions to simplify your own application development for the Gizmo 3 hardware. See the *Neuron Chip Data Book* and the *Neuron C Reference Guide* for more detailed descriptions of the I/O models used. Two of these device driver files may be used with the older Gizmo 2 device.

Driver File	Devices Supported
GIZMO_IO.H	Simple I/O devices. Same for both Gizmo versions.
DISPLAY.H	Seven segment LED display Gizmo 3 has 5 digits, Gizmo 2 has 4 digits

The files ANALOG.H, RTCLOCK.H, and TEMPERAT.H may only be used with the Gizmo 3.

GIZMO_IO.H

This file contains Neuron C declarations for the simpler I/O devices on the Gizmo 3.

```
IO_0 output frequency   clock(1) ioPiezo;
IO_1 output bit         ioRedLED = 1;
IO_2 output bit         ioGreenLED = 1;
IO_3 input  bit         ioRightPB;
IO_4 input  quadrature  ioDial;
IO_7 input  bit         ioLeftPB;
```

This file also contains the following definitions of symbolic literals that may be used to represent the states of the push-buttons and discrete LEDs.

```
#define     LED_OFF     1
#define     LED_ON      0
#define     PB_DOWN     0
#define     PB_UP       1
```

Examples:

```
io_out (ioPiezo, 250);     // create 5 kHz sound
io_out (ioRedLED, LED_ON); // turn on red LED
when (io_changes(ioRightPB) to PB_DOWN) // wait for button to be pressed
when (io_update_occurs(ioDial))  // wait for quad dial to be turned
```

ANALOG.H

This file contains the following I/O declarations for the A/D and D/A converters.

```
IO_8 neurowire master select(IO_1) ioA2D;
IO_1 output bit ioA2DSelect = 1;
IO_8 neurowire master select(IO_3) ioD2A;
IO_3 output bit ioD2ASelect = 1;
```

The Neuron C function definition to read the A/D converter is as follows:

```
long A2DConvert (unsigned muxAddr);
```

The A/D converter has a five-channel multiplexer, and returns a 10-bit result in the range 0 .. 1023, with a full-scale value corresponding to 5 volts on the input.

The application program calls the `A2DConvert()` function, passing in the multiplexer channel address in the range 0 .. 4. This function initiates an A/D conversion on that channel. At the same time, it reads the results of the *previous* A/D conversion, for whatever channel was specified on the *previous* call to `A2DConvert()`. The function returns the value of the latest conversion on the channel specified by the `muxAddr` parameter. If the `muxAddr` parameter is out of range, or if there has been no previous conversion on the specified channel, then the function returns the value -1.

Example:

```
value = A2DConvert(0);  // read the temperature sensor
```

The Neuron C function definition to write to the D/A converter is as follows:

```
void D2AConvert (unsigned data, unsigned muxAddr);
```

The D/A converter has four channels, and six bits of resolution. The application program calls the `D2AConvert()` function, passing in the channel address in the range 0 .. 3, and a data value in the range 0 .. 63. Full scale corresponds to 5 volts on the output.

Example:

```
D2AConvert (25, 2);  // output 1.98 volts on channel 2
```

TEMPERAT.H

This file contains definitions relating to the temperature sensor. The Neuron C function `ConvertRawTemp()` converts the raw reading from channel 0 of the A/D converter to `SNVT_temp` units. The function definition is as follows:

```
SNVT_temp ConvertRawTemp (unsigned long rawTemperature);
```

The standard network variable type SNVT_temp has a resolution of 0.1°C, and a range of -274.0°C to 6279.5°C. See the *SNVT Master List and Programmer's Guide* for more details.

Example:

```
SNVT_temp currentTemperature;
currentTemperature = ConvertRawTemp (A2DConvert (0)); // read temperature
```

This file also contains macro definitions which convert integer decimal constants in degrees Fahrenheit and degrees Celsius to `SNVT_temp` units.

Examples:

```
SNVT_temp maxTemperature = TEMP_DEG_C(85);   // initialize to 85°C
SNVT_temp minTemperature = TEMP_DEG_F(0);    // initialize to 0°F
```

RTCLOCK.H

This file contains the following I/O declarations for the real-time clock chip.

```
IO_8 neurowire master select(IO_6) ioRtc;
IO_6 output bit ioRtcSelect = 1;
```

The Neuron C function definition to write to the real-time clock chip is as follows:

```
void RTCSetTime(const SNVT_time_stamp * pTimeSet);
```

The `pTimeSet` parameter points to a structure of type SNVT_time_stamp with the following declaration:

```
typedef struct {
      unsigned long  year;
      unsigned short month;
      unsigned short day;
      unsigned short hour;
      unsigned short minute;
      unsigned short second;
} SNVT_time_stamp;
```

See the *SNVT Master List and Programmer's Guide* for more details of this structure.

