AN1251

Programming the MC143120 NEURON CHIP

INTRODUCTION

This application note describes how to download an application program to the MC143120 NEURON CHIP over the communications network. Typically this is not implemented with the MC143150, because the program is in external EPROM and only address, binding, and communication information is stored in internal EEPROM. This application note describes programming solely for the MC143120, which has no external address or data lines and is always programmed over the network.

The MC143150 is used as the network manager. When a service pin message is received, a table in the MC143150 application representing the MC143120's application (and configuration data) program is downloaded over the network. IO1 is toggled whenever an acknowledgment is received from the MC143120.

If a new data rate needs to be programmed, the program must be modified to change the configuration at the start and the node reset. This is because the program resets the MC143120 and if changes are made to the configuration data (such as the communication data rate), communication will be lost. Appendix D presents a program called `comm1.nc` which will change the communication rate to 78 kbps, 5 MHz and optionally display the service pin ID on an Echelon Gizmo 2 or Motorola Gizmo 3 box. The changes do not take effect until the node is reset. For further information on configuration changes, refer to Motorola's DL159 *LONWORKS Technology Device Data*, Appendix B.

BACKGROUND

The NEURON CHIP contains a media access processor, a network processor, and an application processor. Most network management commands received are processed by the network processor and do not make it to the application processor.

A NEURON CHIP need not have an application in it to be programmed.

All NEURON CHIP programming can be done over the network. The transceiver on the programming node must be compatible with the receiving node's transceiver. The MC143120 can be programmed in a socket, or after it is soldered to a printed circuit board. The correct approach depends on the transceiver on the printed circuit board.

The defaults of a new MC143120 are 10 MHz, 1.25 Mbps, and differential input. It is possible to program a new MC143120 at 78 kbps by lowering the external clock rate from 10 MHz to 625 kHz. To program a new MC143120, Echelon's 3120 programmer runs at 5 MHz which scales the network speed +625 kbps. Most programmers use the MC143150 to program the MC143120. Lowering the clock from 10 MHz to 5 MHz allows use of a lower cost EPROM.

A new MC143120 initially set up for 1.25 Mbps may not be programmable if a 78 kbps transformer coupled transceiver will be used; however, at short distances a transformer tuned to a specific bit rate may still work at different bit rates. Alternative techniques include connecting another connector off the board to program the MC143120, or programming the device in a socket prior to soldering it down to the board.

OPTIONS FOR PROGRAMMING THE MC143120

Currently, there are five commercially available methods for programming the MC143120:

1. LONBUILDER Developer's Workbench
2. Echelon's 3120 Programmer
3. System General's Gang Programmer
4. Echelon's Application Programmer's Interface (API)
5. M143205EVK — a Motorola Evaluation Kit

LONBUILDER Developer's Workbench

Using this method, a Direct Connect board made by Motorola (M143204EVK) can be used to connect the LONBUILDER's differential direct connect backplane to a custom node. The use of a custom node requires the use of a differential transceiver. At short distances, a differential network connected to the Direct Connect board can communicate to various differential networks, such as an EIA–485 node or a transformer node. Distances of up to 50 meters have been tested with this approach; however, it is not recommended to mix a differential network in normal operations unless a router or gateway is used.

Echelon's 3120 Programmer

Model 21700: LONBUILDER NEURON 3120 Programmer.

The programmer must remain connected to a PC to program the MC143120. Echelon's programmer programs only MC143120s with the following initial parameters: 10 MHz, 1.25 Mbps, and differential input.

System General's Gang Programmer

Part number: "NEURON 3120" used in a "TURPRO–832" base.

This programmer also must be connected to a PC. It programs up to eight MC143120s. System General's programmer carries the same limitations as Echelon's; the MC143120 must be set up with the following parameters: 10 MHz, 1.25 Mbps, and differential. In addition, the MC143120 must be applicationless prior to use of the gang programmer. Once programmed, the MC143120 cannot be reprogrammed by the System General programmer unless returned to an applicationless state. On an MC143120 with firmware version 3, clearing address 0xf029 will reset it back to the factory defaults, which are 10 MHz, 1.25 Mbps, differential input, and applicationless.

The System General uses the NXE format, whereas Echelon's 3120 programmer uses the NEI format. The NXE format is the downloadable application with **no** connection information; NXE is set up only for unconfigured nodes. The NEI format differs from the NXE in that it can be set up as applicationless, unconfigured, or configured. The NEI format is an internal file format used by Echelon; it is also used by the `prog3120.nc` application in this program. For details, refer to the LONBUILDER User's Guide, Chapter 7.

Echelon's API

With the use of a PC, a tool such as Echelon's LONMANAGER, which is written using Echelon's Application Programmer's Interface, can be used to program MC143120s.

It is also possible to write one's own programmer, either of the "single" programmer or "gang" programmer description.

HOW THE APPLICATION WORKS

The steps to download configuration data and an application are detailed in Motorola's DL159, *LONWORKS Technology Device Data,* Appendix B. This application note covers only the actual downloading of the application. The steps are summarized as follows:

1. Take the node off–line.
2. Set the node applicationless.
3. Download the application into the node.
4. Reset the node.
5. Recalculate the checksum.
6. Set the node to the configured state.
7. Set the node on–line.
8. Do the final reset.

This program can be modified to the user's requirements. A copy of the program discussed in this application note is enclosed in the folder entitled `prog3120.zip` on Motorola's Design–NET bulletin board. Design–NET can be accessed through Internet at http://motserv.indirect.com.

The name of the application program is `prog3120.nc`. It was tested using a Motorola Test Board (M143205EVK). This kit was convenient because it has a MC143150 and a MC143120 socket. This kit is not necessary to run the application program; the only requirement is that an MC143150 be connected through a network to the MC143120. The application program is to large for a MC143120.

A note on the Test Board: in order to stop the MC143120 from resetting the MC143150 when a network management reset command is sent, a minor modification was performed on the Test Board. The modification consisted of cutting the reset trace between the MC143120 and MC143150 on the bottom side of the test board, then placing a diode along the trace cut with the anode on the MC143120 side. Any signal diode such as the 1N4148 will work. The diode prevents the MC143120 reset pulse from reaching the MC143150.

