Communication with Dallas Semiconductor MicroLAN devices in sensors on remote locations

Revision 1.00
September 8, 1998

By
David Smiczek
Dallas Semiconductor Inc.
and
Jan Kristoffersen
RAMTEX Engineering ApS for Brüel & Kjær A/S
and
Jørgen Bække
Brüel & Kjær Sound And Vibration Measurement A/S

Table of Contents

Background

Dallas Semiconductor MicroLAN devices are used in remote places where the distance between the device and the Host may exceed the MicroLAN specifications.

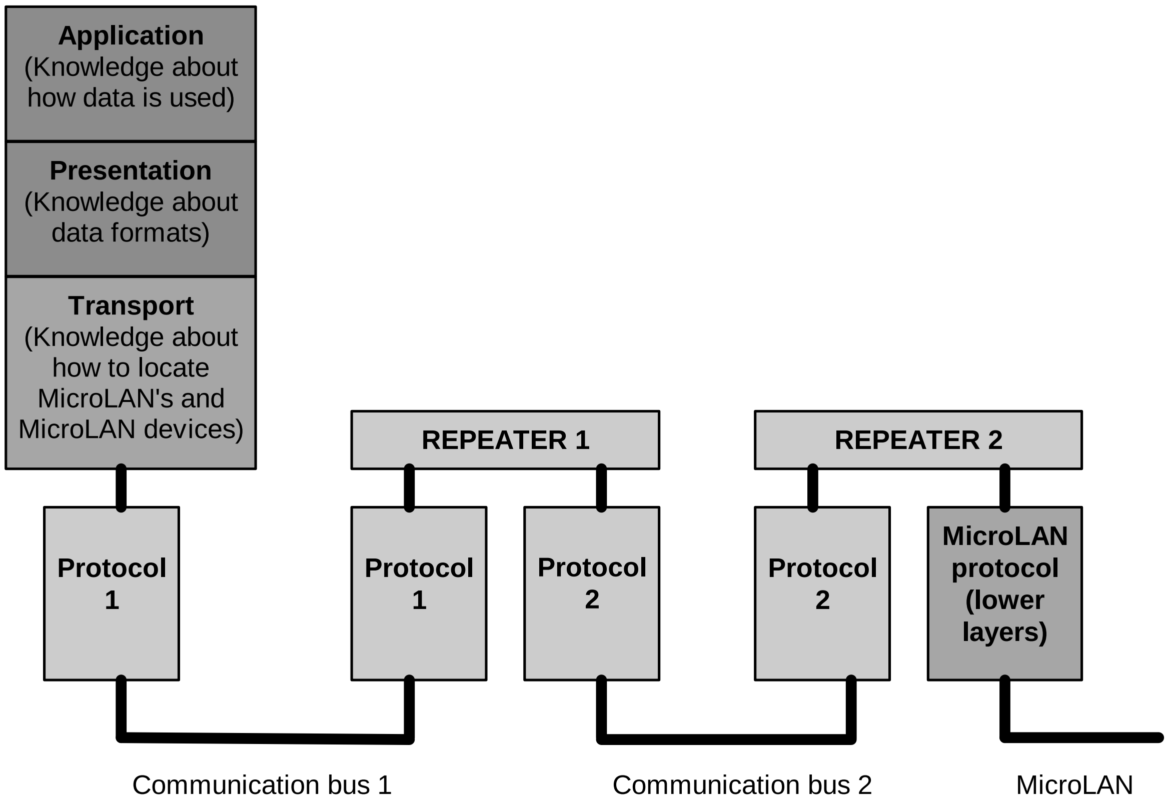
In such cases it is useful to be able to have some kind of communication equipment in between the sensor and the data processing computer which allow data from MicroLAN devices to be efficiently and transparently transferred, for instance over LAN or WAN networks, in a uniform and consistent way.

The goal with this document is to suggest how this problem can be easily solved in a relative simplified manner by inserting a transport layer in the driver software, which operates with a uniform frame buffer format. This extra layer allows different parts of the MicroLAN protocol software to be placed at different physical locations and in this way extend the versatility of the MicroLAN concept.

Scenario

Dallas Semiconductor devices built into remote sensors are, via a MicroLAN bus and a number of individual instruments, connected to a host application. The instruments are connected together via different communication lines and are using different communication protocols. The instruments will in this respect act as communication repeaters, which transfer data transparently between the Host application and devices on the MicroLAN bus.

Figure 1. Repeater in a multiple-protocol scenario


Application
(Knowledge about how data is used)
Presentation
(Knowledge about data formats)
Transport
(Knowledge about how to locate MicroLAN's and MicroLAN devices)
REPEATER 1
REPEATER 2
Protocol 1
Protocol 1
Protocol 2
Protocol 2
MicroLAN protocol (lower layers)
Communication bus 1
Communication bus 2
MicroLAN

Primary Goals

The software in the instruments (the repeaters) should be stable for the lifetime of the instruments (> 10 years) even if new (and yet unknown) Dallas Semiconductor devices are connected the MicroLAN bus.

The communication speed should be optimized. This implies that the number of communication transactions on the other communication busses (bus 1 and bus 2 in the example) should be minimized, as these busses may have a much lower bandwidth than the MicroLAN bus itself, and/or may also be used for other communication tasks not related to the MicroLAN communication.

Derived Goals

All knowledge about specific MicroLAN device types should be isolated to the Host program. The repeaters should not contain any device specific knowledge.

Communication sessions should be based on whole buffers (instead of individual bytes and bits) in order to minimize the communication overhead on the intervening busses (bus 1 and 2 in the example).

A few basic and device transparent MicroLAN transactions for the repeaters should be defined, together with the corresponding buffer formats. These few device transparent transactions should be sufficient for communication with all types of MicroLAN devices.

The minimum buffer size, which a repeater must be able to handle, shall be well defined. (It is assumed that the repeaters may have a very limited buffering capability).

If communication with a Dallas Semiconductor device requires a larger buffer it should be possible to split a MicroLAN transaction over several intervening buffers transferred between the Host and the repeater which have the MicroLAN connection.

The intervening communication protocols will typically pack the MicroLAN buffer frame in “envelopes” using their own format (for instance add some header and tail bytes). This is transparent to the MicroLAN communication and is not a part of this specification.

Accepted Limitations

It is not required that the MicroLAN interface in the repeater handle EPROM programming voltages, higher speed 'overdrive' communication, or strong pull-up power delivery but its use is to be defined by this specification.

It is assumed that all communication initiatives through the repeaters are initiated from the host application.

Transparent MicroLAN Buffer Transactions

The transparent buffer transaction on the MicroLAN bus takes advantage of the fact that transmit and receive can be done at the same time on a bit to bit basis. (A one (1) must be transmitted by the repeater when receiving bit frames from a MicroLAN device).

After a transaction the buffer in the repeater will contain any information read from the MicroLAN (device). The buffer in the repeater with the resulting MicroLAN transaction can then be transmitted back to the host, if needed. All buffer communication initiatives is taken by the host.

Transparent MicroLAN Buffer Protocol Specification

Buffer Formats

The transparent buffer transaction protocol has two communication buffers defined in the repeater. One *inbound* buffer that receives a frame from the host computer and one *outbound* buffer where the return frame is constructed.

General inbound Format

Length byte	Array of Single and Multiple byte Commands

General outbound Format

Length byte	Array of Single and Multiple byte Command results

The first byte in both the *inbound* and *outbound* frames is a length byte representing the number of bytes in the frame not including the length byte.

The *inbound* frame may contain a series of MicroLAN commands. The commands in *inbound* buffer are parsed. If the parsing produces an result, the command and result are put in the *outbound* buffer.

If the length is 0 in an inbound buffer the buffer is ignored and no processing takes place.
The minimum size of the inbound and outbound buffers a repeater must be able to handle is 49 bytes including the Length byte.

General Command Formats

There are two types of MicroLAN commands. Single byte MicroLAN commands and multi-byte MicroLAN commands. The MSB bit of the first byte in the header identifies if it is a single byte or a multi-byte command. If it is a multi-byte command the header consist of two bytes, the command byte and a byte defining the length of the attached block of data bytes.

Inbound Command Format

Multiple byte command

Command code, 1 byte (0xxx xxxx)	data_length 1 byte	data_bytes

Single byte command

Command code, 1 byte (1xxx xxxx)

Outbound Response Format

Multiple byte command response (specified commands)

Command code, 1 byte (0xxx xxxx)	data_length 1 byte	data_bytes

Single byte command response (all commands)

Command code, 1 byte (1xxx xxxx)	return_code 1 byte

A command is always a single byte value. The command is always copied from the *inbound* buffer to the *outbound* buffer if the MicroLAN operation produces a result.

data_length used with multi-byte commands is always a single byte with the value as the number of bytes following the **data_length** byte in the buffer. With DATA_xxx commands the **data_length** value is also used to differentiate between read and write operations on the internal protocol registers. For a register write, **data_length** is different from zero. Data is copied from the *inbound* buffer to the data register identified by the command. No command or data is copied to the *outbound* buffer. For a register read, **data_length** is equal to zero. The command is copied to the *outbound* buffer. The **data_length** for the register identified by the command is copied to the *outbound* followed by the data from the register.

return_code is always a single byte value following a single byte command in the *outbound* buffer.

Command Overview

Table 1a. Single byte commands

Command Name	Description	Code
CMD_ML_RESET	Reset all devices on MicroLAN and report if any devices are responding	80 (hex)
CMD_ML_SEARCH	Perform MicroLAN search using the current search state as specified in the DATA_ID and DATA_SEARCH_STATE registers.	81
CMD_ML_ACCESS	Select the current device as specified in the DATA_ID register using the MicroLAN MATCH_ROM command 55 hex.	82
CMD_ML_OVER-DRIVE_ACCESS	Select the current device as specified in the DATA_ID register using the MicroLAN MATCH_ROM command 69 hex which at the same time sets the device in overdrive mode. If overdrive mode is not supported by the repeater end this command will return RET_CMD_UNKNOWN	83
CMD_RESET	Reset repeater end to default state. Previous processed data in the outbound buffer remains unchanged.	84
CMD_GETBUF	Return the outbound buffer as it is. If this command can be processed normally then the command byte is not copied to the outbound buffer and the length of the outbound buffer remains unchanged. If this command can not be processed the CMD_GETBUF command is returned immediately, typically with the RET_BUSY return code. This is the only command which causes the outbound buffer to be returned. When the CMD_GETBUF command is present in an inbound buffer it must always be the last command in the inbound buffer. When a command (in the next inbound buffer) following CMD_GETBUF is not a CMD_GETBUF then the outbound buffer is cleared before this command is processed. This allow the host to request retransmission of the outbound buffer multiple times.	85
CMD_ERROR	Error command. Is only used in the outbound buffer of the repeater end to signal errors to the host. It can typically be errors resulting from processing of multi-byte commands or any internal errors in the repeater end. If it occurs in an inbound buffer the return status should be RET_CMD_UNKNOWN.	86
(Reserved)	Single byte commands reserved for further extension of this protocol. Should return with the return code RET_CMD_UNKNOWN	87-CF
(Vendor specific)	Single byte commands reserved to be defined by the repeater vendor. If not used, these commands should return with the return code RET_CMD_UNKNOWN	D0-FF

Table 1b. Multi-byte commands, with required repeater data registers.

