APPENDIX A
Neuron CHIP DATA STRUCTURES

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

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

THE SYSTEM IMAGE

This contains the LonTalk protocol, the Neuron C runtime library, and the task scheduler. In the MC143120, this software is in the on-chip 10K ROM. In the MC143150, this software is in an external ROM. For the MC143150, this software is provided as part of the LonBuilder and NodeBuilder software. Using LonBuilder or NodeBuilder software, the user can produce Intel Hex or Motorola S-record files containing the system image so that EPROM or flash devices may be programmed. Flash memory support started with version 6 of the MC143150.

THE APPLICATION IMAGE

This contains the object code generated by the Neuron C compiler from the user's application program, along with other application-specific parameters. These parameters may be queried by a network management node. They include:

- Network variable fixed and self-identification data
- Network variable external interface data (XIF file)
- Program ID string
- Optional self-identification and self-documentation data
- Number of address table entries
- Number of domain table entries
- Number and size of network buffers
- Number and size of application buffers
- Number of receive transaction records
- Input clock speed of target Neuron Chip
- Transceiver type and bit rate

In the MC143150, the application image is typically programmed into an external ROM, or downloaded over the network to EEPROM or flash memory. In the MC143120, the application image is downloaded into the on-chip EEPROM memory. LonBuilder and NodeBuilder software supports the creation of application images.

The application image data structures described here are:

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

THE NETWORK IMAGE

This contains the address assignments of the LONWORKS node, the binding information connecting network variables and message tags between the nodes in the network, parameters of the LonTalk protocol that may be set at installation time, and configuration variables of the application program. A network management node typically downloads the network image over the network into on-chip EEPROM memory when the node is installed. For simple networks, a node can update its own network image.

This section is intended for application programmers who need to understand the internal data structures of the Neuron Chip that are used for address assignment, binding, and configuration. The Neuron C application program running on the Neuron Chip may access these data using run-time library calls and declarations for the purpose of node self-installation. In the LonBuilder or NodeBuilder Neuron C development environment, function prototypes and declarations are to be found in the files `...\INCLUDE\ACCESS.H` and `...\INCLUDE\ADDRDEFS.H`.

These data structures may also be accessed by network management messages received over the network from a network management node (see Appendix B). For network management nodes running on host computers, the LonManager application programmer's interface API provides convenient high-level access to these data structures from a host-based application program. See the *LonManager API Developer's Guide* for more details. For network management tools running on any host, the LonManager NSS-10 Network Services module provides similar services. See the *LonManager API Programmer's Guide* and the *NSS-10 Developer's Guide* for more details.

The network image data structures described here are:

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

ON-CHIP EEPROM LAYOUT

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

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

Table A–1. Data Structure and Memory Location

Structure	Location	
	MC143120B1 MC143150FU/FU1	MC143120E1/E2 MC143150B1FU
Fixed read-only data structure	0xF000	0xF000
Configuration data structure	0xF024	0xF029
Boot ID	0xF1FE (Neuron 3150 Chip only)	0xF1FE (Neuron 3150 Chip only)
Domain table	0xF03D	0xF042
Address table	0xF03D + 15 (decimal) bytes per domain	0xF042 + 15 (decimal) bytes per domain
Network variable configuration table	Address table + 5 bytes per address	Address table + 5 bytes per address

NOTES:
1. MC143120B1 contains version 4 firmware. MC143120DW contains version 3 firmware. MC143120E2 contains version 6 firmware. MC143150 firmware version depends on the software which exported it.
2. Refer to the Reset Process and Timing section for more information on the Boot ID.

The network variable fixed table, the SNVT descriptor table, and the various other parts of the application image are located by the LonBuilder linker at addresses which depend on the memory map of the node. For an MC143150-based node, it is possible to specify that the off-chip ROM is just 16K bytes (64 pages), which will force the whole application image into internal EEPROM.

RAM LAYOUT

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

- *System Base RAM.* This memory contains stacks and system control variables. The amount of this fixed memory is a function of the Neuron Chip model; i.e., MC143120 and MC143150 and firmware version. For versions 4 and 6, the values are 454 and 636 bytes respectively. The portion of this memory allocated to the application data and return stacks are 114 and 232 bytes respectively.
- *Application Timers.* The number of application software timers times 4 bytes per timer.
- *Receive Transactions.* The number of receive transactions times 13 bytes per transaction record.
- *Transmit Transactions.* The number of transmit transactions times 18 bytes per transaction record. The number of transmit transactions is two if priority buffers are present, one otherwise.
- *Buffers.* Buffers are allocated in the following order:
  Application input buffers
  Application output buffers
  Application output priority buffers
  Network input buffers
  Network output buffers
  Network output priority buffers

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

- *I/O Changes Array.* The number of I/O changes events in the program times 3 bytes per event.
- *Application Data.* The application data is allocated from the end of RAM downward towards the end of system RAM. Note that the firmware is not aware of where the application data resides. It relies on the linker to correctly partition the RAM. Thus, increasing system RAM requirements after linking has occurred must be done with caution. To aid such an effort, a node's external interface file contains the amount of total RAM available for system usage.