Downloading an MC143120 Application

To use the `prog3120.nc` program:

1. Compile (and debug if changes were made) to `prog3120.nc` program using the LONBUILDER Developer's Workbench. NOTE: This is an MC14**3150** node.
2. Export the MC143120 application with the following settings:

   NEI
   Motorola S–Record fmt.
   Configure file.

   NOTE: This is for a MC14**3120** node and part of this table will be placed in the `prog3120.nc` application.
3. Re–format the NEI and paste it into the MC143150 data table under `codedata`. This is the most complicated part of this procedure.

   A program entitled `alon.exe` will reformat the NEI file so that the table can be pasted directly into the 'codedata.' This program is available from the LONWORKS folder on the Design–Net bulletin board.

   Motorola assumes no liability arising out of use of this program or any other product or software described in this document. The software described in this document is provided on an "as is" basis and without warranty.
4. Re–compile the MC143150 node and export the file with the following configuration: NRI and Configured.

MC143120 Firmware Versions/Exporting NEI Files

This procedure and application were tested by exporting from LONBUILDER 2.2 software for a Version 3 MC143120. Revisions of the LONBUILDER software may change the way NEI files are written to, making `alon.exe` out–of–date. For exporting purposes, different versions of the MC143120 are not compatible with each other. As of the date of this publication, most MC143120s in the field are Version 3 with Version 4 being the latest release. New versions of the MC143120 will be released as changes to the firmware become necessary for derivative products or enhancements. During the first quarter of 1995, Motorola began production of MC143120s incorporating Version 4 firmware. The firmware in the MC143120 is masked inside the device as ROM, and changes will occur infrequently.

The user may verify which version of the NEURON CHIP he or she is working with by noting the LONBUILDER hardware properties in the field "NEURON CHIP firmware." If the field reads "0" or is empty, it means the application uses the latest LONBUILDER software. LONBUILDER Version 2.2 software defaults to MC143120 Version 3. LONBUILDER Version 3.0 software defaults to MC143120 Version 4. The reason different firmware versions will not be compatible is because the program makes calls into the firmware, and the firmware changes with revisions. For similar reasons, a program exported for an MC143120 will not be downloadable to an MC143150 and vice–versa.

The contents of the NEI file for the MC143120 will depend on how the program was exported: i.e., applicationless, unconfigured, or configured. Applicationless and unconfigured versions will contain no address information. Unconfigured and configured versions will contain an application.

MAXIMUM DATA BYTES

As shown in Appendix B of DL159/D, *LONWORKS Technology Device Data,* no more than 38 data bytes (for a worst case of 10 MHz input clock rate), should be written at one time. The 38 byte limit gives the NEURON CHIP enough time to prevent a watchdog time–out from programming too many EEPROM bytes and re–calculating the application and configuration checksums. The 38 byte limitation may be increased by calculating the checksums in a separate network management operation. 38 bytes is too large for the default input network buffer size of 42 bytes.

16 bytes can be safely written on a new MC143120. This size may be increased depending on the receiving node's clock rate (to prevent watchdog time–out if both checksums are re–calculated), and on the size of the receiving node's buffers.

Acknowledged service is used for downloading application data, and unacknowledged service for everything else. `prog3120.nc` is written so that the service type does not add to the amount of time required for the node to send the commands. `prog3120.nc` uses a timer called `load_image` to know when to send the next command. This approach is used because of the time required for the receiving node to program the EEPROM, which may take significantly more time than eliciting a response from the receiving node.

The maximum number of data bytes a packet can send to the MC143120 is limited by the EEPROM write time of the MC143120 as well as by the buffer sizes. When sending a *write memory* command and optional *recalculate checksum* command to an MC143120, make certain that there is enough time to program all the bytes in the MC143120 before the watchdog time–out occurs.

Worst case timing requires 20 ms to write an EEPROM byte, 10 ms if it is already erased. The translation program, `alon.exe`, converts the NEI file to an output file with no more than 10 data bytes in a packet. 10 data bytes x 10 ms = 100 ms. The time required to load the next command after a network management memory write command is 150 ms. This time differential allows 50 ms tolerance in the event that a message has to be resent, thus adding to the transmission and processing times at the transmitting and receiving nodes. For all other network management commands the timer is set to 100 ms. The only exception to this rule is for the last command, a final reset in which no timer is used, and after which no more traffic is generated from the network management node.

MC143120 PACKET SIZE GUIDELINES

Use the following guidelines to determine the maximum number of data bytes a packet can send to a new MC143120.

1. Listed below is the default for the input network buffers on a NEURON CHIP:

   default size = max(42, 21 + size of (largest NV))

   For a new MC143120 with no application, the default size will be 42 bytes.

2. Listed below is the equation to determine an input network buffer size:

   ```
   net_buf_in_size = max_msg_size
        + protocol_overhead + 6
   ```

   where:

   `max_msg_size` >= largest network variable or network management/ network diagnostic message addressed to the node. Explicit messages size includes data + code. Network variables use size of the network variable + 2.

   `protocol_overhead` = bytes in protocol overhead (addresses, CRC, …). Worst case is NEURON ID addressing with domain ID of 6 bytes. Range is 7 – 20 bytes.

Working backwards, if the default size = 42 bytes, with a worst case `protocol_overhead` addressing of 20 bytes, the largest data size is 16 bytes:

net_buf_in_size	=max_msg_size	+protocol_overhead	+6
42	= 16	+ 20	+ 6

If the addressing size is known and is not the worst case addressing, the `protocol_overhead` will be decreased and the `max_msg_size` increased. If no domain is used, the `max_msg_size` will be increased by six bytes.

`prog3120.nc` Final Notes

`prog3120.nc` is shown in Appendix C. It took approximately 15 seconds to download the application to the MC143120. The amount of time required will depend on the number of bytes to program. The program `prog3120.nc` program will always leave the MC143120 you are attempting to program in the configured, on–line state, even if that is unspecified in the NXE file.

`prog3120.nc` does not perform a complete verification by reading after every write command. A commercial MC143120 programmer should verify the node state after each change; be certain that the MC143120 was actually reset upon request and verify EEPROM writes by performing a memory read command after each memory write command.