The valid range for `year` is 1990 to 2089.

Example:

```
SNVT_time_stamp timeSet = { 1996, 12, 1, 20, 30, 0 }; // 8:30 pm Dec 1,
1996
RTCSetTime (&timeSet);  // set the real-time clock
```

The Neuron C function to read the real-time clock chip is as follows:

```
void RTCGetTime(SNVT_time_stamp *pTimeGet,
      SNVT_date_day *pDayOfWeek);
```

The parameter `pTimeGet` points to a structure of type `SNVT_time_stamp`. For the returned date and time. The parameter `pDayOfWeek` points to an enumeration byte for the returned day of the week, where 0 = Sunday, 1 = Monday … 6 = Saturday. The `pDayofWeek` pointer may be 0, in which case the day of the week is not returned.

Example:

```
SNVT_time_stamp Now;
SNVT_date_day Today;
RTCGetTime (&Now, &Today);    // read the real-time clock
```

DISPLAY.H

This file contains the following I/O declarations for the seven-segment LED display chip.

```
IO_8 neurowire master select(IO_2) ioSevenSeg;
IO_2 output bit io7SegSelect = 1;
```

The file contains Neuron C functions to display decimal data on the Gizmo 3 LED display, functions to display strings consisting of the characters displayable in seven segments, and a function to display temperatures values.

Before using this code, make sure that the number of digits in your display is specified correctly by the `NUM_DIGITS` parameter. For the display in the Gizmo 3, leave the `NUM_DIGITS` parameter at 5. For the display in the older Gizmo 2 device, change `NUM_DIGITS` to 4. The digits are numbered as shown in figure 2.

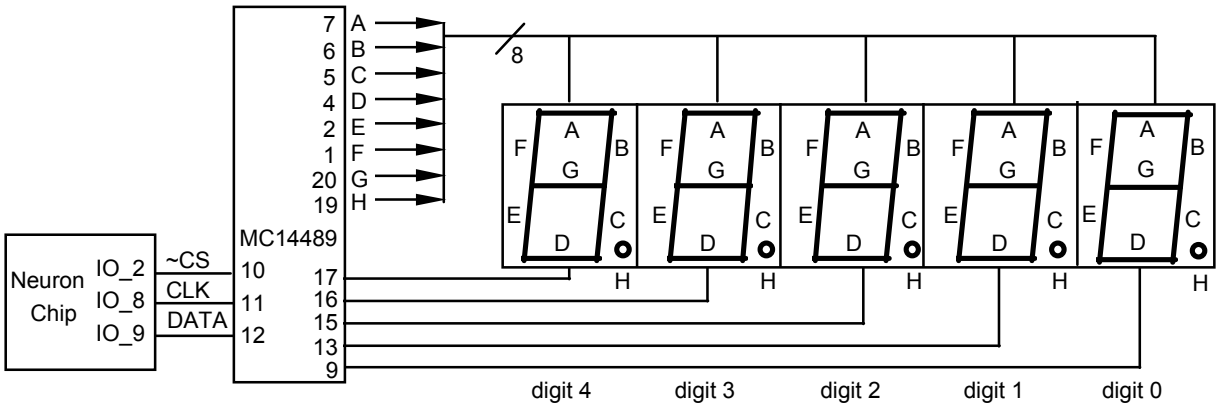


Figure 2. **Gizmo 3 Seven Segment LED Display**

The software functions are divided into three groups; low-level functions, display image update functions, and high-level functions.

Low-Level Functions

The first group of functions provides low-level access to the display controller chip.

The `DspClearImage()` function clears a RAM copy of the configuration and display registers (the variables `dspCfgReg` and `dspDataReg)` to a state that displays all blank characters. It does this by setting all digits to special decode mode, and writing the data for the blank character to all digits.

The `DspUpdateDisplay()` function uses the Neurowire I/O model to write the contents of the RAM copy of the configuration and display registers to the actual MC14489 device registers. The Neurowire device is full-duplex, so that the

`io_out()` operation which updates the hardware registers in the MC14889 also causes data to be shifted in from the IO10 pin and stored in the RAM variables. Therefore a local copy of these variables is used for the `io_out()` operation, so that `DspUpdateDisplay()` may be called repeatedly without having to refresh the RAM copy of the variables.

Display Image Update Functions

The second group of functions are routines that write into the RAM copy of the configuration and display registers. They do not update the hardware device registers.

The `DspInsertDecimal(int digitNumber, int number)` function writes the specified decimal (0 - 9) into the specified digit position in the RAM copy of the display register.

The `DspInsertDecimal2(int rightDigit, unsigned number)` function writes a two-digit decimal number (00 - 99) into the specified digit positions in the RAM copy of the display register.