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

A.1 FIXED READ-ONLY DATA STRUCTURE

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

Declarations from `ACCESS.H` and `ADDRDEFS.H`

```
#define NEURON_ID_LEN 6
#define ID_STR_LEN 8
typedef struct {
     unsigned   neuron_id[ NEURON_ID_LEN ];        // offset 0x00
     unsigned   model_num;                         // offset 0x06
     unsigned                              : 4;
     unsigned   minor_model_num            : 4;    // offset 0x07
const    nv_fixed_struct * nv_fixed;               // offset 0x08
     unsigned   read_write_protect         : 1;    // offset 0x0A
     unsigned                              : 1;
     unsigned   nv_count                   : 6;
const    snvt_struct * snvt;                       // offset 0x0B
     unsigned   id_string[ ID_STR_LEN ];           // offset 0x0D
     unsigned   NV_processing_off          : 1;    // offset 0x15
     unsigned   two_domains                : 1;
     unsigned   explicit_addr              : 1;
     unsigned                              : 5;
     unsigned   address_count              : 4;    // offset 0x16
     unsigned                              : 4;
     unsigned                              : 4;    // offset 0x17
     unsigned   receive_trans_count        : 4;
     unsigned   app_buf_out_size           : 4;    // offset 0x18
     unsigned   app_buf_in_size            : 4;
     unsigned   net_buf_out_size           : 4;    // offset 0x19
     unsigned   net_buf_in_size            : 4;
     unsigned   net_buf_out_priority_count : 4;    // offset 0x1A
     unsigned   app_buf_out_priority_count : 4;
     unsigned   app_buf_out_count          : 4;    // offset 0x1B
     unsigned   app_buf_in_count           : 4;
     unsigned   net_buf_out_count          : 4;    // offset 0x1C
     unsigned   net_buf_in_count           : 4;
     int        reserved1 [6];                     // offset 0x1D
     unsigned                              : 6;    // offset 0x23
     unsigned   tx_by_address              : 1;
     unsigned   idempotent_duplicate       : 1;
```

```
} read_only_data_struct;
      // The following addendum to the read-only data
      // structure is available only in the 3150 firmware
      // version 6 and later (3120 version 4)
typedef struct {
      unsigned                              : 2;   // offset 0x24
      unsigned   alias_count                : 6;   // offset 0x25
      unsigned   msg_tag_count              : 4;
      unsigned                              : 4;
      int        reserved2[3];
} read_only_data_struct_2;

const read_only_data_struct read_only_data;
const read_only_data_struct_2 read_only_data_2;
```

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

A.1.1 Read-Only Structure Field Descriptions

```
unsigned   neuron_id[NEURON_ID_LEN];  // offset 0x00
```

This field is a 6-byte ID assigned by the manufacturer of the Neuron Chip which is unique to each Neuron Chip manufactured. Hardware prevents this field from being written after manufacture. It may be read over the network using the Query ID network management message. It is also part of the unsolicited Service Pin network management message.

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

These are two fields that specify the model of the Neuron Chip. The encoding of the model number field is: MC143150 = 0, MC143120 = 8, 3120E1 = 9, MC143120E2 = 0x0A

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

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

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

This bit specifies that parts of the Neuron Chip memory may not be read or written over the network with the *Read Memory* and *Write Memory* network management messages (Section B.1.5). The application program may set this bit with the Neuron C compiler directive `#pragma read_write_protect`. If this bit is set, only the Read-only Structure (Section A.1), the SNVT Structures (Section A.5), the Configuration Structure (Section A.6), and the application data area may be read, and only the Configuration Structure may be written. The write-protected data includes the `read_write_protect` bit itself, so that once set, the bit may not be reset over the network.

```
unsigned   nv_count         : 6;
```

This field specifies the number of network variables declared in the application program running on this node (0 – 62). Each element of a network variable array is counted separately. If this node is a LONWORKS

network interface (ex: Microprocessor Interface Program) using host network variable selection, this field is zero.

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

This field is a pointer to the data structure that gives self-identification information for the network variables (see Section A.5). If the self-identification information is not present, this is a null (0) pointer. If the Neuron Chip is a LONWORKS network interface (ex: Microprocessor Interface Program) using host network variable selection, this pointer is 0xFFFF. The self-identification information may be suppressed with the Neuron C compiler directive `#pragma disable_snvt_si`.

```
unsigned   id_string[ ID_STR_LEN ];   // offset 0x0D
```

This field contains an 8-byte program identifying information as specified in either of the Neuron C compiler directives:

```
#pragma set_id_string "ssssssss"
```

or

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

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

- The first four bits are the format code (0x8 – 0xF). Format 8 is reserved for devices passing the LONMARK conformance review and format 9 would be used by devices with standard program IDs that have not yet passed the conformance review. Formats 0xA – 0xF are reserved for future use.
- The next 20 bits are the manufacturer code — assigned to manufacturers when they become members of the LONMARK program.
- The next 16 bits are the device class — drawn from a registry of pre-defined class definitions defined by the LONMARK program.
- The next 16 bits are the device subclass — drawn from a registry of pre-defined subclass definitions defined by the LONMARK program.
- The last eight bits are the manufacturer-assigned model number.

If the program ID string is not specified by either of the compiler directives, then it contains the name of the Neuron C source file. The program ID string may be read over the network using the Query ID network management message. It is also part of the unsolicited Service Pin network management message.

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

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

```
unsigned   two_domains     : 1;
```

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

```
unsigned   explicit_addr   : 1;
```

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

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

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

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

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

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

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

Field Value	Buffer Size
2	20
3	21
4	22
5	24
6	26
7	30
8	34
9	42
10	50
11	66
12	82
13	114
14	146
15	210
0	255

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

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

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

```
unsigned   tx_by_addr       : 1;
```

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

```
unsigned   idempotent_duplicate : 1;
```

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

```
unsigned   alias_count      : 6;
```

This field specifies the number of entries in the network variable alias table. If this node is a LONWORKS network interface using host network variable selection, this field is zero. Maximum value for this field is 62.

```
unsigned   msg_tag_count    : 4;
```

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

A.2 THE DOMAIN TABLE

This table defines the domains to which this node belongs. It is located in EEPROM, and is part of the network image written during node installation.