APPENDIX A
S–RECORD INFORMATION

INTRODUCTION

The S–record format for output modules encodes programs and/or data files in a printable format for transportation between computer systems. This facilitates S–record editing and permits visual monitoring of the transportation process.

S–RECORD CONTENT

S–Records are character strings of several fields which identify the record type, length, memory address, code/data, and checksum. Each byte of binary data is encoded as a 2–character hexadecimal number: the first character represents the high–order 4 bits, and the second the low–order 4 bits of the byte.

The 5 fields of an S–record are:

TYPE	RECORD LENGTH	ADDRESS	CODE/ DATA	CHECKSUM

Field compositions are:

Field	Printable Characters	Contents
Type	2	S–record type – S0, S1 – S9. LonBuilder V. 2.2 software uses only S1 and S9 record types.
Record Length	2	The count of the character pairs pairs in the record, excluding the type and record length.
Address	4, 6, 8	The 2–, 3–, or 4–byte address at which the data field is to be loaded into memory. LonBuilder V 2.2 software uses only a 2 byte address. This is due to the Neuron Chip's 16 bit addressing.
Code/Data	0 – 2n	From 0 to n bytes of executable code, memory loadable data, or descriptive information. To ensure a node can talk to another node, keep the data size limited to 11 bytes. This number may be increased only if the network node's characteristics (such as number of buffers, size, and traffic) are understood.
Checksum	2	The least significant byte of the one's complement of the sum of the values represented by the pairs of characters making up the record length, address, and the code/data fields.

The record length (byte count) and checksum fields ensure accuracy of transmission.

S–RECORD TYPES

There are eight types of S–records to accommodate the various needs of the encoding, transportation, and decoding functions. LonBuilder v2.2 software uses only S1 and S9 record types.

An S–record format module may contain S–records of the following types:

Type	Description
S0	The header record for each block of S–records. The code/data field may contain any descriptive information identifying the following block of S–records. The address field is normally zeros. LonBuilder v2.2 software does not use this S–record.
S1	A record containing code/data and the 2–byte address at which the code/data is to reside. If using the NEI file format, a 1–byte data at the r/w bit address (0xf00a) should be moved to the end of the S–records before the last reset (S9 record).
S2 – S8	Not applicable to LonBuilder v2.2 software.
S9	A termination record for a block of S1 records. When encountered using the NEI file format, the node should be reset.

Typically there is only one termination record (S9) but the NEI file may use multiples of these, showing where a reset of the receiving node should be done.

APPENDIX B
TRANSPOSING AN NEI FILE TO ‘prog3120.nc’ FORMAT

A test program called `test_io.nc` was used to export to an NEI file. The node was configured for:

10 MHz, 1.25 MBPS, differential input, configured

Following is the `test_io.nei` file created:

```
S105F0080803F7
S113F02C05AC0301000000000000000000000000001B
S105F0240000E6
S109F026000000000000E0
S104F03C557A
S112F03D000000000000018400FFFFFFFFFFFF41
S123F04C0000000000000000000000000000000000000000000000000000000000000000A0
S123F06C000000000000000000000000000000000000000000000000000000000000000080
S10EF08C000000000000000000000075
S106F0973FFF0F25
S10BF00D544553545F494F00C0
S108F008F1DB01F1EE53
S112F01513F012249B222323000000FF38000471
S123F09B99FE760002713B760102714876020271557603027162760402716F760502717C10
S123F0BB76060375F14976070375F15B76080375F16D76090375F17E760A0375F18FE47537
S123F0DBF19F81188880811874808218748084187475F19F821888808118748082187480F3
S123F0FB84187475F19F84188880811874808218748084187475F19FB4081888808118741F
S123F11B80821874808418747 17AB41018888081187480821874808418747168B42018886B
S123F13B8081187480821874808418747156B44018888081187480821874808418747 1444A
S123F15BB480188880811874808218748084187471328018888181187480821874808418 16
S123F17B747121801888808118748182187480841874711080188880811874808218748161
S123F19B8418742099FF721599FE3ED9FEB40B99FE0AC34781D9FFB409D9FE711299FE3FA5
S123F1BBD9FEB40B99FE0AC34680D9FF81D9FE8099FD8003C93175F1E98099FD8003C931CB
S11EF1DB01EFFDF1CF00040000000108F09AB419D9FD3100080100000E0000E6
S9030000FC
S113F02C25AC010401000100003D000000000000BB
S9030000FC
S104F01514E2
S104F00A0100
S9030000FC
```

The contents of the S records may be analyzed by looking at the memory map structures in Appendix A in DL159 *LonWorks Technology Device Data.*

The next step is to run `alon.exe`. For simplicity, place the file to be converted (`test_io.nei`) in the same directory as `alon.exe.` This step is optional. If you know which directory the NEI file is located in, you can move to that directory through the menus.

Type	Description
`alon`	At the DOS prompt run this program. A menu comes with the heading: ‘Choose NXE File to load’.
`*.nei`	Under Name, change to *.nei. Move the cursor to highlight your NEI file. In this example it is test_io.nei.
`<enter>`	A menu comes up asking: Enter Output File Name:
`test_io.alo`	Type in an output file name that will be created.

A file is created and the program is terminated. The file created has a format similar to the one below (`test_io.alo`). Starting with the second line (in this example it is `2,221,`), paste this and the rest of the table into `prog3120.nc` under the data table called `codedata`. Also, place the number after the size: (in this example it is 733) in the index of codedata.