The `DspInsertMinus(int digitNumber)` function writes the appropriate values in the RAM copy of the display register to illuminate the minus sign (segment G) in the specified digit.

The `DspInsertChar(int digitNumber, char ch)` function writes the data value for an ASCII character in the RAM copy of the display register. The letters available in upper case are "A, C, E, F, H, I, J, L, O, P, S, U, Y, Z", and in lower case "b, c, d, h, l, n, o, r, u, y". Digits 0-9 are displayed, as well as the space ' ', degree '°', minus '-', and equals '=' special characters. If the ASCII character is none of these characters, nothing is written, leaving the display unchanged for that digit.

The `DspInsertNumber(long number, int dpDigit, int rightDigit)` function updates the RAM copy of the display register to display a signed decimal number. The function illuminates a decimal point to the right of the specified digit. If the `dpDigit` parameter is `NO_DP`, no decimal point is illuminated. If the `dpDigit` parameter is `ALL_DPS`, all decimal points will be illuminated. The caller also specifies the right-most digit position of the displayed number. Display data to the right of this position are unchanged. If the number to be displayed does not fit in the specified field, all digits are set to the minus character.

High-Level Functions

The third group of functions forms complete display images and updates the display hardware.

The `DspDisplayBlanks()` function clears the display.

The `DspDisplayString(const char * pString, int dpDigit)` function causes the first 4 or 5 ASCII characters in the specified string to be displayed. Five characters are displayed for a Gizmo 3, and four characters are displayed for a Gizmo 2. The letters available in upper case are "A, C, E, F, H, I, J, L, O, P, S, U, Y, Z", and in lower case "b, c, d, h, l, n, o, r, u, y". Digits 0-9 are displayed, as well as the space ' ', degree '°', minus '-', and equals '=' special characters. If other letters, or more elegant letters are desired, an alphanumeric display should be used instead of a seven-segment display.

The `DspDisplayNumber(long number, int dpDigit, int rightDigit)` function displays positive or negative decimal numbers with a decimal point to the right of the specified digit, with suppression of leading zeroes. The special values `NO_DP` and `ALL_DPS` may be used as the decimal point digit number to illuminate none or all, respectively, of the decimal points. The parameter `rightDigit` indicates the digit position for the least significant digit of the displayed number. Numbers that do not fit into the specified field are displayed as all minus characters '`-----`'.

The following examples show the display produced by different input values on a five-digit display.

Examples:

```
DspDisplayNumber(1234, 0, 0)         =>     1 2 3 4.
DspDisplayNumber(1234, 1, 0)         =>     1 2 3.4
DspDisplayNumber(1234, 2, 0)         =>     1 2.3 4
DspDisplayNumber(1234, 3, 0)         =>     1.2 3 4
DspDisplayNumber(1234, 4, 0)         =>   0.1 2 3 4
DspDisplayNumber(1234, NO_DP, 0)     =>     1 2 3 4
DspDisplayNumber(1234, ALL_DPS, 0)   =>    .1.2.3.4.

DspDisplayNumber(-1234, 0, 0)        =>   - 1 2 3 4.
DspDisplayNumber(-1234, 1, 0)        =>   - 1 2 3.4
DspDisplayNumber(-1234, 2, 0)        =>   - 1 2.3 4
DspDisplayNumber(-1234, 3, 0)        =>   - 1.2 3 4
DspDisplayNumber(-1234, 4, 0)        =>   -.1 2 3 4
DspDisplayNumber(-1234, NO_DP, 0)    =>   - 1 2 3 4
DspDisplayNumber(-1234, ALL_DPS, 0) =>   -.1.2.3.4.
```

The `DspDisplayTemp(SNVT_temp temp, boolean dspFahrenheit)` function displays temperature values in either Celsius or Fahrenheit, with one decimal place. For more details on the Standard Network Variable Type `SNVT_temp`, see the *SNVT Master List and Programmer's Guide.*

Examples:

`DspDisplayTemp(2940, FALSE)` displays `'20.0C'`

`DspDisplayTemp(2940, TRUE)` displays `'68.0F'`

Part Number 005-0016-01 Rev. H

Echelon Corporation
4015 Miranda Avenue
Palo Alto, CA 94304
+1-408-938-5200
+1-408-328-3801 fax
www.echelon.com

Echelon UK
16, The Courtyards
Hatters Lane
Watford
Herts. WD18 8YH
United Kingdom
+44 (0)1923 430200
+44 (0)1923 430300 fax

Echelon Japan
8F 1-25-13 Higashi-Gotanda
Shinagawa-ku, Tokyo 141
Japan
+81-3-3440-7781
+81-3-3440-7782 fax