Declarations from ACCESS.H and ADDRDEFS.H

```
#define AUTH_KEY_LEN 6
#define DOMAIN_ID_LEN 6
typedef struct {
     unsigned   id[ DOMAIN_ID_LEN ];       // offset 0x00
     unsigned   subnet;                    // offset 0x06
     unsigned        : 1;                  // clone_domain_bit
     unsigned   node  : 7;                 // offset 0x07
     unsigned   len;                       // offset 0x08
     unsigned   key[ AUTH_KEY_LEN ];       // offset 0x09
```

```
} domain_struct;

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

The application program may read or write any entry in this table using the access routines `access_domain()`, `update_domain()` and `update_clone_domain()`. Normally, the function `update_domain()` should be used to write into the node's domain table. The function `update_clone_domain()` can be used for self-installation applications where it is not necessary to assign unique subnet and node IDs to each node. It will allow the node to receive messages from another node with the same subnet and node ID, but it will prevent the node from receiving any message addressed with subnet/node addressing mode in that domain. If `update_clone_domain()` is used, the node can only receive messages addressed with Neuron ID, group unacknowledged or broadcast addressing modes.

The domain table consists of up to two entries, each 15 bytes in length. The default number of entries is two, which may be overridden by the Neuron C compiler directive `#pragma one_domain`. The entries in this table may be written and read over the network with the Update Domain, Leave Domain, and Query Domain network management messages.

All packets out on the network have the `clone_domain_bit` set. When cloning domains, this bit is cleared in the domain table guaranteeing a packet, sent by a node with the same address, will not match the node's address.

A.2.1 Domain Table Field Descriptions

```
unsigned   id[ DOMAIN_ID_LEN ];
```

Each domain in a LONWORKS network has a unique ID of zero, one, three, or six bytes in length. If the ID is shorter than six bytes, it is left justified in this field. In the LonBuilder software, the user specifies the size and value of the domain ID when creating the domain object. In the NodeBuilder software, the subnet is assigned automatically when the device is created.

```
unsigned   subnet;
```

This field specifies the ID of the subnet within this domain to which this node belongs. A subnet ID may be in the range 1 – 255 for each domain. In the development environment, this is assigned automatically when the subnet object is created. Zero is an invalid subnet ID.

```
unsigned        : 1;
```

If no clone domain is used, then the `clone_domain_bit` is set. A bit in every outgoing network packet has this bit reset. When cloning domains, this bit is cleared in the domain table guaranteeing it will not match the incoming packets, making the address different from the receiving node.

```
unsigned   node  : 7;
```

This field specifies the ID of the node within this subnet (1 – 127). In the development environment, this is assigned automatically when the node specification object is created. The value zero means that this domain table entry is not in use.

```
unsigned   len;
```

This field specifies the length of the domain ID in bytes (zero, one, three, or six). The value 0xFF (255) means that this domain table entry is not in use.

```
unsigned   key[ AUTH_KEY_LEN ];
```

This field specifies the six-byte authentication key to be used in this domain for authenticated transactions. This key must match the key of all the other nodes on this domain that participate in authenticated transactions with this node. In the development environment, the user specifies the key in the node specification screen. NodeBuilder software uses a fixed key that can be changed by a network management tool. The authentication key may be incremented over the network using the *Update Key* network management message.

A.3 THE ADDRESS TABLE

This table defines the network addresses to which this node may send implicitly-addressed messages and network variables. It also defines the groups to which this node belongs. It is located in EEPROM, and is part of the network image written during node installation.

Declarations from ACCESS.H and ADDRDEFS.H

```
typedef enum { UNBOUND, SUBNET_NODE, NEURON_ID, BROADCAST } addr_type;
typedef union {
     group_struct          gp;
     snode_struct          sn;
     bcast_struct          bc;
     turnaround_struct     ta;
     } address_struct;
const address_struct *access_address( int index );
update_address( const address_struct *address_entry, int index );
int addr_table_index( message_tag_name );
```

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

Each entry may be in one of five formats: group address, subnet/node address, broadcast address, turnaround address, or not in use. A group address is used for multicast addressing, when a network variable or message tag is used in a connection having more than two members. A subnet/node address is used for unicast addressing, when an output network variable or message tag is used in a connection with one other node. Broadcast address is not used by the LonBuilder, LonMaker, LonManager API, or NSS-10 binders when they create network variable or message tag connections. However, broadcast addressing is supported in the protocol, and may be used with explicit addressing. A turnaround address is used for network variables that are only bound to other network variables in the same node, and not to any network variables on other nodes. Destination addresses in the unique 48-bit Neuron Chip ID format are never used in the address table, but they may be used as destination addresses in explicitly addressed messages. For completeness, this format is described below. The declaration of this format is in the Neuron C include file MSG_ADDR.H.

In the development environment, entries in the address table are created when the network manager loads the network image into the node. The entries in this table may be read and written over the network with the Query Address and Update Address Table network management messages. An entry using group

address format may be updated over the network with the Update Group Address Data network management message.

The first byte in an address table entry specifies the format of the entry:

0	not in use/turnaround format
1	(subnet, node) format
3	broadcast format
128 – 255	group format

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

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

A.3.1 Declaration of Group Address Format

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

A.3.2 Group Address Field Descriptions

```
unsigned   type       : 1;
```

This bit is one for a group address, zero for any of the other formats.