The second line represents the byte size of the array with the following format:

size of array = 1st number x 256 + 2nd number

For example:

2,221 ==> array size = 2 x 256 + 221 = 733 bytes.

```
size: 733
2,221,
0x02,0xF0,0x08,  8,  3,
0x0A,0xF0,0x08,241,147,  1,241,166, 84, 69, 83, 84, 95,
0x0A,0xF0,0x12, 73, 79,  0, 19, 16, 18, 36,155, 34, 35,
0x0A,0xF0,0x1C, 35,  0,  0,  0,255, 56,  0,  4,  0,  4,
0x02,0xF0,0x26,  0,  0,
0x0A,0xF0,0x28,  0,  0,  0,  0,  5,172,  3,  1,  0,  0,
0x0A,0xF0,0x32,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0x0A,0xF0,0x3C, 85,  0,  0,  0,  0,  0,  0,  1,131,  0,
0x02,0xF0,0x46,255,255,
0x0A,0xF0,0x48,255,255,255,255,  0,  0,  0,  0,  0, 63,
0x0A,0xF0,0x52,255, 15,  0,153,254,118,  0,  2,113, 59,
0x0A,0xF0,0x5C,118,  1,  2,113, 72,118,  2,  2,113, 85,
0x02,0xF0,0x66,118,  3,
0x0A,0xF0,0x68,  2,113, 98,118,  4,  2,113,111,118,  5,
0x0A,0xF0,0x72,  2,113,124,118,  6,  3,117,241,  3,118,
0x0A,0xF0,0x7C,  7,  3,117,241, 21,118,  8,  3,117,241,
0x02,0xF0,0x86, 39,118,
0x0A,0xF0,0x88,  9,  3,117,241, 56,118, 10,  3,117,241,
0x0A,0xF0,0x92, 72,228,117,241, 87,129, 24,136,128,129,
0x0A,0xF0,0x9C, 24,116,128,130, 24,116,128,132, 24,116,
0x02,0xF0,0xA6,117,241,
0x0A,0xF0,0xA8, 87,130, 24,136,128,129, 24,116,128,130,
0x0A,0xF0,0xB2, 24,116,128,132, 24,116,117,241, 87,132,
0x0A,0xF0,0xBC, 24,136,128,129, 24,116,128,130, 24,116,
0x02,0xF0,0xC6,128,132,
0x0A,0xF0,0xC8, 24,116,117,241, 87,180,  8, 24,136,128,
0x0A,0xF0,0xD2,129, 24,116,128,130, 24,116,128,132, 24,
0x0A,0xF0,0xDC,116,113,120,180, 16, 24,136,128,129, 24,
0x02,0xF0,0xE6,116,128,
0x0A,0xF0,0xE8,130, 24,116,128,132, 24,116,113,102,180,
0x0A,0xF0,0xF2, 32, 24,136,128,129, 24,116,128,130, 24,
0x0A,0xF0,0xFC,116,128,132, 24,116,113, 84,180, 64, 24,
0x02,0xF1,0x06,136,128,
0x0A,0xF1,0x08,129, 24,116,128,130, 24,116,128,132, 24,
0x0A,0xF1,0x12,116,113, 66,180,128, 24,136,128,129, 24,
0x0A,0xF1,0x1C,116,128,130, 24,116,128,132, 24,116,113,
0x02,0xF1,0x26, 48,128,
0x0A,0xF1,0x28, 24,136,129,164, 24,116,128,130, 24,116,
0x0A,0xF1,0x32,128,132, 24,116,113, 31,128, 24,136,128,
0x0A,0xF1,0x3C,129, 24,116,129,130, 24,116,128,132, 24,
0x02,0xF1,0x46,116, 47,
0x0A,0xF1,0x48,128, 24,136,128,129, 24,116,128,130, 24,
0x0A,0xF1,0x52,116,129,132, 24,116,153,255,114, 21,153,
0x0A,0xF1,0x5C,254, 62,217,254,180, 11,153,254, 10,195,
0x02,0xF1,0x66, 71,129,
0x0A,0xF1,0x68,217,255,180,  9,217,254,113, 18,153,254,
0x0A,0xF1,0x72, 63,217,254,180, 11,153,254, 10,195, 70,
0x0A,0xF1,0x7C,128,217,255,129,217,254,128,153,253,128,
0x02,0xF1,0x86,  3,201,
0x0A,0xF1,0x88, 49,117,241,161,128,153,253,128,  3,201,
0x0A,0xF1,0x92, 49,  1,239,253,241,135,  0,  4,  0,  0,
0x0A,0xF1,0x9C,  0,  1,  8,240, 84,180,200,217,253, 49,
0x02,0xF1,0xA6,  0,  8,
0x0A,0xF1,0xA8,  1,  0,  0, 14,  0,  0,  0,  0,  0,  0,
0x0A,0xF1,0xB2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0x0A,0xF1,0xBC,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0x02,0xF1,0xC6,  0,  0,
0x0A,0xF1,0xC8,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0x0A,0xF1,0xD2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0x0A,0xF1,0xDC,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0x02,0xF1,0xE6,  0,  0,
```

```
0x0A,0xF1,0xE8,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0x0A,0xF1,0xF2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0x04,0xF1,0xFC,  0,  0,  0,  0,
0x00,
0x0A,0xF0,0x2C,  5,172,  1,  4,  0,  0,  0,  0,  0,103,
0x06,0xF0,0x36,  4,  0,  0,  0,  0,  0,
0x00,
0x01,0xF0,0x15, 20,
0x01,0xF0,0x0A,  1,
0x00
```

`alon.exe` takes the NEI file, which is in Motorola's S–record format (Appendix A of this databook explains the S–record format) and converts it into a new file with the following rules:

1. The format is:

	<# of data bytes>,	<2 byte address>,	<data>
ex:	`0x02,`	`0xF0,0x08,`	`8, 3,`

where:

<# of data bytes> is between `0` and `0x0A`, in hex

<2 byte address> is in hex

<data> is between 0 – 10 bytes of data (in decimal) to be downloaded

2. Replaces S9 record with `0x00` record meaning to reset the MC143120.
3. If found, move a single byte data record at address `$f00a` to the bottom before the node is reset (a `0x00` record as in Step 1).
4. No more than 10 data bytes per record. If a number higher than 10 is encountered, a new record(s) is created.

REFERENCES

1. "Packaging Manual for ASIC Arrays" by Joellen Cascante, Motorola, Issue A 1990.
2. NEURON CHIP Test Board User's Manual, M143205EVK.