Command Name	Description	Command	Register size
DATA_ID	Write or read the 64 bit MicroLAN ID number register. If the data length of a write command is less than 8 and more than 0 then the remaining register bytes are cleared.	00 (hex)	8 (bytes)
DATA_SEARCH_STATE	Write or read the 2 byte MicroLAN search state register. During register write the internal search algorithm state is cleared. Write to the first register presets LastDiscrepancy, (the DATA_ID bit index for search start). The second register, LastFamilyDiscrepancy is always cleared by write.	01	2
DATA_SEARCH_CMD	Write or read MicroLAN search command register. This is the MicroLAN command used during the CMD_ML_SEARCH command.	02	1
DATA_MODE	Write or read register which define the options, speed and level of the MicroLAN bus	03	1
DATA_CAPABILITY	Read MicroLAN capabilities of repeater (Operation assumes an inbound data_length value of 0)	04	(1) Constant value
DATA_OUTBOUND_MAX	Read max length of outbound buffer in bytes. (Operation assumes an inboumd data_length value of 0)	05	(1) Constant value
DATA_INBOUND_MAX	Read max length of inbound buffer in bytes. (Operation assumes an inbound data_length value of 0)	06	(1) Constant value
DATA_PROTOCOL	Read protocol version identification as a NUL (/0) terminated C string. The current version 1.00 protocol is “ML100”. (Operation assumes an inbound data_length value of 0)	07	(Max 20 incl. \0) Constant value
DATA_VENDOR	Read repeater vendor identification data as a NUL (/0) terminated C string. (Operation assumes an inbound data_length value of 0)	08	(Max 20 incl. \0) Constant value
CMD_ML_BIT	Initiates write_read MicroLAN communication bit using the LS bit of the each data byte provided.	09	(na)
CMD_ML_DATA	Initiates a MicroLAN communication block. The first byte in data defines the total number of MicroLAN data bytes processed on the MicroLAN bus called block_length. MicroLAN processing starts with the data byte following	0A	(na)

	this byte. If the number of data_bytes to process is larger than the header block_length-1 then the remaining bytes are processed equal to an inbound data value of FF hex. The result of the MicroLAN processing is placed in the outbound register.		
CMD_DELAY	Perform a delay which length is defined by the attached data byte. data_length must be 1.	0B	(na)
(Reserved)	Multi-byte commands reserved for further extension of this protocol specification. If the command is unknown to the repeater end, then the command CMD_ERROR with return code RET_CMD_UNKNOWN is placed into the outbound buffer.	0C-4F	
(Vendor specific)	Multi-byte commands reserved for further vendor specific purposes. If the command is unknown to the repeater end, then the command CMD_ERROR with return code RET_CMD_UNKNOWN is placed into the outbound buffer.	50-7F	

Total: 12 RAM register bytes

Table 2. Return codes (always follow single byte commands in outbound buffer).

Return Code Name	**Description**	**Return Code**
RET_SUCCESS	Command operation successful	00 (hex)
RET_END_SEARCH	End of device search, the last device in ID search was the previous device found and the search state will now be reset.	01
RET_BUSY	Previous buffer has not been processed yet.	02
RET_ERROR	Unspecified error (stops inbound buffer processing)	03
RET_NO_DEVICE	No devices present on the MicroLAN (stops inbound buffer processing)	04
RET_ML_SHORTED	MicroLAN appears to be shorted (stops inbound buffer processing)	05
RET_OUTBOUND_OVERRUN	Outbound buffer overrun error (stops inbound buffer processing)	06
RET_INBOUND_OVERRUN	Inbound buffer overrun error (stops inbound buffer processing)	07
RET_REG_OVERRUN	Data register overrun error (stops inbound buffer processing)	08
RET_END_OF_INBOUND	Unexpected end of inbound buffer (stops inbound buffer processing)	09
RET_READ_ONLY	Attempt to write a read-only data register (data_length not 0, stops inbound buffer processing)	0A
RET_WRITE_ONLY	Attempt to read a write-only data register (data_length is 0, stops inbound buffer processing)	0B
RET_CMD_UNKNOWN	Command unknown (stops inbound buffer processing)	0C
(Reserved)	Reserved for future expansion of this protocol specification	0D to 7F
(Vendor specific)	Vendor specific return codes	80 to FF

Before any vendor specific commands are used by the host the DATA_VENDOR command should be used to identify that the expected repeater type is present. This precaution will prevent command contention between different vendors.

Command Processing Description

Command Processing Sequence

The *inbound* and *outbound* buffers may contain multiple commands in a sequence.

The *inbound* buffer is parsed and processes sequentially. Most of the commands being processes will append results to the *outbound* buffer. The commands sequence in the *outbound* buffer will thus match the command sequence order in *inbound* buffer. The only exception to this is when CMD_ERROR is inserted in the *outbound* buffer, and when a busy state RET_BUSY is returned immediately as result of a CMD_GETBUF command.

The outbound buffer is cleared when an inbound buffer is received, except if the first command in the inbound buffer is a CMD_GETBUF, which instead causes the outbound buffer to be (re-)transmitted.

If the reception of an *inbound* buffer results in *inbound* buffer overflow, the CMD_ERROR command is inserted in the *outbound* buffer with a RET_INBOUND_OVERRUN status. The remaining contents of the *inbound* buffer is ignored.

If the processing of an *inbound* buffer results in *outbound* buffer overflow, either the current command, if it is a single byte command, or the CMD_ERROR command is inserted in the *outbound* buffer with the RET_ OUTBOUND_OVERRUN status. The processing of current command is stopped, and any further command processing of the inbound buffer stops.

Inbound may also be halted due to MicroLAN conditions of no device present RET_NO_DEVICE or a shorting of the MicroLAN bus RET_ML_SHORTED. Unknown or improper commands will return codes (RET_RET_OVERRUN, RET_END_OF_INBOUND, RET_READ_ONLY, RET_WRITE_ONLY, RET_CMD_UNKNOWN, RET_OUTBOUND_OVERRUN) and stop *inbound* command processing. Any error result that halts *inbound* command processing will be considered the final error message.

A repeater implementation must assure that there always is place in the *outbound* buffer for one final error message (CMD_ERROR + error status). After the final error message has been put in the *outbound* buffer all processing in the repeater is allowed to stop, as described above, until the *outbound* buffer has been transmitted or reset.

If successive error events are detected by the repeater end, after the final error message has been placed in the *outbound* buffer, then any following error events should be ignored. This state presets until the outbound buffer has been reset after transmission or by the CMD_ML_RESET command. This assures that error information is given to the host in the same sequence as they occur in the slave and that no previous information in the buffer is lost or overwritten.

When processing of an *inbound* buffer is halted due to an error condition, the *inbound* buffer is scanned for a CMD_GETBUF command. If found, the present content of *outbound* buffer is send back to the host. If a CMD_GETBUF is not found, no further command processing takes place until another *inbound* buffer is received.

Buffer Frame Synchronization

The outbound buffer is transmitted by the repeater end when a CMD_GETBUF command is received.

The CMD_GETBUF can be looked upon as a “token”. When the repeater end is given the CMD_GETBUF “token” from the host it is allowed to transmit the *outbound* buffer once. The *outbound* buffer must only be transmitted once for each CMD_GETBUF “token”, and any transmission must not start before a CMD_GETBUF “token” is received (and processed).

If the repeater end is busy the CMD_GETBUF “token” is returned back to the host immediately. The host end is then allowed to try to send the token back again (single bus polling) or to give it to some other low-level MicroLAN protocol in operation elsewhere (multibus polling).

CMD_GETBUF must always be the last command (if not the only command) in an *inbound* buffer as any further command parsing of the *inbound* buffer is stopped.

Figure 2a. Receiving Inbound buffer

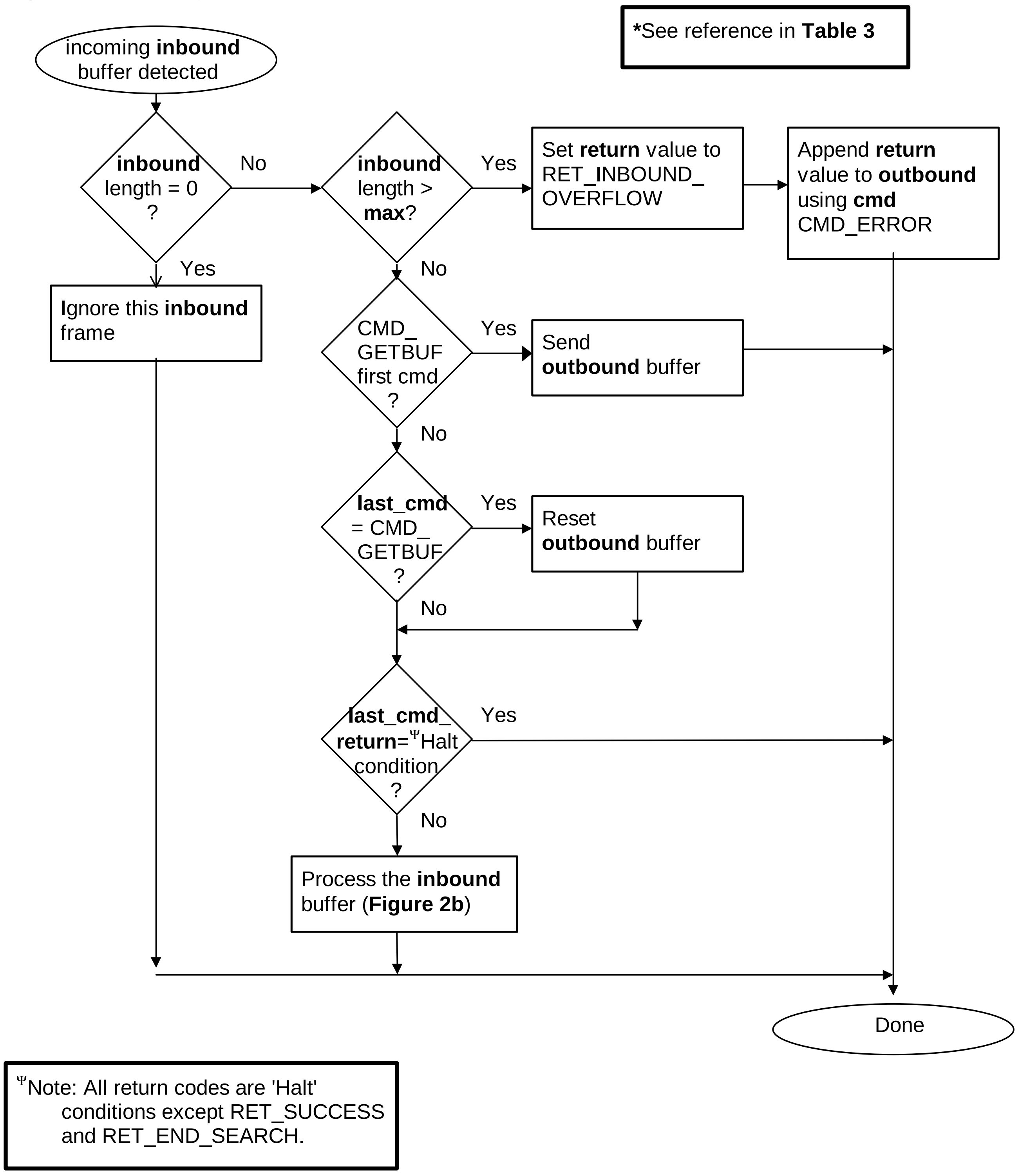

*****See reference in **Table 3**
incoming **inbound** buffer detected
inbound length = 0 ?
No
Yes
Ignore this **inbound** frame
inbound length > **max**?
Yes
Set **return** value to RET_INBOUND_ OVERFLOW
Append **return** value to **outbound** using **cmd** CMD_ERROR
No
CMD_ GETBUF first cmd ?
Yes
Send **outbound** buffer
No
last_cmd = CMD_ GETBUF ?
Yes
Reset **outbound** buffer
No
last_cmd_ return=ΨHalt condition ?
Yes
No
Process the **inbound** buffer (**Figure 2b**)
Done
ΨNote: All return codes are 'Halt' conditions except RET_SUCCESS and RET_END_SEARCH.