```
unsigned   size       : 7;
```

This field specifies the size of the group (2 – 64). The size of group includes the sender of the message. If this field is zero, then the group is of unlimited size, and unacknowledged or unacknowledged-repeated service must be used.

```
unsigned   domain     : 1;
```

This field specifies the index into the domain table for this address (zero or one).

```
unsigned   member     : 7;
```

This field specifies the member ID of this node within this group (0 – 63). A group of unlimited size has zero in this field. The member ID is used in acknowledgments to allow the sender of an acknowledged multicast message to keep track of which nodes have responded.

```
unsigned   group      : 8;
```

This field specifies the ID of this group within this domain. A group ID may be in the range of 0 – 255. In the development environment, the group ID is allocated by the binder.

A.3.3 Declaration of Subnet/Node Address Format

```
typedef struct {
     addr_type  type;                 // offset 0x00
     unsigned   domain     : 1;       // offset 0x01
     unsigned   node       : 7;
     unsigned   rpt_timer  : 4;       // offset 0x02
     unsigned   retry      : 4;
     unsigned   rcv_timer  : 4;       // offset 0x03
     unsigned   tx_timer   : 4;
     unsigned   subnet     : 8;       // offset 0x04
} snode_struct;
```

A.3.4 Subnet/Node Address Field Descriptions

```
addr_type  type;
```

This field contains the value `SUBNET_NODE (1)`.

```
unsigned   domain     : 1;
```

This field specifies the index into the domain table for this address (zero or one).

```
unsigned   node       : 7;
```

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

```
unsigned   subnet     : 8;
```

This field specifies the subnet ID (1 – 255) within the specified domain. Zero is not a valid subnet ID.

A.3.5 Declaration of Broadcast Address Format

```
typedef struct {
     addr_type  type;                 // offset 0x00
     unsigned   domain     : 1;       // offset 0x01
     unsigned              : 1;
     unsigned   backlog    : 6;       // offset 0x01
     unsigned   rpt_timer  : 4;       // offset 0x02
     unsigned   retry      : 4;
     unsigned              : 4;       // offset 0x03
     unsigned   tx_timer   : 4;
     unsigned   subnet     : 8;       // offset 0x04
} bcast_struct;
```

A.3.6 Broadcast Address Field Descriptions

```
addr_type  type;
```

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

```
unsigned   domain     : 1;
```

This field specifies the index into the domain table for this address (zero or one).

```
unsigned   backlog    : 6;
```

This field specifies an estimate of the channel backlog that would be created by an acknowledged or request/response message broadcast using this address. It should be set to the expected number of

acknowledgments or responses. For example, this might be the worst case number of nodes expected to respond to a QUERY_ID message on a channel. If this is unknown, this field can be set to zero in which case a backlog of 15 is assumed.

```
unsigned   subnet     : 8;
```

This field specifies the subnet number (1 – 255) within the specified domain. The message is delivered to all nodes in this subnet. If the subnet number is zero, the message is delivered to all nodes in the domain. If Request/Response or Acknowledged service is used with a broadcast address, the transaction completes as soon as the first response or acknowledgment is received. Subsequent responses or acknowledgments are discarded.

A.3.7 Declaration of Turnaround Address Format

```
typedef struct {
     addr_type  type;                 // offset 0x00
     unsigned   turnaround;           // offset 0x01
     unsigned   rpt_timer; : 4;       // offset 0x02
     unsigned   retry      : 4;
     unsigned              : 4;       // offset 0x03
     unsigned   tx_timer;  : 4;
} turnaround_struct;
```

A.3.8 Turnaround Address Field Descriptions

```
addr_type  type;
```

Contains the value `UNBOUND (0)`.

```
unsigned   turnaround;
```

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

A.3.9 Declaration of Neuron ID Address Format

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

```
typedef struct {
     addr_type  type;                 // offset 0x00
     unsigned   domain     : 1;       // offset 0x01
     unsigned              : 7;
     unsigned   rpt_timer  : 4;       // offset 0x02
     unsigned   retry      : 4;
     unsigned              : 4;       // offset 0x03
     unsigned   tx_timer   : 4;
     unsigned   subnet     : 8;       // offset 0x04
     unsigned   nid[ NEURON_ID_LEN ]; // offset 0x05
} nrnid_struct;
```

A.3.10 Neuron ID Address Field Descriptions

```
addr_type  type;
```

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

```
unsigned   domain     : 1;
```

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

```
unsigned   subnet     : 8;
```

This field specifies the destination subnet number (1 – 255) within the domain. It is only used for routing of the message, and may be set to zero if the message should pass through all routers in the domain.

```
unsigned   nid[ NEURON_ID_LEN ];
```

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

A.3.11 Timer Field Descriptions

In the development environment, the user specifies these values in the network variable or message connection screens.

```
unsigned   rpt_timer  : 4;
```

This field specifies the time interval between repetitions of an outgoing message when unacknowledged-repeated service is used. The encoding of this field is specified in Table A–1.

```
unsigned   retry      : 4;
```

This field specifies the number of retries for acknowledged, request/response, or unacknowledged-repeated service (0 – 15). The maximum number of messages sent is one more than this number.