APPENDIX C

```
/*************************************************************************************

Filename:     prog3120.nc
Motorola, Inc
Disclaimer:   Motorola reserves the right to make changes to this
              software without further notice herein. Motorola
              makes no warranty, representation or guarantee regarding
              the suitability of this software for any particular
              purpose nor does Motorola assume any liability arising
              out of the application or use of it, and specifically
              disclaims any and all liability, including without
              limitation consequential or incidental damages.

0.1    02/16/93 original
0.2    05/27/94 change codedata using testnei.alo
0.2    06/01/94 change time to pulse IO1

Description:      When receive a service pin message, download the table
                  under 'codedata.' 'codedata' is only set up to program
                  a 3120 version 3.

I/O inputs:       none
I/O outputs:      IO_1:  pulse when rx a msg_succeeds from node
                         trying to program. Then one long pulse
                         when finished.

net inputs:       none
net outputs:      none
message tags:     write_image

Memory Requirements:
ROM Usage:
       System Data                          2  bytes
       Application Code & Const Data     1288  bytes
       Library Code & Const Data            0  bytes
       Self-Identification Data             6  bytes
                                        -----
       Total ROM Requirement             1296  bytes
       Remaining ROM                    15088  bytes

EEPROM Usage:   (not necessarily in order of physical layout)
       System Data & Parameters            74  bytes
       Domain & Address Tables             20  bytes
       Network Variable Config Tables       0  bytes
       Application EEPROM Variables         0  bytes
       Library EEPROM Variables             0  bytes
       Application Code & Const Data        0  bytes
       Library Code & Const Data            0  bytes
                                        -----
       Total EEPROM Requirement            94  bytes
       Remaining EEPROM                   418  bytes

RAM Usage:      (not necessarily in order of physical layout)
       System Data & Parameters           572  bytes
       Transaction Control Blocks         122  bytes
       Appl Timers & I/O Change Events      4  bytes
       Network & Application Buffers      528  bytes
```

```
       Application RAM Variables          19  bytes
       Library RAM Variables               0  bytes
                                       -----
       Total RAM Requirement            1245  bytes
       Remaining RAM                     803  bytes
```

Required header files: none

Timing: Acknowledgments is used for network memory write commands for downloading bytes to be programmed in EEPROM, unacknowledgments otherwise. 150 ms timer is used for the memory write commands before the next packet is sent, 100 ms otherwise.

Testing: verified 3120 program working.

Notes:
1. node state is different than node mode.

Node state tells if:
a. applicationless: no application or configuration checksum
b. unconfigured: application checksum, no configuration checksum
c. configured: application and configuration checksum.
If the configuration or application checksum fails, the node goes applicationless.

Node mode tells if:
a. online
b. offline: the only when statements that work here are: reset, online, offline and wink

2. Timing:
a. Since unacknowledge is used, ‘load_image’ is a timer that tells when to set up the next packet. This may need to be increased (decreased) depending on clock rate of this node and the receiving node, baud rate, routers, # of bytes written to, to name a few.
b. To ensure that any node can be written to, less than or equal to 16 bytes is written to at a time. This program uses 10. The data book shows 38 bytes as a maximum for the 10 MHz clock rate (see data book B.1.5) when the configuration and application checksums are used.
This size may be increased if the receiving node buffers sizes are known. Depending on receiving node clock rate, and amount of RAM dedicated to buffers, increasing the number of data bytes placed in a packet can drastically increase the time to program a Neuron Chip.

3. The only time ack. service is used is loading the application. All other cases unack. service is used. For a possible noisy environment, this may want to be changed to unack. This program would really speed up if unack. is used. Currently there is a timer (load_image) that is used to know when to set up the next packet. This program could be changed so that the ack. is waited for.