Figure 2b. Inbound command processing

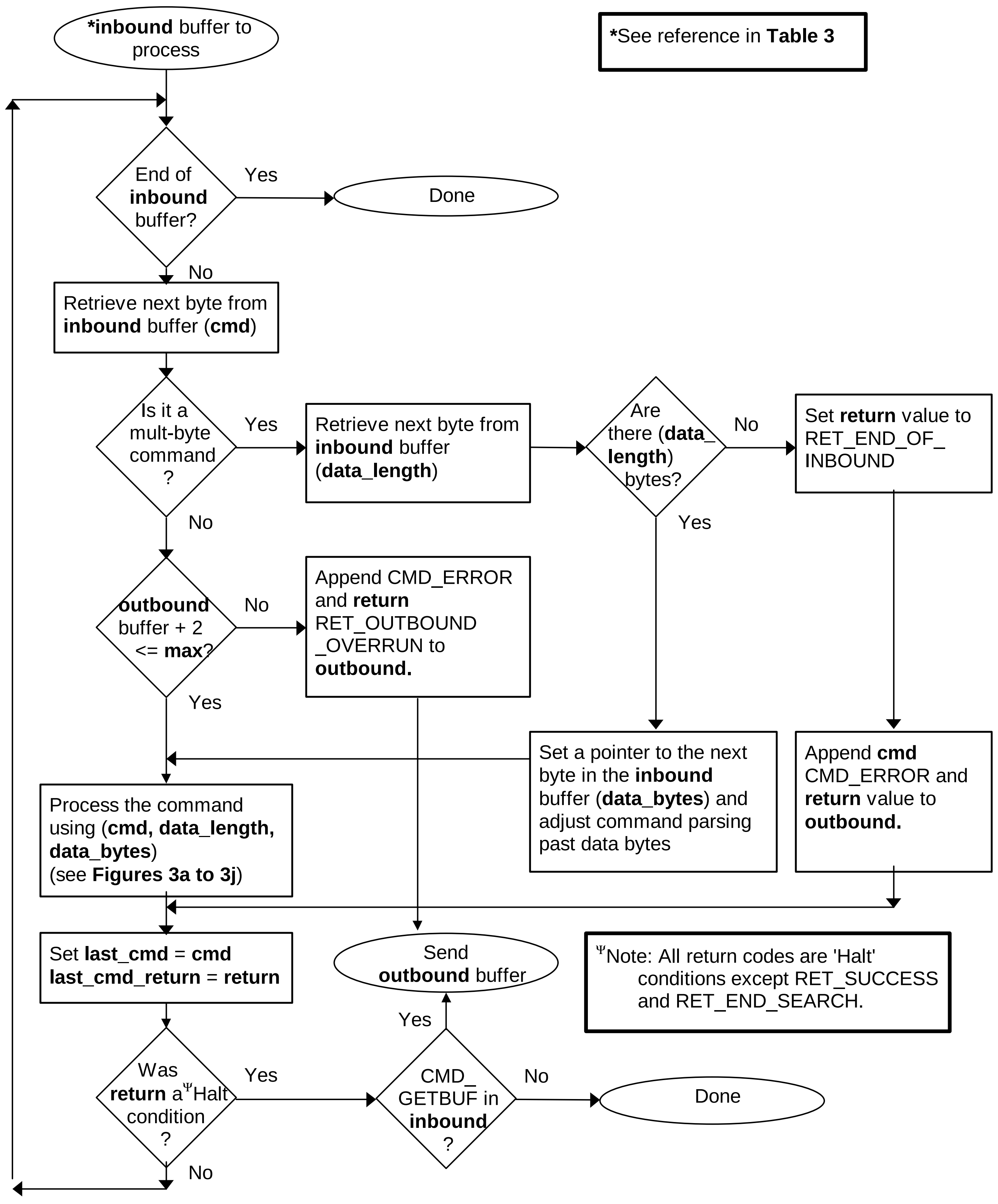
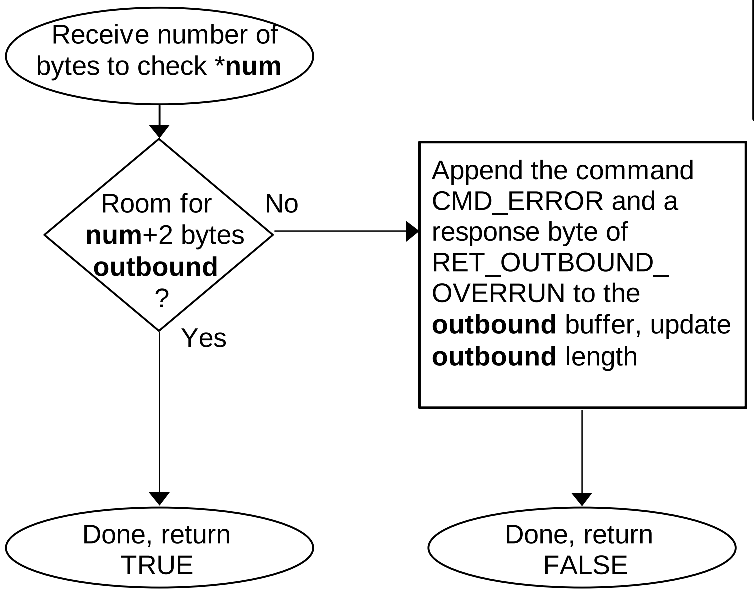

*inbound buffer to process
*See reference in Table 3
End of inbound buffer?
Yes
Done
No
Retrieve next byte from inbound buffer (cmd)
Is it a mult-byte command ?
Yes
Retrieve next byte from inbound buffer (data_length)
Are there (data_ length) bytes?
No
Set return value to RET_END_OF_ INBOUND
No
Yes
outbound buffer + 2 <= max?
No
Append CMD_ERROR and return RET_OUTBOUND _OVERRUN to outbound.
Yes
Set a pointer to the next byte in the inbound buffer (data_bytes) and adjust command parsing past data bytes
Append cmd CMD_ERROR and return value to outbound.
Process the command using (cmd, data_length, data_bytes) (see Figures 3a to 3j)
Set last_cmd = cmd last_cmd_return = return
Send outbound buffer
ΨNote: All return codes are 'Halt' conditions except RET_SUCCESS and RET_END_SEARCH.
Yes
Was return a ΨHalt condition ?
Yes
CMD_ GETBUF in inbound ?
No
Done
No

Table 3. Flow-chart variable descriptions.

Flow Variable	Description
cmd	the current command code being processed
data_length	number of data bytes if current command is a mult-byte command
data_bytes	pointer to the start of the data bytes in a mult-byte command
return	current result byte of the command being processed
inbound	buffer containing the incoming list of commands from host
outbound	buffer containing the outgoing response back to the host resulting from commands in inbound
max	the maximum size of the inbound buffer, same as DATA_INBOUND_MAX
last_cmd	last command that was evaluated
last_cmd_return	result of last command that was evaluated

Figure 2c. Outbound space verification


Receive number of bytes to check *num
Room for num+2 bytes outbound ?
No
Yes
Append the command CMD_ERROR and a response byte of RET_OUTBOUND_ OVERRUN to the outbound buffer, update outbound length
Done, return TRUE
Done, return FALSE

Reference
num - the number of bytes needed in outbound buffer
***See reference in Table 3**

Detailed Command Description

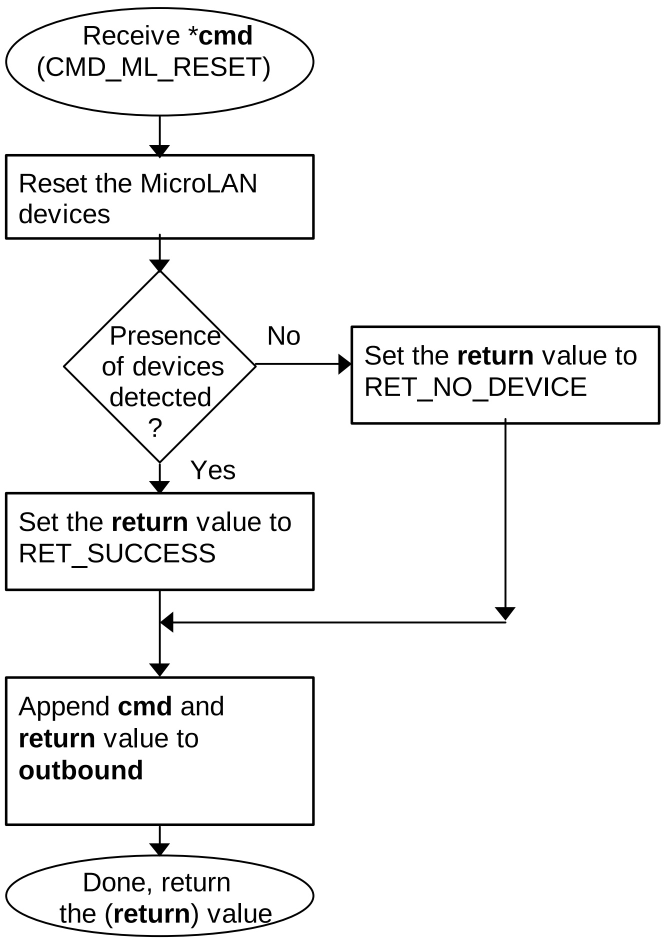
CMD_ML_RESET

The CMD_ML_RESET command resets all MicroLAN devices and detects whether at least one device is present. If a device is not present then the return code RET_NO_DEVICE is placed in the *outbound* buffer and *inbound* buffer processing stops. This command uses the DATA_MODE data register for the communication speed at which the reset signal is sent to the MicroLAN.

Example: reset devices on MicroLAN

inbound	CMD_ML_RESET
outbound	CMD_ML_RESET <return byte>

Figure 3a. Processing command CMD_ML_RESET



Receive *cmd (CMD_ML_RESET)
Reset the MicroLAN devices
Presence of devices detected ?
No
Yes
Set the return value to RET_NO_DEVICE
Set the return value to RET_SUCCESS
Append cmd and return value to outbound
Done, return the (return) value