```
unsigned   rcv_timer  : 4;
```

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

```
unsigned   tx_timer   : 4;
```

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

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

Table A–1. Encoding of Timer Field Values (ms)

Value	rpt_timer	rcv_timer	tx_timer	non_group_timer
0	16	128	16	128
1	24	192	24	192
2	32	256	32	256
3	48	384	48	384
4	64	512	64	512
5	96	768	96	768
6	128	1,024	128	1,024
7	192	1.536	192	1,536
8	256	2,048	256	2,048
9	384	3,072	384	3,072
10	512	4,096	512	4,096
11	768	6,144	768	6,144
12	1,024	8,192	1,024	8,192
13	1,536	12,288	1,536	12,288
14	2,048	16,384	2,048	16,384
15	3,072	24,576	3,072	24,576

A.4 NETWORK VARIABLE AND ALIAS TABLES

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

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

Declarations from ACCESS.H

```
#define MAX_NVS 62
typedef struct {
     unsigned   nv_priority    : 1;   // offset 0x00
     unsigned   nv_direction   : 1;
     unsigned   nv_selector_hi : 6;
     unsigned   nv_selector_lo : 8;   // offset 0x01
     unsigned   nv_turnaround  : 1;   // offset 0x02
     unsigned   nv_service     : 2;
     unsigned   nv_auth        : 1;
     unsigned   nv_addr_index  : 4;
} nv_struct;
typedef struct {
     nv_struct  nv_cnfg;              // offset 0x00
     unsigned   primary;              // offset 0x03
     unsigned   long host_primary;    // offset 0x04
```

```
} alias_struct;

const nv_struct *access_nv( int index );
void update_nv( const nv_struct *nv_entry, int index );
const alias_struct *access_alias (int index);
void update_alias ( const alias_struct *alias_entry, int index );

typedef struct {
     unsigned   nv_sync       : 1;   // offset 0x00
     unsigned                 : 2;
     unsigned   nv_length     : 5;
     void       *nv_address;         // offset 0x01
} nv_fixed_struct;
```

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

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

For the network variable configuration and fixed tables on a Neuron Chip-hosted node, each of the tables consists of up to 62 entries, each three bytes in length. The number of entries is determined by the number of network variables declared in the application program — each element of a network variable array counts separately. For a node running the microprocessor interface program, the network variable tables are implemented on the host microprocessor and not in the Neuron Chip memory. In this case, the number of network variables is 4096. For a node running a regular application program, the index into either of these tables corresponding to a particular network variable is determined by the order of declaration of the network variables in the application program. Entries in the network variable configuration table may be read and written over the network with the *Query/Update Net Variable Configuration* network management messages. The network variable fixed table may not be written, except when downloading the application image. The application image may be in ROM for a MC143150.

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

A.4.1 Network Variable Configuration Table Field Descriptions

```
unsigned   nv_priority     : 1;
```

This bit is set to one if the network variable uses priority messaging. Specified by `bind_info(priority | nonpriority)` in the Neuron C declaration of the network variable. This may be overridden in the LonBuilder connection screen.

```
unsigned   nv_direction    : 1;
```

This bit is set to one if this is an output network variable, zero if an input. This bit must not be changed by the application program or an *Update Net Variable Configuration* network management message. This bit is technically not configuration data; it resides in this table for efficiency.

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

These two fields form a 14-bit network variable selector in the range 0 – 0x3FFF. Selector values 0x3000 – 0x3FFF are reserved for unbound network variables, with the selector value equal to 0x3FFF minus the network variable index. Selector values 0 – 0x2FFF are available for bound network variables. The input network variables on any one node must all have different selectors. The output network variables on any one node must all have different selectors. The LonBuilder, LonManager, and NSS binders ensure this by assigning the same network variable selector to the union of all the connection sets in which a network variable participates.

```
unsigned   nv_turnaround    : 1;
```

This bit is set to one if this is a turnaround network variable, that is, bound to another network variable on the same node.

```
unsigned   nv_service       : 2;
```

This field specifies the type of service used to deliver this network variable. Specified by `bind_info(ackd | unackd_rpt | unackd)` in the Neuron C declaration of the network variable. This may be overridden in the LonBuilder connection screen. The encoding is as follows:

ACKD	= 0	acknowledged
UNACKD_RPT	= 1	unacknowledged/repeated
UNACKD	= 2	unacknowledged

```
unsigned   nv_auth          : 1;
```

This bit is set to one if this network variable uses authenticated transactions. Specified by `bind_info(authenticated | nonauthenticated)` in the Neuron C declaration of the network variable. This may be overridden in the LonBuilder connection screen.

```
unsigned   nv_addr_index    : 4;
```

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

A.4.2 Network Variable Alias Table Field Descriptions

```
nv_struct nv_cnfg;
```

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

```
unsigned primary;
```

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

```
unsigned long host_primary;
```

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

A.4.3 Network Variable Fixed Table Field Descriptions

```
unsigned   nv_sync          : 1;
```

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

```
unsigned   nv_length        : 5;
```

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

```
void       *nv_address;
```

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

A.5 THE STANDARD NETWORK VARIABLE TYPE (SNVT) STRUCTURES

There are five structures associated with self-identification and self-documentation as follows:

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

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

Declarations from `ACCESS.H`

```
 typedef struct {
     unsigned long length;              // offset 0x00
     unsigned          num_netvars;     // offset 0x02
     unsigned          version;         // offset 0x03
union {
     struct {
         unsigned msb_num_netvars;      // offset 0x04
         unsigned mtag_count;           // offset 0x05
         } ver1;
     struct {
         unsigned mtag_count;           // offset 0x04
         } ver0;
} variable_part;
} snvt_struct;
 typedef struct {
     unsigned   ext_rec    : 1;         // offset 0x00
     unsigned   nv_sync    : 1;
     unsigned   nv_polled  : 1;
     unsigned   nv_offline : 1;
     unsigned   nv_service_type_config   : 1;
     unsigned   nv_priority_config       : 1;
     unsigned   nv_auth_config           : 1;
```

```
    unsigned   nv_config_class          : 1;
    unsigned   snvt_type_index;
} snvt_desc_struct;

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

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

Following the SNVT descriptor table is a null-terminated text string containing the self-documentation of the node. This string may be up to 1023 bytes in length, and it is the string specified in the Neuron C compiler directive `#pragma set_node_sd_string "sss"`. If there is no self-documentation string, the null termination byte is still present.