4. Other possible additions:
a. Check if the node to be programmed has its’ read/write bit set.
b. Maximum number of times to resend a message.

```
c.  If the version number on the node to be programmed is the
    same as what is expected. For example, do not program a
    version 2 3120, with a version 3 exported file.
d.  Program configuration data. If configuration data is
    programmed, program it first. In addition, make sure
    your NEI file is program with the new configuration
    information. The reason the configuration is programmed
    first is because the NEI file contains these parameters
    and when it resets, it will lose communication with the
    node unless the node is already programmed to that
    data rate.
5.  NEI file contains both application and configuration data
    (data rate).
6.  Program configuration data before programming the application
    so when reset the node, the node matches the data rate in
    the application.
*****************************************************************************************/

/****************************** Compiler directives *************************************/

#pragma scheduler_reset
#pragma enable_io_pullups
#pragma num_addr_table_entries  1
#pragma one_domain
#pragma app_buf_out_priority_count  0
#pragma net_buf_out_priority_count  0

/********************************* Include files ****************************************/

#include <addrdefs.h>
#include <access.h>
#include <msg_addr.h>
#include <netmgmt.h>
#include <control.h>

/********************************** I/O Objects *****************************************/

IO_1 output oneshot clock(7) lamp;        //  everytime rx a msg_succeeds
                                          //  response from the node
                                          //  we are trying to program,
                                          //  pulse IO_1.

/******************************** Network Variables *************************************/

// none

/*********************************** Message Tags ***************************************/

msg_tag write_image;

/*********************************** Constants *****************************************/
/*
‘codedata’ is the data to be programmed. This data came from
the *.nxe file. If the first record has a data length of one byte,
then it should not be written until after all the other records
have been written. This is because it contains the read/write
protect bit which could prevent further downloading if it is set.
If a record has a data count of zero, then the node should be
reset at this point.
```

```
The structure of 'codedata' is as follows:
    1.  first 2 bytes – the length of the table.
        lenght of table = first byte times 256 + second byte.
        The length of the table is what is used to define the
        size of the array. For example, if 2,136 are the first
        two entries in the table, the array size is:
            2 x 256 + 136 = 648 ==> codedata[648].
    2.  Each record, or each data packet is put on a separate line.
        Each record begins with a data length field:
        If the data length = 0, it means to reset the neuron,
            and the record ends.
        Else if the data length is > 0, it is followed by the address
            (in two bytes) and the data bytes themselves.
    3.  The data length and address is in hex, the data in decimal.
    4.  Example #1:
<data length:1 entry><address:2 entries><data bytes: up to 10 entries>
 0x02,           0xF0,  0x08             8,3
*/

// using test_io.nei set up for a 3120, 10 MHz, 78 KBPS, differential,
// then used alon.exe to make test_io.alo
const unsigned char codedata[728] = {
2,216,
 0x02, 0xF0, 0x08,    8,    3,
 0x0A, 0xF0, 0x2C,    5,  172,    3,    1,    0,    0,    0,    0,    0,    0,
 0x06, 0xF0, 0x36,    0,    0,    0,    0,    0,    0,
 0x02, 0xF0, 0x24,    0,    0,
 0x06, 0xF0, 0x26,    0,    0,    0,    0,    0,    0,
 0x01, 0xF0, 0x3C,   85,
 0x0A, 0xF0, 0x3D,    0,    0,    0,    0,    0,    0,    1,  132,    0,  225,
 0x05, 0xF0, 0x47,  255,  255,  255,  255,  255,
 0x0A, 0xF0, 0x4C,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
 0x0A, 0xF0,  0x56    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
 0x0A, 0xF0, 0x60,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
 0x02, 0xF0, 0x6A,    0,    0,
 0x0A, 0xF0, 0x6C,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
 0x0A, 0xF0, 0x76,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
 0x0A, 0xF0, 0x80,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
 0x02, 0xF0, 0x8A,    0,    0,
 0x0A, 0xF0, 0x8C,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
 0x01, 0xF0, 0x96,    0,
 0x03, 0xF0, 0x97,   63,  255,   15,
 0x08, 0xF0, 0x0D,   84,   69,   83,   84,   95,   73,   79,    0,
 0x05, 0xF0, 0x08,  241,  219,    1,  241,  238,
 0x0A, 0xF0, 0x15,   19,  240,   18,   36,  155,   34,   35,   35,    0,    0,
 0x05, 0xF0, 0x1F,    0,  255,   56,    0,    4,
 0x0A, 0xF0, 0x9B,  153,  254,  118,    0,    2,  113,   59,  118,    1,    2,
 0x0A, 0xF0, 0xA5,  113,   72,  118,    2,    2,  113,   85,  118,    3,    2,
 0x0A, 0xF0, 0xAF,  113,   98,  118,    4,    2,  113,  111,  118,    5,    2,
 0x02, 0xF0, 0xB9,  113,  124,
 0x0A, 0xF0, 0xBB,  118,    6,    3,  117,  241,   73,  118,    7,    3,  117,
 0x0A, 0xF0, 0xC5,  241,   91,  118,    8,    3,  117,  241,  109,  118,    9,
 0x0A, 0xF0, 0xCF,    3,  117,  241,  126,  118,   10,    3,  117,  241,  143,
 0x02, 0xF0, 0xD9,  228,  117,
 0x0A, 0xF0, 0xDB,  241,  159,  129,   24,  136,  128,  129,   24,  116,  128,
 0x0A, 0xF0, 0xE5,  130,   24,  116,  128,  132,   24,  116,  117,  241,  159,
 0x0A, 0xF0, 0xEF,  130,   24,  136,  128,  129,   24,  116,  128,  130,   24,
 0x02, 0xF0, 0xF9,  116,  128,
```

```
 0x0A, 0xF0, 0xFB,  132,   24,  116,  117,  241,  159,  132,   24,  136,  128,
 0x0A, 0xF1, 0x05,  129,   24,  116,  128,  130,   24,  116,  128,  132,   24,
 0x0A, 0xF1, 0x0F,  116,  117,  241,  159,  180,    8,   24,  136,  128,  129,
 0x02, 0xF1, 0x19,   24,  116,
 0x0A, 0xF1, 0x1B,  128,  130,   24,  116,  128,  132,   24,  116,  113,  122,
 0x0A, 0xF1, 0x25,  180,   16,   24,  136,  128,  129,   24,  116,  128,  130,
 0x0A, 0xF1, 0x2F,   24,  116,  128,  132,   24,  116,  113,  104,  180,   32,
 0x02, 0xF1, 0x39,   24,  136,
 0x0A, 0xF1, 0x3B,  128,  129,   24,  116,  128,  130,   24,  116,  128,  132,
 0x0A, 0xF1, 0x45,   24,  116,  113,   86,  180,   64,   24,  136,  128,  129,
 0x0A, 0xF1, 0x4F,   24,  116,  128,  130,   24,  116,  128,  132,   24,  116,
 0x02, 0xF1, 0x59,  113,   68,
 0x0A, 0xF1, 0x5B,  180,  128,   24,  136,  128,  129,   24,  116,  128,  130,
 0x0A, 0xF1, 0x65,   24,  116,  128,  132,   24,  116,  113,   50,  128,   24,
 0x0A, 0xF1, 0x6F,  136,  129,  129,   24,  116,  128,  130,   24,  116,  128,
 0x02, 0xF1, 0x79,  132,   24,
 0x0A, 0xF1, 0x7B,  116,  113,   33,  128,   24,  136,  128,  129,   24,  116,
 0x0A, 0xF1, 0x85,  129,  130,   24,  116,  128,  132,   24,  116,  113,   16,
 0x0A, 0xF1, 0x8F,  128,   24,  136,  128,  129,   24,  116,  128,  130,   24,
 0x02, 0xF1, 0x99,  116,  129,
 0x0A, 0xF1, 0x9B,  132,   24,  116,   32,  153,  255,  114,   21,  153,  254,
 0x0A, 0xF1, 0xA5,   62,  217,  254,  180,   11,  153,  254,   10,  195,   71,