***See reference in Table 3**

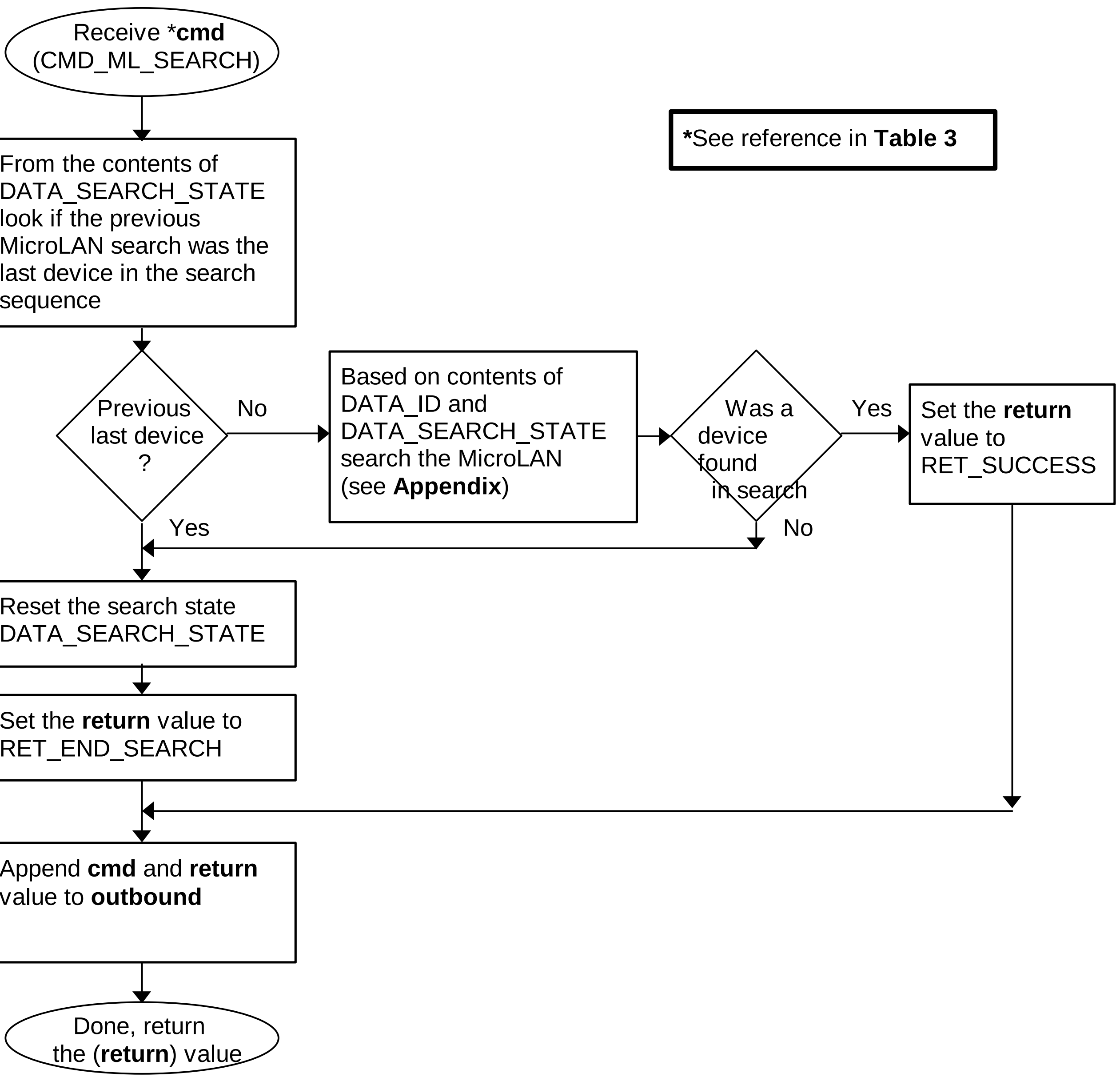
CMD_ ML_SEARCH

The CMD_ML_SEARCH command performs a search using the current search state in the repeater to find the 'next' device on the MicroLAN. The command does NOT do a MicroLAN reset before the search. A CMD_ML_RESET command must be used before CMD_ML_SEARCH in most cases. This command uses the search state information in the repeater data register DATA_SEARCH_STATE and DATA_ID. To reset the search to find the 'first' device on the MicroLAN, set the two bytes in the DATA_SEARCH_STATE data register to 0. See the DATA_SEARCH_STATE command description for more details on its use. This command uses the DATA_MODE data register for the communication speed at which the search is performed on the MicroLAN. See **Appendix** for a detailed description of the MicroLAN search algorithm.

```
Example: search for the first device on the MicroLAN.  Reset the search state
         and then do a search.  Read the ID of the discovered device.

         inbound       DATA_SEARCH_STATE <2><0,0>
                       CMD_ML_RESET
                       CMD_ML_SEARCH
                       DATA_ID <0>

         outbound      CMD_ML_RESET <return byte>
                       CMD_ML_SEARCH <return byte>
                       DATA_ID <8><8 bytes of ROM>
```

```
Example: search for the next two devices on the MicroLAN and return the ID's of
         these devices.

         inbound       CMD_ML_RESET
                       CMD_ML_SEARCH
                       DATA_ID <0>
                       CMD_ML_RESET
                       CMD_ML_SEARCH
                       DATA_ID <0>

         outbound      CMD_ML_RESET <return byte>
                       CMD_ML_SEARCH <return byte>
                       DATA_ID <8><8 bytes of ROM>
                       CMD_ML_RESET <return byte>
                       CMD_ML_SEARCH <return byte>
                       DATA_ID <8><8 bytes of ROM>
```

Figure 3b. Processing command CMD_ML_SEARCH


Receive ***cmd** (CMD_ML_SEARCH)
From the contents of DATA_SEARCH_STATE look if the previous MicroLAN search was the last device in the search sequence
Previous last device ?
No
Yes
Based on contents of DATA_ID and DATA_SEARCH_STATE search the MicroLAN (see **Appendix**)
Was a device found in search
Yes
No
Set the **return** value to RET_SUCCESS
Reset the search state DATA_SEARCH_STATE
Set the **return** value to RET_END_SEARCH
Append **cmd** and **return** value to **outbound**
Done, return the (**return**) value
***See reference in Table 3**

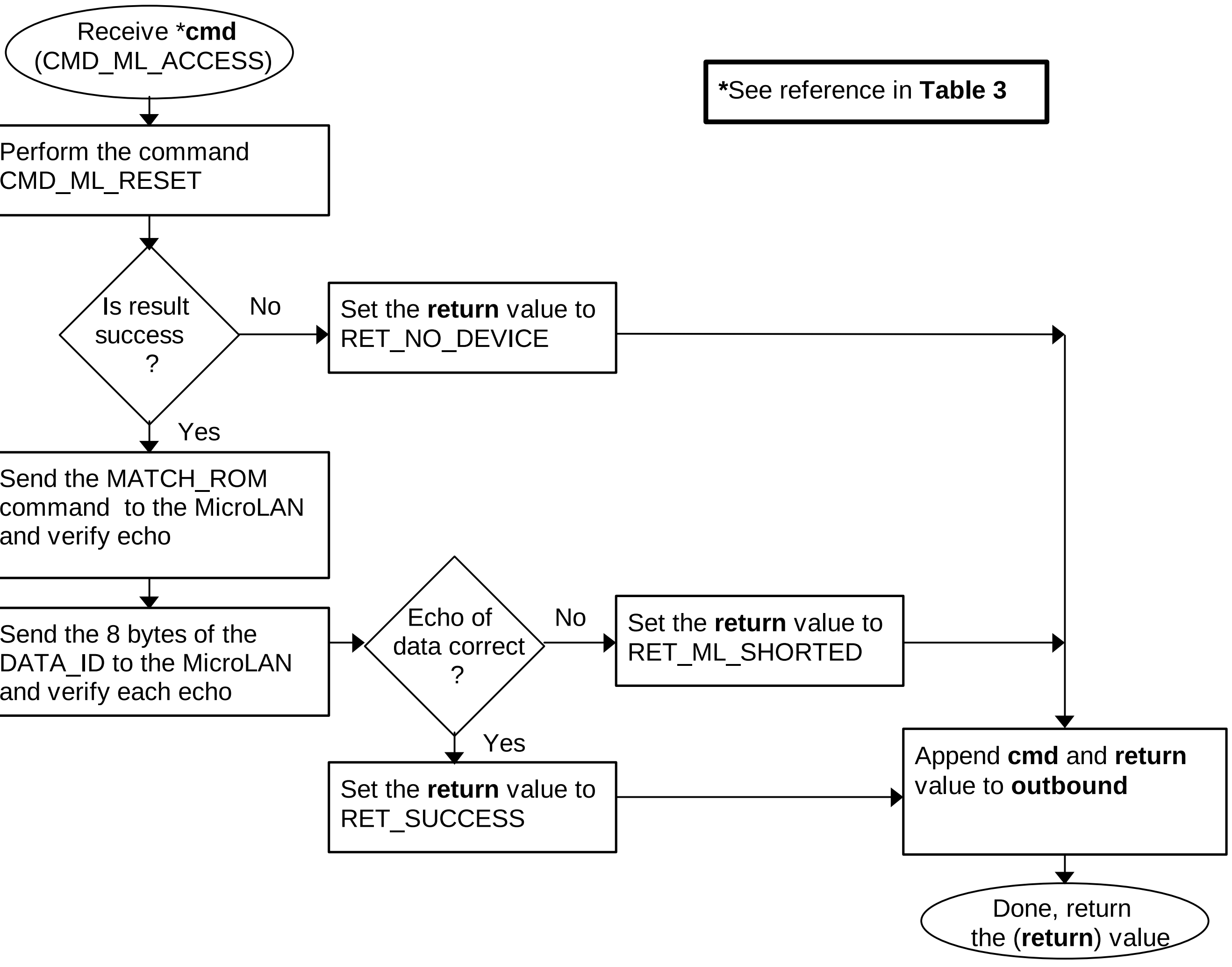
CMD_ ML_ACCESS

The CMD_ML_ACCESS command selects the device whose ID number is in the data register DATA_ID. The MicroLAN device is selected by using the 'Match ROM' command. This command is used by first resetting the line with the CMD_ML_RESET command, sending the 'Match ROM' command of 55 hex and then sending the 8 byte ID from DATA_ID.
At this point the MicroLAN device will be 'accessed'. It is then ready for device specific commands.
This command returns the return code *RET_NO_DEVICE* if CMD_ML_RESET fails and RET_ML_SHORTED if any other problem is detected. On success the return code is RET_SUCCESS.
This command uses Speed bit in the DATA_MODE data register to select the communication speed at which the access is performed on the MicroLAN.

Example: set the current device ID and then select that device.

```
        inbound          DATA_ID <8><8 bytes of ID>
                         CMD_ML_ACCESS

        outbound         CMD_ML_ACCESS <return byte>
```

Figure 3c. Processing command CMD_ML_ACCESS


Receive *cmd (CMD_ML_ACCESS)
*See reference in Table 3
Perform the command CMD_ML_RESET
Is result success ?
No
Set the return value to RET_NO_DEVICE
Yes
Send the MATCH_ROM command to the MicroLAN and verify echo
Send the 8 bytes of the DATA_ID to the MicroLAN and verify each echo
Echo of data correct ?
No
Set the return value to RET_ML_SHORTED
Yes
Set the return value to RET_SUCCESS
Append cmd and return value to outbound
Done, return the (return) value