Following the self-documentation string for the node are extension records for those network variables that require them. For nodes with a Neuron Chip host, the self-identification and self-documentation information may be read over the network with the *Read Memory* network management command using `address_mode=0`.

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

A.5.1 SNVT Structure Field Descriptions

```
unsigned long length;
```

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

```
unsigned              num_net_vars;
```

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

```
unsigned              version;
```

This field contains the version number of the SNVT definition file used to compile the application program in this node. If the network management node's SNVT version is less than the SNVT version of the node being managed, then there may be some SNVTs in the node that are unknown to the network management node. Data types in the SNVT definition file are never deleted or modified, but new data types may be added in later versions of the file.

```
unsigned              msb_num_net_vars;
```

If the version number in the previous byte is zero, this byte is not present. If the version number is one, this byte is the most significant byte of the number of network variables declared in this node. If the firmware version number is 1, this byte is the most significant byte of the number of network variables declared in this node, where an NV array counts as only one item.

```
unsigned              mtag_count;
```

This field specifies the number of bindable message tags declared by the application program on this node. These message tags take the first entries in the address table (see Section A.3).

A.5.2 SNVT Descriptor Table Field Descriptions

```
unsigned   ext_rec                : 1;
```

This bit is set to one if this network variable has an extension record following the node's self-documentation string.

```
unsigned   nv_sync                : 1;
```

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

```
unsigned   nv_polled              : 1;
```

This bit is set to one if this network variable is polled. If it is an output network variable, this is specified with the modifier *polled* modifier in the Neuron C declaration. If this is an input network variable, this means that the network variable is mentioned as the argument of a qualified *poll( )* system call, or that there is an unqualified *poll( )* call in the application program.

```
unsigned   nv_offline             : 1;
```

This bit is set to one if the network management node should take the node off-line before this network variable is updated. Specified by `bind_info( offline )` in the Neuron C declaration of the network variable.

```
unsigned   nv_service_type_config  : 1;
```

This bit is set to one if the service type of this network variable `(ACKD, UNACKD, UNACKD_RPT)` may be modified by a network management message. Specified by `bind_info( service_type( config | nonconfig ) )` in the Neuron C declaration of the network variable.

```
unsigned   nv_priority_config     : 1;
```

This bit is set to one if the priority of this network variable may be modified by a network management message. Specified by `bind_info( priority | nonpriority ( config | nonconfig ) )` in the Neuron C declaration of the network variable.

```
unsigned   nv_auth_config         : 1;
```

This bit is set to one if the authentication of this network variable may be modified by a network management message. Specified by `bind_info( auth | nonauth ( config | nonconfig ) )` in the Neuron C declaration of the network variable.

```
unsigned   nv_config_class        : 1;
```

This bit is set to one if this is a configuration network variable whose value is stored in EEPROM. Specified with the config modifier in the Neuron C declaration of the network variable.

```
unsigned   snvt_type_index;
```

If this is a network variable of a standard type, this field specifies the type (1 – 250). For a listing of the Standard Network Variable Types, see *The SNVT Master List and Programmer's Guide*. If this field is zero, the network variable is of a non-standard type.

A.5.3 SNVT Table Extension Records

Extension records may be present for any of the network variables. If an extension record is present, the field `ext_rec` is set to one in the SNVT descriptor. The extension records appear in the order that the network variables were declared in the application program. Each extension record begins with a one-byte bit-mask defining which fields are to follow. The fields follow in the order of the bits in the bit mask defined here. These bits appear in the mask starting with the most significant bit.

```
unsigned   mre : 1;
```

If this bit is set to one, the extension record contains an estimate of the maximum rate at which this network variable is updated. The field is an unsigned int in the range 0 – 127 representing the rate in the range 0 – 1878.0 as specified by `bind_info( max_rate_est(` *nnn* `) )` in the Neuron C declaration of the network variable. The rate is given by the formula $2^{(n/8)-5}$ messages per second, rounded to the nearest tenth of a message per second.

```
unsigned   re : 1;
```

If this bit is set to one, the extension record contains an estimate of the average rate at which this network variable is updated. The field is an unsigned int in the range 0 – 127 representing the rate in the range 0 – 1878.0 as specified by `bind_info( rate_est(` *nnn* `) )` in the Neuron C declaration of the network variable. The rate is given by the formula $2^{(n/8)-5}$ messages/per second, rounded to the nearest tenth of a message per second.

```
unsigned   nm : 1;
```

If this bit is set to one, the extension record contains the name of the network variable as declared in the Neuron C program. This information is present if the compiler directive `#pragma enable_ sd_nv_names` is specified. The name is represented as a null terminated string of up to 17 bytes, or up to 22 bytes if it is a network variable array element. Each array element has its own extension record containing the name of the array, a left square bracket, one, two, or three decimal digits denoting the index of the element, and a right square bracket.

```
unsigned   sd : 1;
```

If this bit is set to one, the extension record contains the self-documentation string for the network variable. This is a null-terminated string of up to 1023 bytes, which may be specified by the modifier `sd_string ( "`*sss*`" )` in the Neuron C declaration of the network variable.

```
unsigned   nc : 1;
```

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

A.5.4 SNVT Alias Field Descriptions

```
unsigned   binding_II : 1;
```

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

does not initiate polls. Note that polling of such output network variables by a network management or monitor node would have to be done using *Fetch Network Variable* network management messages rather than network variable poll messages.

```
unsigned   query_stats : 1;
```

This bit is one if the query statistics addressing mode of the *Read Memory* network management command can be used to extract the extended statistics information.

```
unsigned   alias_count : 6;
```

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

```
unsigned long  host_alias;
```

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

A.6 THE CONFIGURATION STRUCTURE

This structure defines the hardware and transceiver properties of this node. It is located in EEPROM, and is part of both the application image written during node manufacture, and the network image written during node installation.