 0x0A, 0xF1, 0xAF,  129,  217,  255,  180,    9,  217,  254,  113,   18,  153,
 0x02, 0xF1, 0xB9,  254,   63,
 0x0A, 0xF1, 0xBB,  217,  254,  180,   11,  153,  254,   10,  195,   70,  128,
 0x0A, 0xF1, 0xC5,  217,  255,  129,  217,  254,  128,  153,  253,  128,    3,
 0x0A, 0xF1, 0xCF,  201,   49,  117,  241,  233,  128,  153,  253,  128,    3,
 0x02, 0xF1, 0xD9,  201,   49,
 0x0A, 0xF1, 0xDB,    1,  239,  253,  241,  207,    0,    4,    0,    0,    0,
 0x0A, 0xF1, 0xE5,    1,    8,  240,  154,  180,   25,  217,  253,   49,    0,
 0x07, 0xF1, 0xEF,    8,    1,    0,    0,   14,    0,    0,
 0x00,
 0x0A, 0xF0, 0x2C,   37,  172,    1,    4,    1,    0,    1,    0,    0,   61,
 0x06, 0xF0, 0x36,    0,    0,    0,    0,    0,    0,
 0x00,
 0x01, 0xF0, 0x15,   20,
 0x01, 0xF0, 0x0A,    1,
 0x00,
};

/********************************** Globals **************************************/

NM_service_pin_msg svc_pin_msg;                    // copy of service pin message
const char *image_ptr;      // points to 1 byte in ‘codedata’. This is
                            // next byte to write
const char *last_image_ptr;

enum {
           set_offline,     // unack
           set_appless,     // unack
           load_info,       // if size field = 0 (reset node): unack,
                            // else ack (typically it will be this)
           reset_node,      // unack
           recalculate_cs,  // ack
           set_config,      // unack
           set_online,      // unack
           final_reset      // unack
      } image_state = set_offline;
```

```
/********************************* Timers *************************************/

mtimer load_image;      // when to send ready next packet

/******************************** Functions ***********************************/

void config_message(service_type type, int code) {
      msg_out.priority_on=FALSE;
      msg_out.authenticated=FALSE;
      msg_out.dest_addr.nrnid.type=NEURON_ID;
      msg_out.dest_addr.nrnid.domain=0;
      msg_out.dest_addr.nrnid.subnet=0;
      msg_out.service=type;
      memcpy(msg_out.dest_addr.nrnid.nid,svc_pin_msg.neuron_id,6);
      msg_out.dest_addr.nrnid.retry=15;
      msg_out.dest_addr.nrnid.tx_timer=10;
      msg_out.code=code;
      msg_send();
}

/********************************** Reset ***********************************/

/* when (reset) {
      ***not needed***
} */

/************************* Priority When Clauses ****************************/

// none

/*********************** Non–Priority When Clauses **************************/

when ( msg_arrives ( NM_service_pin | NM_opcode_base) )
{
      memcpy( &svc_pin_msg, msg_in.data, sizeof(NM_service_pin_msg) );
            // get local copy of service pin message
      image_state = set_offline;  // 1st step in the sequence
      image_ptr = &codedata[0];   // point to start in array
      load_image = 1;    // set timer to start sending image 1 ms from now
}

when(timer_expires(load_image)) {
char count;
char size;
      if(msg_alloc()) {  // returns true if the Neuron chip has
                         // buffers for the message. This way
                         // will never go into pre–emption mode.
      msg_out.tag = write_image;
      switch(image_state) {
         case set_offline:
           msg_out.data[0] = 0;     // set mode state:  appl_offline
           config_message(UNACKD,NM_set_node_mode | NM_opcode_base);
           break;

         case set_online:
           msg_out.data[0] = 1;     // set mode state:  appl_online
           config_message(UNACKD,NM_set_node_mode | NM_opcode_base);
           break;

         case set_appless:
           msg_out.data[0] = 3;     // set mode state:  change
           msg_out.data[1] = 3;     // if [0] = 3, need to
                                    // tell which option:  appl'ess
           config_message(UNACKD,NM_set_node_mode | NM_opcode_base);
           break;
```

```
        case set_config:
          msg_out.data[0] = 3;     // set node state:
          msg_out.data[1] = 4;     // configured, online
          config_message(UNACKD,NM_set_node_mode | NM_opcode_base);
          break;

        case load_info:
          last_image_ptr=image_ptr;
          if(*image_ptr == 0) {    // if size field is zero,
                                   // reset the node
             image_ptr++;     // point to next byte in table
             msg_out.data[0] = 2;      // reset node
             config_message(UNACKD,NM_set_node_mode|NM_opcode_base);
          }
          else {
             msg_out.data[0] = 0;
             msg_out.data[3] = size = *image_ptr;
             image_ptr++;     // point to next byte in table
             msg_out.data[1] = *image_ptr;      // low byte of address
             image_ptr++;     // point to next byte in table
             msg_out.data[2] = *image_ptr;      // high byte of address
             image_ptr++;     // point to next byte in table
             msg_out.data[4] = 0; // do not recalculate checksum
             for (count = 0; count < size; count++) {
                   msg_out.data[ 5+count ] = *image_ptr;
                   image_ptr++;    // point to next byte in table
             }
             config_message(ACKD, NM_write_memory | NM_opcode_base);
          }
          break;

        case reset_node:
        case final_reset:
          msg_out.data[0] = 2;     // reset node
          config_message(UNACKD, NM_set_node_mode | NM_opcode_base);
          break;

        case recalculate_cs:
          msg_out.data[0] = 1;     // both checksums
          config_message(UNACKD, NM_checksum_recalc | NM_opcode_base);
          break;
        }
    }
    else load_image = 1;      // if msg_alloc fails, will try in again
                              // in 1 ms.
}

When(msg_fails(write_image)) {
    if(image_state == load_info) image_ptr = last_image_ptr;
    load_image = 1;
}

when(msg_succeeds(write_image)) {
    io_out(lamp,1000);        // indicate finished with write
    switch(image_state) {
        case set_offline:
        case set_appless:
        case reset_node:
        case recalculate_cs:
        case set_config:
        case set_online:
```

```
          image_state++;      // next command in sequence
          load_image = 100;        // when to send next packet. These
                                   // commands typically do not take as long
                                   // as a write message. Although, according
                                   // to the data book, a reset can take
                                   // up to 18 seconds worst case. See
                                   // appendix B under application
                                   // downloading.
          break;

        case load_info:
          if (( (long unsigned) image_ptr) >=
             ( ( * (const long unsigned *) &codedata) +
             ( (long unsigned) &codedata) ))
                  image_state = reset_node;
          load_image = 150;        // give more time for up to
                                   // 10 byte xfers
          break;

        case final reset:     // installation complete;
          io_out(lamp,50000);          // indicate finished with process
          break;
      }
}

when(msg_arrives) {
}

when(resp_arrives) {
}

when(msg_completes) {
}
```