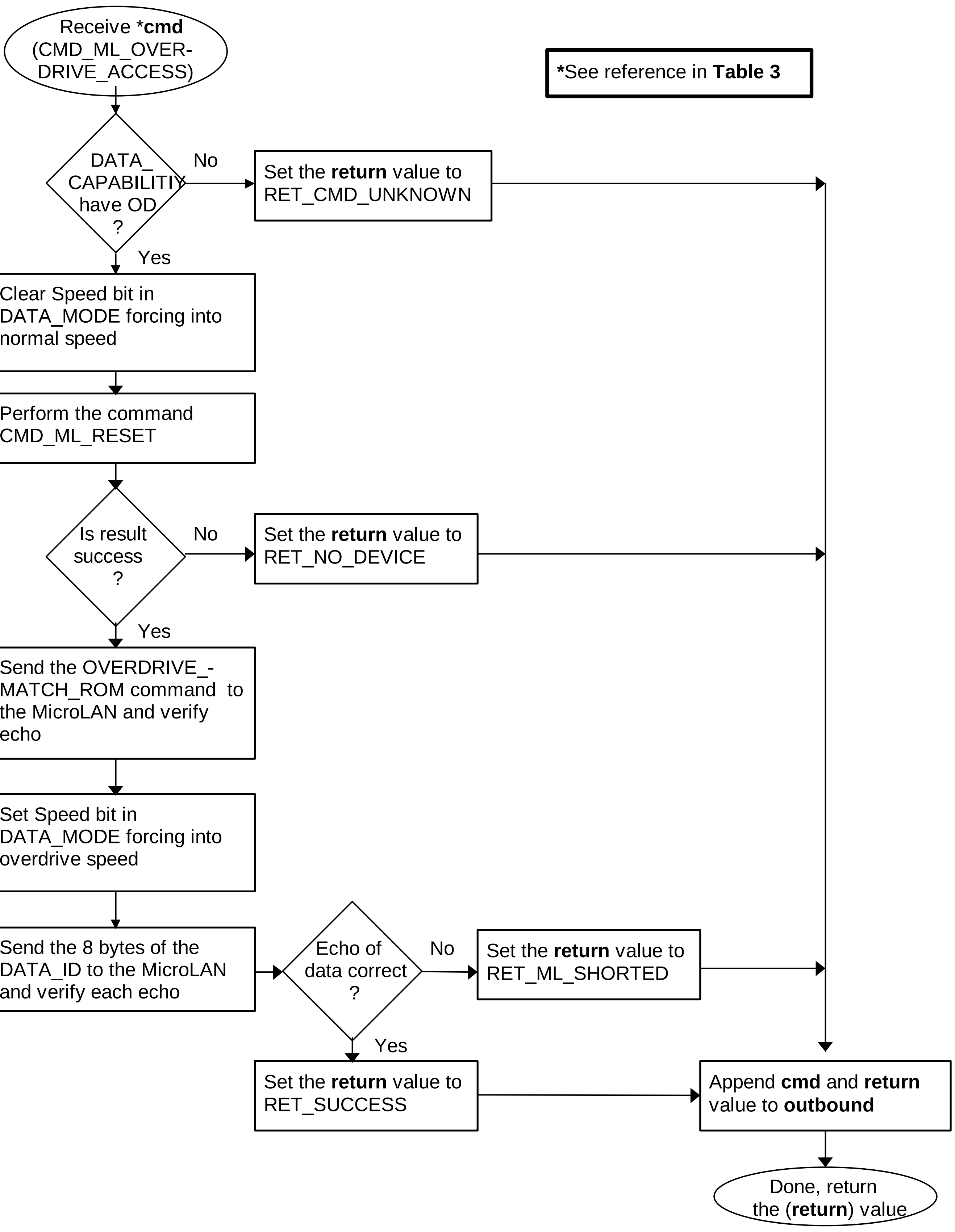
CMD_ ML_OVERDRIVE_ACCESS

The CMD_ML_OVERDRIVE_ACCESS command selects the device whose ID number is in the data register DATA_ID and at the same time places the device and repeater into Overdrive communication speed. This is done by first forcing the repeater into normal speed by clearing the *Speed* bit in the DATA_MODE register. The MicroLAN is then reset at normal speed with the CMD_ML_RESET command.
If CMD_ML_RESET detects a device presence then the 'Overdrive Match ROM' command (69 hex) is sent also at normal speed. At this point the *Speed* bit in the DATA_MODE register is set forcing the repeater into Overdrive communication speed. The 8 byte ID in DATA_ID is then transmitted at Overdrive speed. The *Speed* bit remains set in Overdrive after this command is completed. This command returns the return code *RET_NO_DEVICE* if CMD_ML_RESET fails and RET_ML_SHORTED if any other problem is detected. On success the return code is RET_SUCCESS.

Note that for this command to operate the repeater must be capable of Overdrive speed (see DATA_CAPABILITY command) and the current device whose ID is in DATA_ID must be an Overdrive capable device. If overdrive mode is not supported by the repeater then use of this command will result in RET_CMD_UNKNOWN.

```
Example: set the current device ID and then select that device and place it and
         the repeater into Overdrive..

         inbound        DATA_ID <8><8 bytes of ID>
                        CMD_ML_OVERDRIVE_ACCESS
                        DATA_MODE <00>

         outbound       CMD_ML_OVERDRIVE_ACCESS <return byte>
                        DATA_MODE <01><01 (Overdrive)>
```

Figure 3d. Processing command CMD_ML_OVERDRIVE_ACCESS


Receive ***cmd** (CMD_ML_OVER-DRIVE_ACCESS)
*See reference in **Table 3**
DATA_ CAPABILITIY have OD ?
No
Set the **return** value to RET_CMD_UNKNOWN
Yes
Clear Speed bit in DATA_MODE forcing into normal speed
Perform the command CMD_ML_RESET
Is result success ?
No
Set the **return** value to RET_NO_DEVICE
Yes
Send the OVERDRIVE_-MATCH_ROM command to the MicroLAN and verify echo
Set Speed bit in DATA_MODE forcing into overdrive speed
Send the 8 bytes of the DATA_ID to the MicroLAN and verify each echo
Echo of data correct ?
No
Set the **return** value to RET_ML_SHORTED
Yes
Set the **return** value to RET_SUCCESS
Append **cmd** and **return** value to **outbound**
Done, return the (**return**) value

CMD_RESET

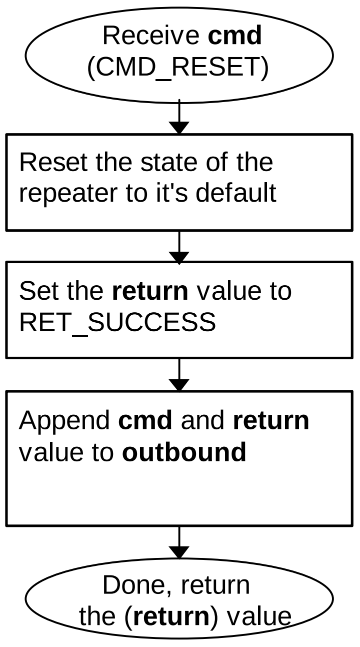
Repeater Reset resets the repeater and brings it up in the default state. Any data content in the outbound buffer not already read by the host will be lost after CMD_RESET. See **Table 4** for the default values that are set by CMD_RESET.

```
Example: reset the state of the repeater to its default

inbound        CMD_RESET

outbound       CMD_RESET <return byte>
```

Table 4. Default values

Repeater State	**Default Value**
DATA_ID	0,0,0,0,0,0,0,0
DATA_SEARCH_STATE	0,0
DATA_SEARCH_CMD	F0 hex
DATA_MODE	0 (Normal speed)
DATA_CAPABILITY	repeater specific
DATA_OUTBOUND_MAX	repeater specific, 49 bytes minimum
DATA_INBOUND_MAX	repeater specific, 49 bytes minimum
DATA_PROTOCOL	"ML100" for this specification
DATA_VENDOR	repeater specific
outbound length	0
last_cmd	CMD_ML_RESET (80 hex)
last_cmd_return	RET_SUCCESS (00 hex)
LastDeviceFlag	0

Figure 3e. Processing command CMD_RESET


Receive **cmd** (CMD_RESET)
Reset the state of the repeater to it's default
Set the **return** value to RET_SUCCESS
Append **cmd** and **return** value to **outbound**
Done, return the (**return**) value

*See reference in **Table 3**

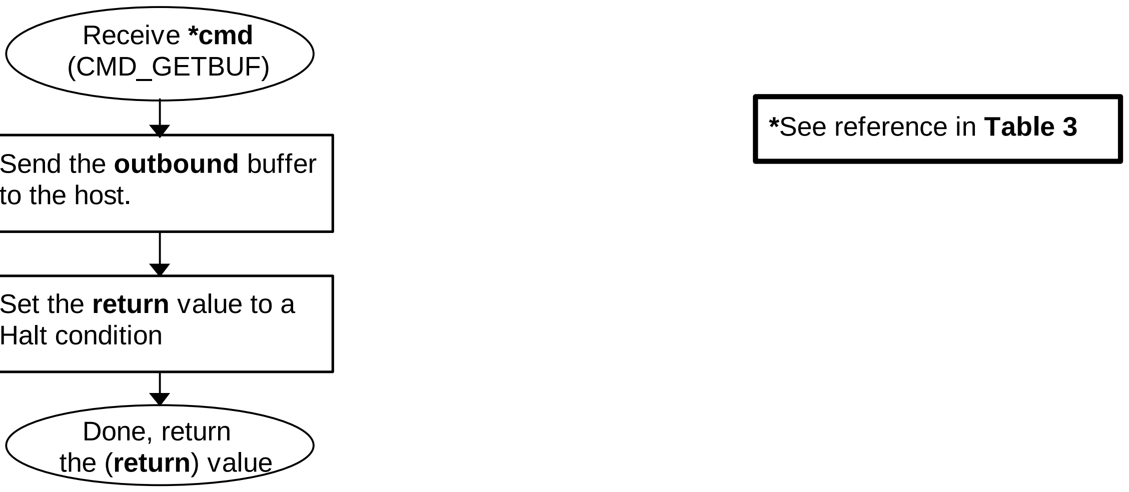
CMD_GETBUF

The CMD_GETBUF command sends the current contents of the *outbound* buffer back to the host. Any further commands in the *inbound* buffer is ignored. The CMD_GETBUF command should therefore always be the last command in the *inbound* buffer.

The *outbound* buffer remain unchanged after processing of CMD_GETBUF. The host can therefore always request retransmission of the *outbound* buffer by sending a new CMD_GETBUF command (should something have gone wrong during the previous transmission).

A command in the *inbound* buffer following processing of a CMD_GETBUF command will reset the outbound buffer before the new command is processed. See the Command Processing Description for details on CMD_GETBUF.

Figure 3f. Processing command CMD_GETBUF


Receive *cmd (CMD_GETBUF)
Send the outbound buffer to the host.
Set the return value to a Halt condition
Done, return the (return) value
*See reference in Table 3

CMD_ERROR

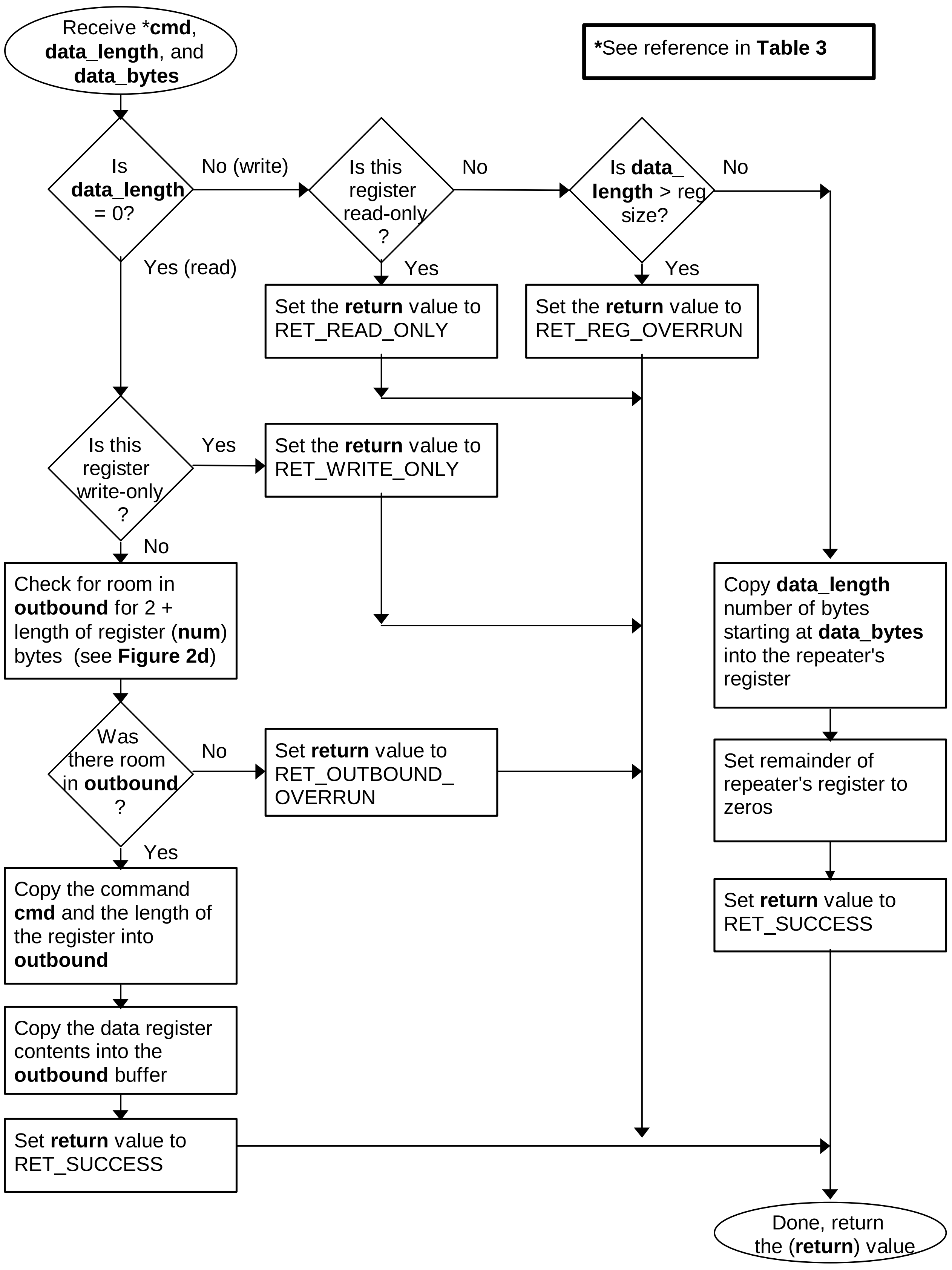
The error command is only used in the *outbound* buffer as a way to convey errors back to the host. It can typically be errors resulting from processing of multi-byte commands. If this command occurs in the *inbound* buffer it is copied to the *outbound* buffer with the return status RET_CMD_UNKNOWN.