Declarations from `ACCESS.H`

```
#define LOCATION_LEN 6
#define NUM_COMM_PARAMS 7

typedef struct {
      unsigned long channel_id;                      // offset 0x00
      char              location[ LOCATION_LEN ];    // offset 0x02
      unsigned          comm_clock          : 5;     // offset 0x08
      unsigned          input_clock         : 3;
      unsigned          comm_type           : 3;     // offset 0x09
      unsigned          comm_pin_dir        : 5;
      unsigned          reserved[ 5 ];               // offset 0x0A
      unsigned          node_priority;               // offset 0x0F
      unsigned          channel_priorities;          // offset 0x10
      union {
        unsigned  xcvr_params[ NUM_COMM_PARAMS ];
        direct_param_struct   dir_params;            // offset 0x11
}
      unsigned          non_group_timer     : 4;     // offset 0x18
      unsigned          nm_auth             : 1;
      unsigned          preemption_timeout  : 3;
} config_data_struct;

typedef struct {
      unsigned          collision_detect    : 1;     // offset 0x11
      unsigned          bit_sync_threshold  : 2;
      unsigned          filter              : 2;
      unsigned          hysteresis          : 3;
      unsigned                              : 6;     // offset 0x12
      unsigned          cd_tail             : 1;
```

```
    unsigned        cd_preamble       : 1;
} direct_param_struct;

const config_data_struct config_data;

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

The application program may read this structure using the global declaration `config_data`, and may write it using the function `update_config_data()`. The structure is 25 bytes long, and it may be read and written over the network using the *Read Memory* and *Write Memory* network management messages with `address_mode=2`. The Media Access Control processor reads the channel parameters (offsets 0x08 through 0x17) from the configuration structure when the node is reset and initializes the transceiver. Therefore, after changing any of the channel parameters, the node should be reset for the changes to take effect.

A.6.1 Configuration Structure Field Descriptions

```
unsigned long channel_id;
```

This field specifies the ID of the channel that this node is assigned. In the LonBuilder environment, this is assigned automatically when the channel object is created. The NodeBuilder software uses a fixed channel ID. The Neuron Chip firmware does not reference this field, but it is available to assist in recovering the network topology by interrogating the nodes.

```
char            location[ LOCATION_LEN ];
```

This location field is used to pass a six-byte ASCII string describing the physical location of the node to the network management node. In the LonBuilder environment, it is defined in the node specification screen.

```
unsigned        comm_clock          : 5;
```

For direct mode transceivers, this field specifies the ratio between the Neuron Chip input clock oscillator frequency and the transceiver bit rate. For special-purpose mode transceivers, it specifies the rate of the bit clock between the Neuron Chip and the transceiver. In the development environment, the transceiver bit rate is specified in the channel screen. Table A–2 shows the transceiver bit rate as a function of the `input_clock` field and the `comm_clock` field.

Table A–2. Transceiver Bit Rate (kbps) as a Function of comm_clock and input_clock

		input_clock				
comm_clock	ratio	5 10 MHz	4 5 MHz	3 2.5 MHz	2 1.25 MHz	1 625 kHz
0	8:1	1,250	625	312.5	156.3	78.1
1	16:1	625	312.5	156.3	78.1	39.1
2	32:1	312.5	156.3	78.1	39.1	19.5
3	64:1	156.3	78.1	39.1	19.5	9.8
4	128:1	78.1	39.1	19.5	9.8	4.9
5	256:1	39.1	19.5	9.8	4.9	2.4
6	512:1	19.5	9.8	4.9	2.4	1.2
7	1,024:1	9.8	4.9	2.4	1.2	0.6

```
unsigned        input_clock          : 3;
```

This field specifies the Neuron Chip input clock (oscillator frequency). In the LonBuilder environment, the user specifies this value in the hardware properties screen for the node. The encoding is as follows:

Input Clock	Frequency
5	10.0 MHz
4	5.0 MHz
3	2.5 MHz
2	1.25 MHz
1	625 kHz
0	not used

```
unsigned        comm_type            : 3;
```

This field specifies the type of transceiver. In the LonBuilder environment, it is specified in the channel screen. The encoding is as follows:

0	Blank Neuron Chip
1	Single-ended
2	Special-purpose
5	Differential

```
unsigned        comm_pin_dir         : 5;
```

This field specifies the direction of the Neuron Chip's communications port pins. Zero indicates an input, one indicates an output with respect to the Neuron Chip. The least significant bit corresponds to pin CP0. Values used in this field include:

0x00	Blank Neuron Chip
0x0C	Direct mode — differential
0x0E	Direct mode — single-ended
0x17	Special-purpose — wake-up pin is an input
0x1E	Special-purpose — wake-up pin is an output