APPENDIX D

```
/*******************************************************************************************

Filename:     comm1.nc
Motorola, Inc
Disclaimer:   Motorola reserves the right to make changes to this
              software without further notice herein. Motorola
              makes no warranty, representation or guarantee regarding
              the suitability of this software for any particular
              purpose nor does Motorola assume any liability arising
              out of the application or use of it, and specifically
              disclaims any and all liability, including without
              limitation consequential or incidental damages.
0.1   06/07/94    get svc pin msg, program to 78 kbps, 5 mhz

Description:      When get svc pin msg, program to 78 kbps, 5 MHz.
                  If using a gizmo 2 or 3 will sound buzzer,
                  turn on IO_1 red LED and display ID. Pushing
                  button corresponding to IO_7 will scroll to
                  next part on service pin message and turn off
                  red LED. When get to beginning of service pin
                  message red LED goes back on.

I/O inputs:       IO_7 input bit IO_left_sw;
                  IO_8 neurowire master select (IO_2) IO_display;

I/O outputs:      IO_0 output frequency clock (7) IO_sound = 0;
                  IO_1 output bit IO_red_led = OFF;
                  IO_2 output bit IO_display_select = 1; active low

net inputs:       none
net outputs:      none

Memory Requirements:  for the 3150
Link Memory Usage Statistics:

ROM Usage:
      System Data                          2  bytes
      Application Code & Const Data      277  bytes
      Library Code & Const Data            0  bytes
      Self-Identification Data             6  bytes
                                       -----
      Total ROM Requirement              285  bytes
      Remaining ROM                    16099  bytes

EEPROM Usage:   (not necessarily in order of physical layout)
      System Data & Parameters            74  bytes
      Domain & Address Tables            105  bytes
      Network Variable Config Tables       0  bytes
      Application EEPROM Variables         0  bytes
      Library EEPROM Variables             0  bytes
      Application Code & Const Data        0  bytes
      Library Code & Const Data            0  bytes
                                       -----
      Total EEPROM Requirement           179  bytes
      Remaining EEPROM                   333  bytes

RAM Usage:      (not necessarily in order of physical layout)
      System Data & Parameters           572  bytes
      Transaction Control Blocks         140  bytes
```

```
       Appl Timers & I/O Change Events      7  bytes
       Network & Application Buffers      792  bytes
       Application RAM Variables           19  bytes
       Library RAM Variables                0  bytes
                                        -----
       Total RAM Requirement             1530  bytes
       Remaining RAM                      518  bytes

Required header files:  #include <addrdefs.h>
                        #include <access.h>
                        #include <netmgmt.h>
                        #include <msg_addr.h>

Timing:

Testing:

Notes:
1.   This program only shows that it is possible to change
     the communication parameters. Refer to Appendix B
     in the data book under 'Configuration Changes' for
     the proper sequence to take.

*******************************************************************************************/

/****************************** Compiler directives *************************************/

#pragma enable_io_pullups
#pragma scheduler_reset

#define PRESSED 0       /* Switch pressed */
#define ON 0            /* Led is on if given a 0 */
#define OFF 1           /* Led is off if given a 1 */

/********************************* Include files *****************************************/

#include <addrdefs.h>
#include <access.h>
#include <netmgmt.h>
#include <msg_addr.h>

/********************************** I/O Objects *****************************************/

IO_0 output frequency clock(7) IO_sound = 0;
IO_1 output bit IO_red_led = OFF;
IO_2 output bit IO_display_select = 1;      /* active low */
IO_7 input bit IO_left_sw;
IO_8 neurowire master select (IO_2) IO_display;

/******************************** Network Variables *************************************/

// none

/*********************************** Message Tags ****************************************/

// none

/*********************************** Constants *******************************************/

// none

/************************************** Globals *****************************************/

NM_service_pin_msg svc_pin_msg;        /* copy of service pin message */
unsigned char dd_config = 0x01;   /* 8 bits=>display config reg */
unsigned char dd_data[3];        /* 24 bits=>display data reg */
unsigned int display_byte_index = 0;
// unsigned int flash_static = 0;      // for test board, this means on
```

```
/********************************** Timers ***************************************/

mtimer flash_LED;       // flash IO3–6 if bad, on if good

/********************************* Functions *************************************/

void config_message(service_type type, int code) {
      msg_out.priority_on=FALSE;
      msg_out.authenticated=FALSE;
      msg_out.dest_addr.nrnid.type=NEURON_ID;
      msg_out.dest_addr.nrnid.domain=0;
      msg_out.dest_addr.nrnid.subnet=0;
      msg_out.service=type;
      memcpy(msg_out.dest_addr.nrnid.nid,svc_pin_msg.neuron_id,6);
      msg_out.dest_addr.nrnid.retry=15;
      msg_out.dest_addr.nrnid.tx_timer=10;
      msg_out.code=code;
      msg_send();
}

/********************************* Reset ***********************************/

/* when (reset)
{
      io_out (IO_display, &dd_config, 8);
      dd_data[0] = 0x80;
      dd_data[1] = 0x00;
      dd_data[2] = 0x00;
      io_out (IO_display, dd_data, 24);
}

/************************* Priority When Clauses ****************************/

// none

/*********************** Non–Priority When Clauses **************************/

when ( msg_arrives ( NM_opcode_base + NM_service_pin ) )     // code 0x7f
{
      io_out(IO_red_led, ON);
      io_out ( IO_sound, 30);
      memcpy (&svc_pin_msg, msg_in.data, sizeof (NM_service_pin_msg ) );
      dd_data[0] = 0x80; // update display with service pin id
      display_byte_index = 0;
      dd_data[1] = svc_pin_msg.neuron_id[0]; // most significant nibble
      dd_data[2] = svc_pin_msg.neuron_id[1]; // next nibble
      io_out (IO_display, dd_data, 24);// output to display

// program neuron chip with 78 KBPS, 5 MHz
      msg_out.data[0] = 2;  // config_relative
      msg_out.data[1] = 0;  // comm_clock, input_clock
      msg_out.data[2] = 8;  // 1:2 is a long,
      msg_out.data[3] = 1;  // # of bytes to write
      msg_out.data[4] = 4;  // cnfg_cs_recalc
      msg_out.data[5] = 0x1c;     // 5 MHz, 78 KBPS
      config_message(UNACKD, NM_write_memory | NM_opcode_base);
}

when (io_changes(IO_left_sw) to PRESSED)
{
      io_out(IO_red_led, OFF);
      io_out ( IO_sound, 0 );
      display_byte_index = display_byte_index + 2;
      if (display_byte_index >= NEURON_ID_LEN)              /* 6 bytes */
```

```
{
      display_byte_index = 0;
      io_out(IO_red_led, ON);
}
```