DATA_ ID

The DATA_ID command allows reading and writing of the 8 byte device ID register in the repeater. This register contains the ID of the last device found on the MicroLAN. This register is both used in the current search to find the 'next' device on the MicroLAN and is also the location for the result of that search. The length is 8 bytes with a default value of all 0's.

The **Figure 3g** flow diagram displays the general flow for commands that read or write repeater registers. Note that some repeater registers can only be read (read-only, length byte zero) and some can only be written (write-only, length byte non-zero).

Figure 3g. Processing data register commands


Receive *cmd, data_length, and data_bytes
*See reference in Table 3
Is data_length = 0?
No (write)
Is this register read-only ?
No
Is data_ length > reg size?
No
Yes (read)
Yes
Set the return value to RET_READ_ONLY
Yes
Set the return value to RET_REG_OVERRUN
Is this register write-only ?
Yes
Set the return value to RET_WRITE_ONLY
No
Check for room in outbound for 2 + length of register (num) bytes (see Figure 2d)
Copy data_length number of bytes starting at data_bytes into the repeater's register
Was there room in outbound ?
No
Set return value to RET_OUTBOUND_ OVERRUN
Set remainder of repeater's register to zeros
Yes
Copy the command cmd and the length of the register into outbound
Set return value to RET_SUCCESS
Copy the data register contents into the outbound buffer
Set return value to RET_SUCCESS
Done, return the (return) value

DATA_ SEARCH_STATE

The DATA_SEARCH_STATE command enables reading and writing to the two byte register that keeps that count of the last search and is used to find the 'next' device in the current search. These two bytes can be set in combination with DATA_ID to achieve targeted searches of a particular family code. The default value is all 0's. The first byte in this search state is the *LastDiscrepancy* number. This indicates the search path that was taken on the last search. This number is needed to continue a search where the previous search left off. The second byte is the *LastFamilyDiscrepancy* which in indicates that last search direction that was taken within the key family code byte of the DATA_ID. A third byte in the search state is a flag *LastDeviceFlag* that indicates the last search was the final device on this search of the MicroLAN. The *LastDeviceFlag* is internal to the repeater and is automatically cleared when writing to DATA_SEARCH_STATE. The **Figure 3g** flow diagram displays the general flow for commands that read or write repeater registers. See **Appendix** for a detailed description of the MicroLAN search algorithm.

Table 5. MicroLAN search state description

Byte variable Name	**Description**	**Byte Number**
LastDiscrepancy	Bit index to the DATA_ID register. Identifies from which bit the (next) search discrepancy check should start. For example will a value of 9 cause the next search discrepancy to start from the 9th bit in the DATA_ID register. The search is therefore limited to devices identified by the first 8 bits in DATA_ID (the device family code). The default value is 0 (search for all devices).	0
LastFamilyDiscrepancy	Bit index to the DATA_ID register. It is updated during search to identify the first bit in DATA_ID where a selection between two MicroLAN devices was made. It is only updated within the first 8 bits of DATA_ID (the device family bits). If the next search starts from this bit index the search will be for devices in the next device family. See **Appendix** for a description of how this value is updated by the search algorithm.	1

There are five types of operations that can be performed by using the CMD_ML_SEARCH command and manipulating the DATA_SEARCH_STATE and DATA_ID register values. These operations concern discovery and verification of the ID's of MicroLAN devices.

FIRST

The 'FIRST' operation is to search on the MicroLAN for the first device. This is performed by setting all three bytes of DATA_SEARCH_STATE to zero and calling CMD_ML_SEARCH. The resulting ID number can then be read from the DATA_ID register. If no devices are present on the MicroLAN the CMD_ML_RESET will return RET_NO_DEVICE. If an error occurred during the search itself then CMD_ML_SEARCH will return RET_END_SEARCH.

Example: Find the first device on the MicroLAN and read the ID.

```
inbound     DATA_SEARCH_STATE <2><0,0>
            CMD_ML_RESET
            CMD_ML_SEARCH
            DATA_ID <0>

outbound    CMD_ML_RESET <return byte>
            CMD_ML_SEARCH <return byte>
            DATA_ID <8><8 bytes of ID>
```

NEXT

The 'NEXT' operation is to search on the MicroLAN for the next device. This search is usually performed after a 'FIRST' operation or another 'NEXT' operation. This is performed by leaving the two bytes of DATA_SEARCH_STATE unchanged from the previous search and calling CMD_ML_SEARCH. The resulting ID number can then be read from the DATA_ID register. If the last search was the last device on the MicroLAN or an error occurred during the search itself then CMD_ML_SEARCH command will return RET_END_SEARCH.

Example: Find the next device on the MicroLAN and read the ID.

```
inbound     CMD_ML_RESET
            CMD_ML_SEARCH
            DATA_ID <0>

outbound    CMD_ML_RESET <return byte>
            CMD_ML_SEARCH <return byte>
            DATA_ID <8><8 bytes of ROM>
```

TARGET

The 'TARGET' operation is a way to pre-set the search state to first find a particular family type. Each MicroLAN device has a one byte 'family code' embedded within the ID number. This 'family code' allows the MicroLAN master to know what operations this device is capable of. If there are multiple devices on the MicroLAN it is common practice to target a search to only the family of devices that are of interest. To target a family set the DATA_SEARCH_STATE to 09, 00 (hex). This sets the *LastDiscrepancy* to beyond the family code. Then set the desired family code byte into the first byte of the DATA_ID register.
Now call the CMD_ML_SEARCH function and then read the resulting ID in the DATA_ID register. Note that if no device of the desired family are currently on the MicroLAN another type will be found so the family code in the DATA_ID must be checked.

```
Example: Target a family type and find the first device of that type on the
         MicroLAN and read it's ID.

         inbound      DATA_SEARCH_STATE <2><09,00>
                      DATA_ID <1><family code>
                      CMD_ML_RESET
                      CMD_ML_SEARCH
                      DATA_ID <0>
                      DATA_SEARCH_STATE <0>

         outbound     CMD_ML_RESET <return byte>
                      CMD_ML_SEARCH <return byte>
                      DATA_ID <8><8 bytes of ROM>
                      DATA_SEARCH_STATE <2><2 bytes of search state>
```

SKIP

The 'SKIP' operation is to skip all of the devices that have the family type that were found in the previous search on the MicroLAN. This operation can only be performed after a search. It is accomplished by copying the *LastFamilyDiscrepancy* (byte 1) into the *LastDiscrepancy* (byte 0) of the DATA_SEARCH_STATE and then performing another search with CMD_ML_SEARCH. The following example assumes that we have already performed a search and know the contents of DATA_SEARCH_STATE.

```
Example: Skip all MicroLAN devices with the family type found on last search
         and find the next device of a different type and read it's ID.

         inbound      DATA_ SEARCH_STATE <2><LastFamilyDescrepancy,
                           00>
                      CMD_ML_RESET
                      CMD_ML_SEARCH
                      DATA_ID <0>

         outbound     CMD_ML_RESET <return byte>
                      CMD_ML_SEARCH <return byte>
                      DATA_ID <8><8 bytes of ROM>
```

VERIFY

The 'VERIFY' operation verifies if a device with a know ID is currently connected to the MicroLAN. It is accomplished by supplying the ID and doing a targeted search on that ID to verify it is present. First, set the DATA_ID register to the known ID. Then set the *LastDiscrepancy* (byte 0) in the DATA_SEARCH_STATE to 64 (40 hex). Perform the search operation with CMD_ML_SEARCH and then read the DATA_ID result. If the search was successful and the DATA_ID remains the ID that was being searched for then the device is currently on the MicroLAN.

```
Example: Set the ID and verify that this MicroLAN device is currently connected.

        inbound       DATA_ SEARCH_STATE <2><40, 00>
                      DATA_ID <8><ID of device to verify>
                      CMD_ML_RESET
                      CMD_ML_SEARCH
                      DATA_ID <0>

        outbound      CMD_ML_RESET <return byte>
                      CMD_ML_SEARCH <return byte>
                      DATA_ID <8><8 bytes of ROM>
```

DATA_ SEARCH_CMD

The DATA_SEARCH_CMD command enables reading and writing to the one byte register that contains the command used during a search operation. Currently the two valid commands are F0 (hex) for a normal search and EC (hex) to find only alarming devices. The length is 1 byte with a default value of F0 (hex). The **Figure 3g** flow diagram displays the general flow for commands that read or write repeater registers.

DATA_MODE

The DATA_MODE command enables reading and writing to the one byte register that contains the current speed and level modes of the MicroLAN on the repeater. **Table 6** describes the predefined mode bit flags. Writing to this register will result in an immediate change in the state of MicroLAN so that the mode can be manipulated in the middle of a command block. If the repeater does not have the capability to do the operation specified in the bit flags then there will be no effect. Consult the DATA_CAPABILIY data register. The **Figure 3g** flow diagram displays the general flow for commands that read or write repeater registers.

Table 6. Bit description of MicroLAN mode flags in the DATA_MODE register.

Mode Bit Name	Description	Bit Number
Speed	Normal speed if 0 and overdrive if 1	0
PowerDelivery	Normal 5 volt pull-up if 0 and strong pull-up if 1	1
ProgramVoltage	12 volt programming voltage disabled if 0 and enabled if 1 (PowerDelivery and PowerDown must be disabled)	2
PowerDown	low impedance zero voltage used to power down the MicroLAN bus (PowerDelivery and ProgramVoltage must be disabled)	3
(Reserved)	Reserved for future expansion of this protocol specification. Use 0,0 as default.	4,5
(Vendor specific)	Vendor specific mode flags. Before setting any of these bits the host should use the DATA_VENDOR command to identify that the expected repeater type is present. This precaution will prevent functionality contention between different repeater vendors. Use 0,0 as default.	6,7

DATA_CAPABILITY

The DATA_CAPABILITY command enables reading the one byte register that contains the capabilities of repeater for MicroLAN communication power delivery and speed. **Table 7** describes the predefined feature bit flags. The **Figure 3g** flow diagram displays the general flow for commands that read or write repeater registers. Note that the DATA_CAPABILITY register is read-only.

Table 7. Bit description of MicroLAN capability flags in the DATA_CAPABILITY register.