```
unsigned        reserved             [ 5 ];
```

The following five fields specify the raw transceiver parameters used by the Media Access Control processor. In the LonBuilder environment, these parameters are specified as the Raw Data in the channel screen. The values in these fields are the repetition counts for software delay loops in the firmware that control the timing of the media access algorithm. In the following descriptions, the timing formula are given in terms of "v," the value in the control field, which may be 0 .. 255 unless otherwise specified. A Neuron Chip processor cycle is 0.6 μs at 10 MHz input clock, 1.2 μs at 5 MHz and so on. See Figure 4–5 for a description of the timing of packet transmission. The fields are the following. These numbers are determined by the transceiver design and/or the interoperability guidelines.

`preamble_length`	This field determines the length of the preamble for direct mode. Time = [(209 .. 223) + (32) v] cycles (v = 1 .. 253). The minimum preamble time for a 10 MHz input clock is 600 ns x 209 = 125 μs. For special-purpose mode transceivers, this value should be zero.
`packet_cycle`	This field determines the packet cycle duration for counting down the backlog. Time = 1675 v cycles for direct mode, 1794 v cycles for special-purpose mode.
`beta2_control`	This field determines the *beta*2 slot width. Time = 40 + 20 v cycles.

`xmit_interpacket` This field determines the interpacket padding after transmitting. Time = 41 v cycles for v < 128, 145 (v – 128) for v ≥ 128.

`recv_interpacket` This field determines the interpacket padding after receiving. Time = 41 v cycles for v < 128, 145 (v – 128) for v ≥ 128.

```
unsigned        node_priority;
```

This field specifies the priority slot used by the node when sending priority messages on the channel (1 – 255). It should not be greater than the number of priority slots on the channel. If the node has no priority slot allocated, this is zero. In the LonBuilder environment, the node priority is specified in the hardware properties screen for the node.

```
unsigned        channel_priorities;
```

This field specifies the number of priority slots on the channel (0 – 255). The slots are numbered starting at one. In the LonBuilder environment, this is specified in the channel screen, with a maximum value of 127.

```
unsigned        xcvr_params[ NUM_COMM_PARAMS ];
```

This field forms an array of seven transceiver-specific parameters for special-purpose mode transceivers. All seven parameters are loaded into the transceiver when the node is initialized. In the LonBuilder environment, these are specified as General Purpose Data in the channel screen. The most significant bit of the first transceiver parameter is defined to be the alternate channel bit. The alternate channel is used for the last two transmission attempts when using acknowledged, request/response, or repeated service. In the case of the powerline transceiver, this bit determines whether the transceiver uses QPSK modulation at 9600 bps (bit is zero), or BPSK modulation at 4800 bps (bit is one). All other transceiver parameters are user-defined. For direct-mode transceivers, the first two bytes are overlaid by the direct mode parameter structure defined below.

```
unsigned        non_group_timer     : 4;
```

When the node receives a unicast (non-group) or broadcast message requiring a response or acknowledgment, the receive timer is set to the time interval specified by this field. If a message with the same transaction ID, priority, source, and destination address is received before the receive timer expires, it is considered to be a retry of the previous message. In the LonBuilder environment, this is implicitly set to the maximum of the non-group receive timers specified in the connection screens (with a minimum value of 768 ms). The encoding is of this field is shown in Table A–1. When network management messages using Neuron ID addressing are received, a receiver timer of 8.192 seconds is used, instead of the value of this field.

```
unsigned        nm_auth             : 1;
```

This field specifies that network management messages are to be authenticated. Setting this bit prevents the node from being configured by an unauthorized network management tool. However, network management messages received when the node is unconfigured can not be authenticated. Before setting a node's state to configured, a network management tool should ensure that the network management authentication bit is set to the desired state. In the LonBuilder environment, network management authentication is defined in the node specification screen.

```
unsigned        preemption_timeout  : 3;
```

This field specifies the maximum time the node will wait for a free buffer in preemption mode. A preemption mode timeout logs an error and resets the Neuron Chip. In the LonBuilder environment, this is defined in the node specification screen. The encoding is as follows:

0	forever
1	2 seconds
2	4 seconds
3	6 seconds
4	8 seconds
5	10 seconds
6	12 seconds
7	14 seconds

A.6.2 Direct-Mode Transceiver Parameters Field Descriptions

For direct (single-ended or differential) mode transceivers, these parameters are used to control the operation of the transceiver port. In the LonBuilder environment, these parameters are specified in the channel screen. In the NodeBuilder environment, these parameters are specified in the device template.

```
unsigned        collision_detect      : 1;
```

This field specifies that the Neuron Chip monitors pin CP4 for an indication of a collision on the network.

```
unsigned        bit_sync_threshold  : 2;
```

This field specifies the number of logic one bits received that are to be interpreted as the bit sync indicating the start of a packet. The encoding is as follows:

Threshold	Number of bits
0	4
1	5
2	6
3	7

```
unsigned        filter              : 2;
```

For differential mode transceivers, this field specifies the setting of the receive glitch filter (0 – 3). See the electrical specifications of the Neuron Chip (Table 4–4) for details of the available glitch filter settings.

```
unsigned     hysteresis          : 3;
```

For differential mode transceivers, this field specifies the setting of the receive hysteresis filter (0 – 7). See the electrical specifications of the Neuron Chip (Table 4–3) for details of the available hysteresis filter settings.

```
unsigned        cd_to_end_packet      : 6;
```

This field controls how close to the end of a packet the collision detect signal is checked in a transmitting Neuron Chip. It is set as a function of the bit rate and the input clock rate.

```
unsigned        cd_tail             : 1;
```

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

```
unsigned        cd_preamble         : 1;
```

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