Capability Bit Name	Description	Bit Number
Overdrive_C	Overdrive speeds available if 1, only normal speed is available if 0	0
PowerDelivery_C	Strong 5-volt pull-up power delivery available if 1, only normal communication pull-up available if 0	1
ProgramVoltage_C	12 volt programming voltage available if 1, not available if 0	2
PowerDown_C	low impedance zero voltage available if 1, not available if 0	3
(Reserved)	Reserved for future expansion of this protocol specification	4,5
(Vendor specific)	Vendor specific mode flags	6,7

DATA_OUTBOUND_MAX

The DATA_OUTBOUND_MAX command enables reading the one byte register that contains the predefined maximum data length in bytes of the *outbound* buffer. The minimum size of the *outbound* buffer is 48 bytes not including the length byte. The **Figure 3g** flow diagram displays the general flow for commands that read or write repeater registers. Note that the DATA_OUTBOUND_MAX register is read-only.

Note that because there should always be room for a final error message (two bytes) in the outbound buffer, the effective size which can be depended on during MicroLAN communication is two bytes less than DATA_OUTBOUND_MAX.

DATA_INBOUND_MAX

The DATA_INBOUND_MAX command enables reading the one byte register that contains the predefined maximum data length in bytes of the *inbound* buffer. The minimum size of the *inbound* buffer is 48 bytes not including the length byte. The **Figure 3g** flow diagram displays the general flow for commands that read or write repeater registers. Note that the DATA_INBOUND_MAX register is read-only.

DATA_PROTOCOL

The DATA_PROTOCOL command enables reading the zero terminated string that represents the protocol name and version. This specification describes version 1.00, represented by the DATA_PROTOCOL string "ML100". The **Figure 3g** flow diagram displays the general flow for commands that read or write repeater registers. Note that the DATA_PROTOCOL register is read-only. The maximum length of this C-string is 20 bytes including the 0 termination.

DATA_VENDOR

The DATA_VENDOR command enables reading the zero terminated string that represents the vendor name. This is used to identify vendor-specific commands and modes. The **Figure 3g** flow diagram displays the general flow for commands that read or write repeater registers. Note that the DATA_VENDOR register is read-only. The maximum length of this C-string is 20 bytes including the 0 termination.

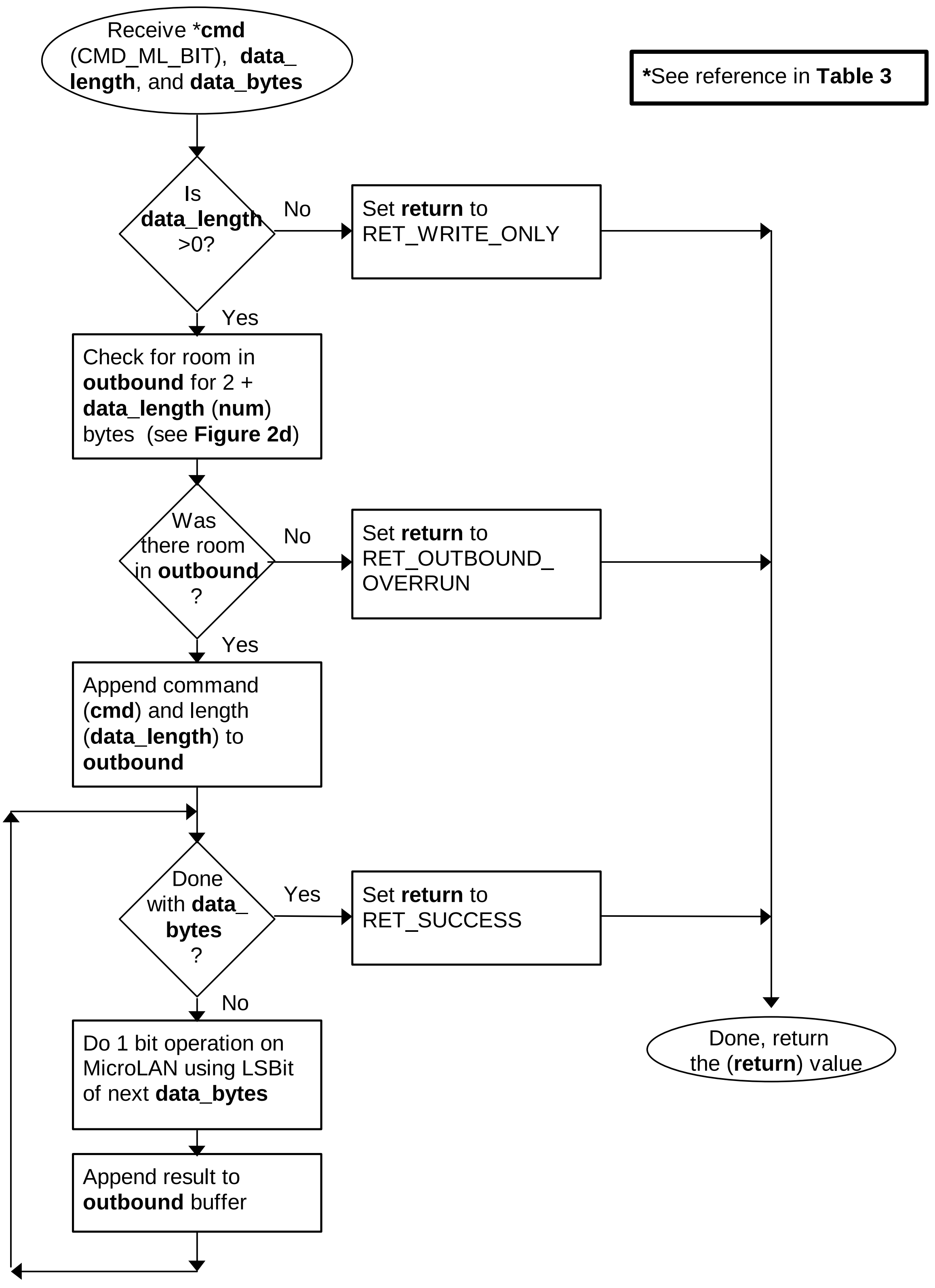
CMD_ ML_BIT

The CMD_ML_BIT gives bit level communication with the MicroLAN. The CMD_ML_BIT is a multi-byte command so it provides a length byte that must be greater then 0 and one or more data bytes. Each data byte provided represents one bit of communication. The least significant bit of each data byte is sent to the MicroLAN and the result of that bit communication is placed into a byte in the *outbound* buffer in a multi-byte read format. This command uses the DATA_MODE data register for the communication speed at which the bit operation is performed on the MicroLAN.

Example: Do the first two bits of the search algorithm manually

```
        inbound         CMD_ ML_RESET
                        CMD_ ML_DATA <2><length=1><0F>
                        CMD_ ML_BIT <2><01,01>

        outbound        CMD_ML_RESET <return byte>
                        CMD_ ML_DATA <1><0F>
                        CMD_ML_BIT <2><result1,result2>
```

Figure 3h. Processing command CMD_ML_BIT

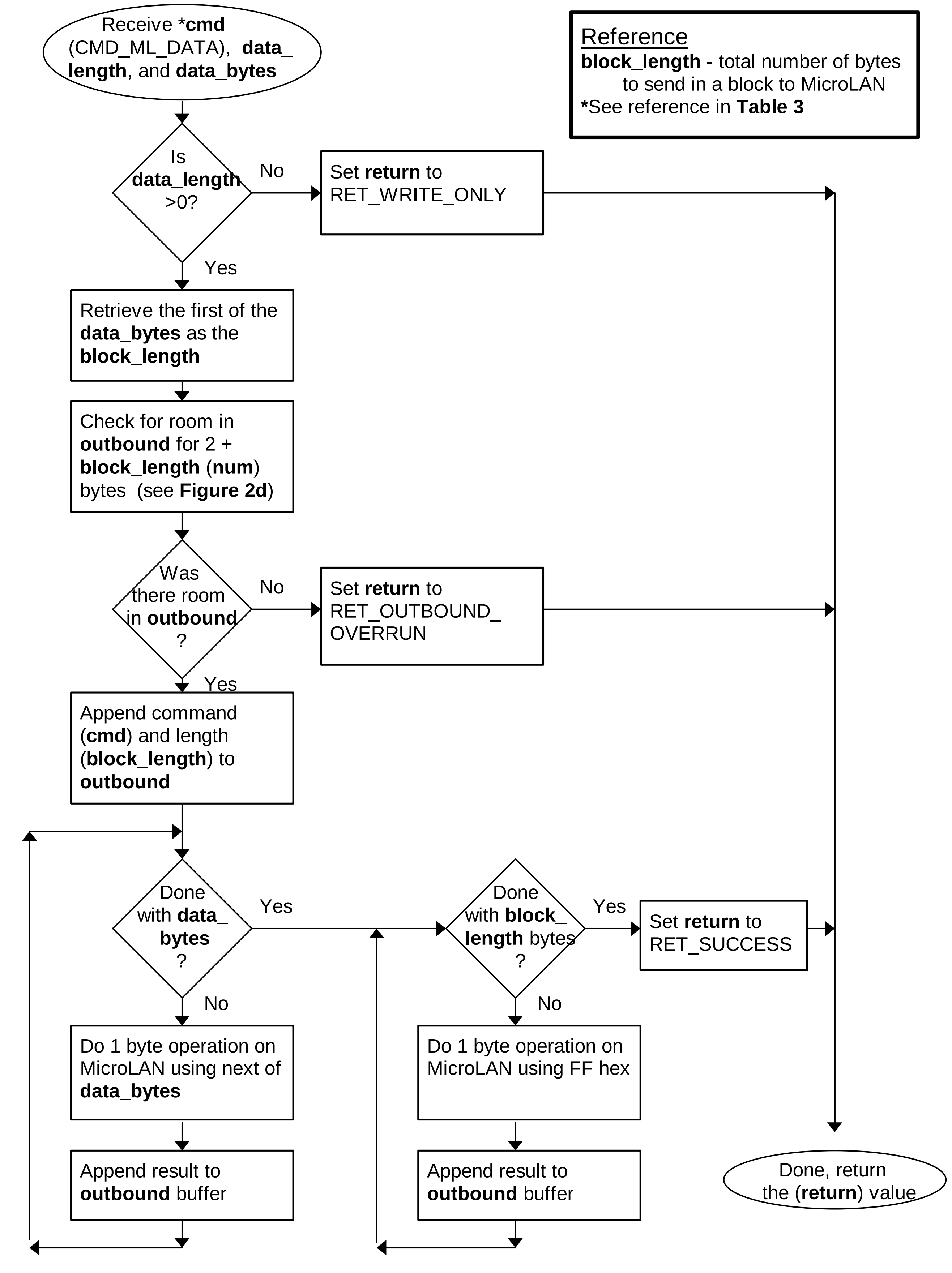

Receive ***cmd** (CMD_ML_BIT), **data_ length**, and **data_bytes**
***See reference in Table 3**
Is **data_length** >0?
No
Set **return** to RET_WRITE_ONLY
Yes
Check for room in **outbound** for 2 + **data_length** (**num**) bytes (see **Figure 2d**)
Was there room in **outbound** ?
No
Set **return** to RET_OUTBOUND_ OVERRUN
Yes
Append command (**cmd**) and length (**data_length**) to **outbound**
Done with **data_ bytes** ?
Yes
Set **return** to RET_SUCCESS
No
Do 1 bit operation on MicroLAN using LSBit of next **data_bytes**
Append result to **outbound** buffer
Done, return the (**return**) value

CMD_ ML_DATA

The CMD_ML_DATA gives block level communication with the MicroLAN. The CMD_ML_BLOCK is a multi-byte command so it provides a length byte that must be greater then 0 and one or more data bytes. The first data byte defines the total MicroLAN block length in bytes. The data bytes following the block length are sent to the MicroLAN and the result of that byte communication is placed into a byte in the *outbound* buffer in a multi-byte read format. If the block length is greater then the provided number of data bytes then the remainder of the block length are processes as FF hex bytes. This is normally a read operation from a MicroLAN device. This command uses the DATA_MODE data register for the communication speed at which the block operation is performed on the MicroLAN.

```
Example: Read the first 32 bytes of memory from the MicroLAN memory device
         with the ID number in DATA_ID.

         inbound        CMD_ ML_ACCESS
                        CMD_ ML_DATA <3><length=34><F0, 00>

         outbound       CMD_ML_ACCESS <return byte>
                        CMD_ ML_DATA <34><2 bytes of write data echo and
                                  32 bytes of read data>
```

Figure 3i. Processing command CMD_ML_DATA



Receive *cmd (CMD_ML_DATA), data_length, and data_bytes
Reference
block_length - total number of bytes to send in a block to MicroLAN
*See reference in Table 3
Is data_length >0?
No
Set return to RET_WRITE_ONLY
Yes
Retrieve the first of the data_bytes as the block_length
Check for room in outbound for 2 + block_length (num) bytes (see Figure 2d)
Was there room in outbound ?
No
Set return to RET_OUTBOUND_OVERRUN
Yes
Append command (cmd) and length (block_length) to outbound
Done with data_bytes ?
Yes
Done with block_length bytes ?
Yes
Set return to RET_SUCCESS
No
Do 1 byte operation on MicroLAN using next of data_bytes
No
Do 1 byte operation on MicroLAN using FF hex
Append result to outbound buffer
Append result to outbound buffer
Done, return the (return) value

CMD_DELAY

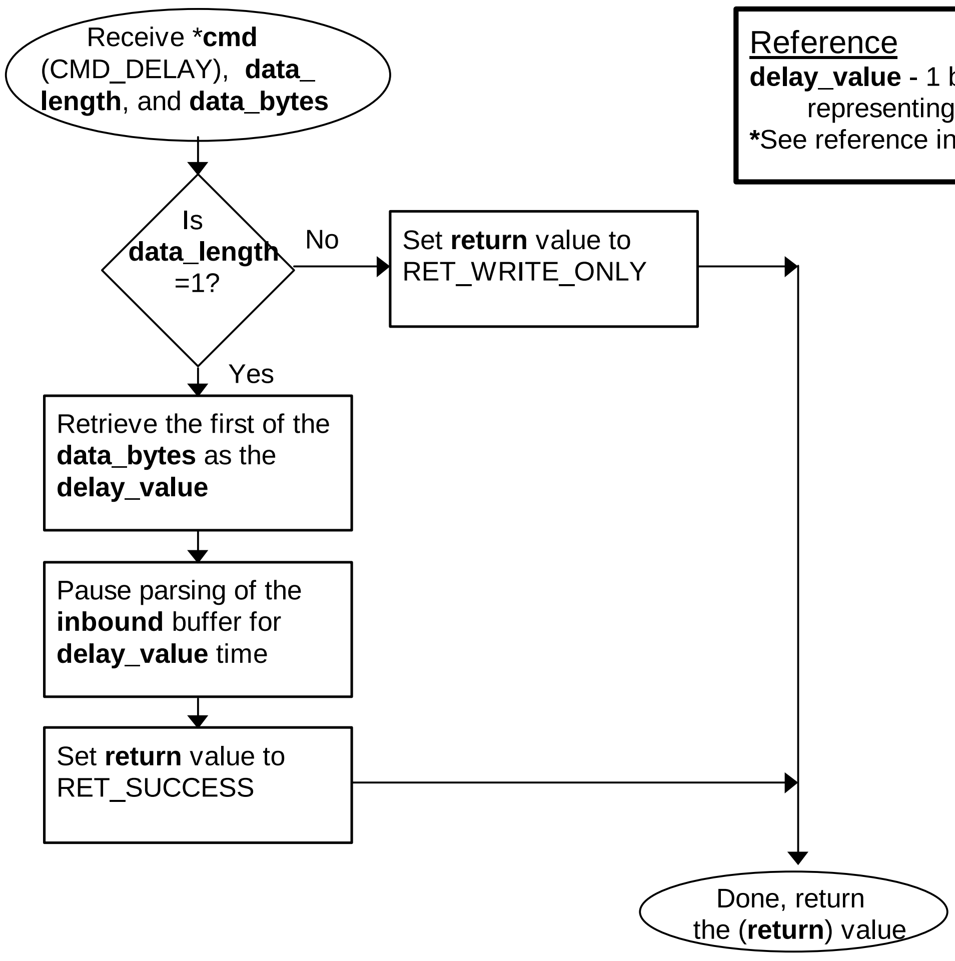
The CMD_DELAY command pauses the execution of the parsing of the inbound buffer by the amount of time specified in the one data byte provided. The delay command must at minimum delay the prescribed amount. It may however go longer. This command is used to time programming and power delivery type MicroLAN functions usually in conjunction with the DATA_MODE command. This one byte value provides a wide range of delay times by providing the following meaning to the bit values. The most significant bit is a flag that when set indicates the value will be in milliseconds and when not set the value is in microseconds. The lower 3 bits represented by X will be used in the following formula 2^(5+X) to give the values displayed in **Table 8**.

```
Example: send a EPROM programming pulse on the MicroLAN.

        inbound         DATA_MODE <04 (hex) (12 volt pulse on)>
                        CMD_DELAY <1><04 (hex) 512 microseconds)>
                        DATA_MODE <00 (hex) (12 volt pulse off)>

        outbound
```

Table 8. Delay byte time values.

Delay Byte	Time
00 (hex)	32 microseconds
01	64
02	128
03	256
04	512
05	1024
06	2048
07	4096
80	32 milliseconds
81	64
82	128
83	256
84	512
85	1024
86	2048
87	4096

Figure 3j. Processing command CMD_DELAY


Receive ***cmd** (CMD_DELAY), **data_length**, and **data_bytes**
Is **data_length** =1?
No
Set **return** value to RET_WRITE_ONLY
Yes
Retrieve the first of the **data_bytes** as the **delay_value**
Pause parsing of the **inbound** buffer for **delay_value** time
Set **return** value to RET_SUCCESS
Done, return the (**return**) value

Reference

delay_value - 1 byte value representing the delay time

***See reference in Table 3**

Appendix

MicroLAN Search Algorithm

Dallas Semiconductor's MicroLAN devices each have a 64 unique ID that is used to address them individually. If the ID's of the devices on the MicroLAN are not know, they can be discovered by going through the *Search Algorithm*. The *Search Algorithm* begins with the devices on the MicroLAN being reset using the CMD_ML_RESET command. It this is successful then the one byte search command is sent. The search command readies the MicroLAN devices to begin the search.

The search command resides in the data register DATA_SEARCH_CMD. The search command is configurable because there are currently two types of searches. The *normal* search command (0F hex) will perform a search with all devices participating. The *alarm* search command (EC hex) will perform a search with only the devices that are in some sort of alarm state. This reduces the search pool to quickly respond to devices that need attention.

The actual search then begins with all of the participating devices simultaneously sending the first bit in their ID. Due to the characteristics of the MicroLAN, this will be a logical AND of the first bit in all of devices. Next, the devices send the compliment of the first bit in their ID. This also is the logical AND of the compliment of the first bit. From these two bits, information about the first bit in the ID is know (See **Table A1**).

Table A1. Bit search information.

Bit	Compliment Bit	Information Known
0	0	there are both 0's and 1's in the bit of the participating ID's
0	1	there are only 0's in the bit of the participating ID's
1	0	there are only 1's in the bit of the participating ID's
1	1	no devices participating in search

The *Search Algorithm* must then broadcast a bit back to the participating devices. If the participating device has that bit value, it continues participating. If it does not have that bit value, then it goes into a shutdown state until the next MicroLAN reset is detected. This 'read two bits', and 'write one bit' pattern is then repeated for the remaining 63 bits of the ID. In this way the *Search Algorithm* forces all but one device to go into the shutdown state. At the end, the ID number of this last device is known. On subsequent passes of the search, a different path is taken to find the other device ID's.

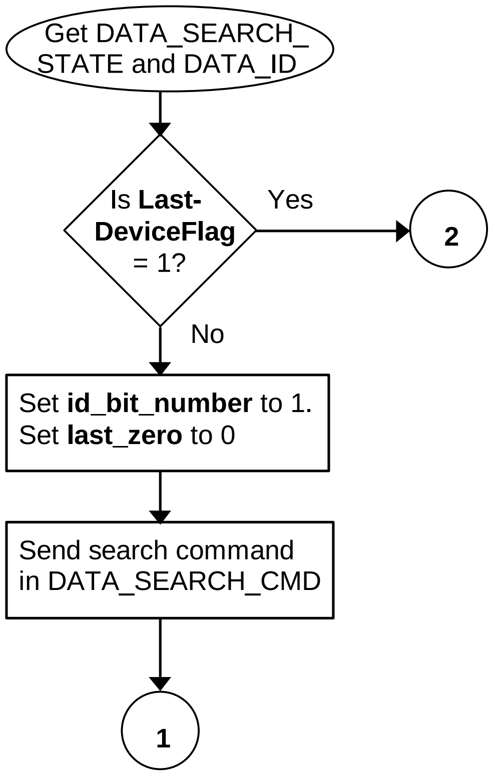
On examination of **Table A1**, it is obvious that if all of the participating devices have a 0 or 1 then that is path that should be taken. The condition where 'no devices are participating' is an atypical situation which may arise if the device being discovered is removed from the MicroLAN during the search. The condition where there are both 0's and 1's in the bit position is called a discrepancy and is the key to the search. The *Search Algorithm* specifies that on the first time through, when there is a discrepancy (bit/compliment = 0/0), the '0' path is taken. The bit position for the last discrepancy is recorded for use in the next search. **Table A2** describes the paths that are taken on subsequent searches.

Table A2. Path direction based on the search bit position and the last discrepancy

Search Bit Position verses Last Discrepancy	Path Taken
=	take the '1' path
<	take the same path as last time (from DATA_ID)
>	take the '0' path

The *Search Algorithm* also keeps track of the last discrepancy that occurs within the first 8 bits of the algorithm. The first 8 bits of the 64 bit ID number is a *family type*. As a result, the devices discovered during the search are grouped into family types. The last discrepancy within that *family type* can be used to selectively skip whole groups of MicroLAN devices. See the description of DATA_ SEARCH_STATE for a description of doing selective searches. The 64 bit ID number also contains an 8 bit cyclic-redundancy-check (CRC). This CRC value is verified to make sure an erroneous ID is not discovered.

Figure A1. Searching the MicroLAN based on DATA_ID and DATA_SEARCH_STATE.


Get DATA_SEARCH_ STATE and DATA_ID
Is **Last-DeviceFlag** = 1?
Yes
2
No
Set **id_bit_number** to 1. Set **last_zero** to 0
Send search command in DATA_SEARCH_CMD
1

Reference

id_bit_number - the ID bit number 1 to 64 currently being searched

last_zero - bit position of the last zero written where there was a discrepancy

id_bit - the first bit read in a bit search sequence. This bit is the AND of all of the id_bit_number bits of the devices that are still participating in the search.

cmp_id_bit - the compliment of the id_bit. This bit is the AND of the compliment of all of the id_bit_number bits of the devices that are still participating in the search.

search_direction - bit value indicating the direction of the search. All devices with this bit stay in the search and the rest wait for a reset.

LastDeviceFlag - flag to indicate previous search was the last device.

LastDiscrepancy, **LastFamilyDiscrepancy** - (see **Table 5**)

Figure A1. (continued)

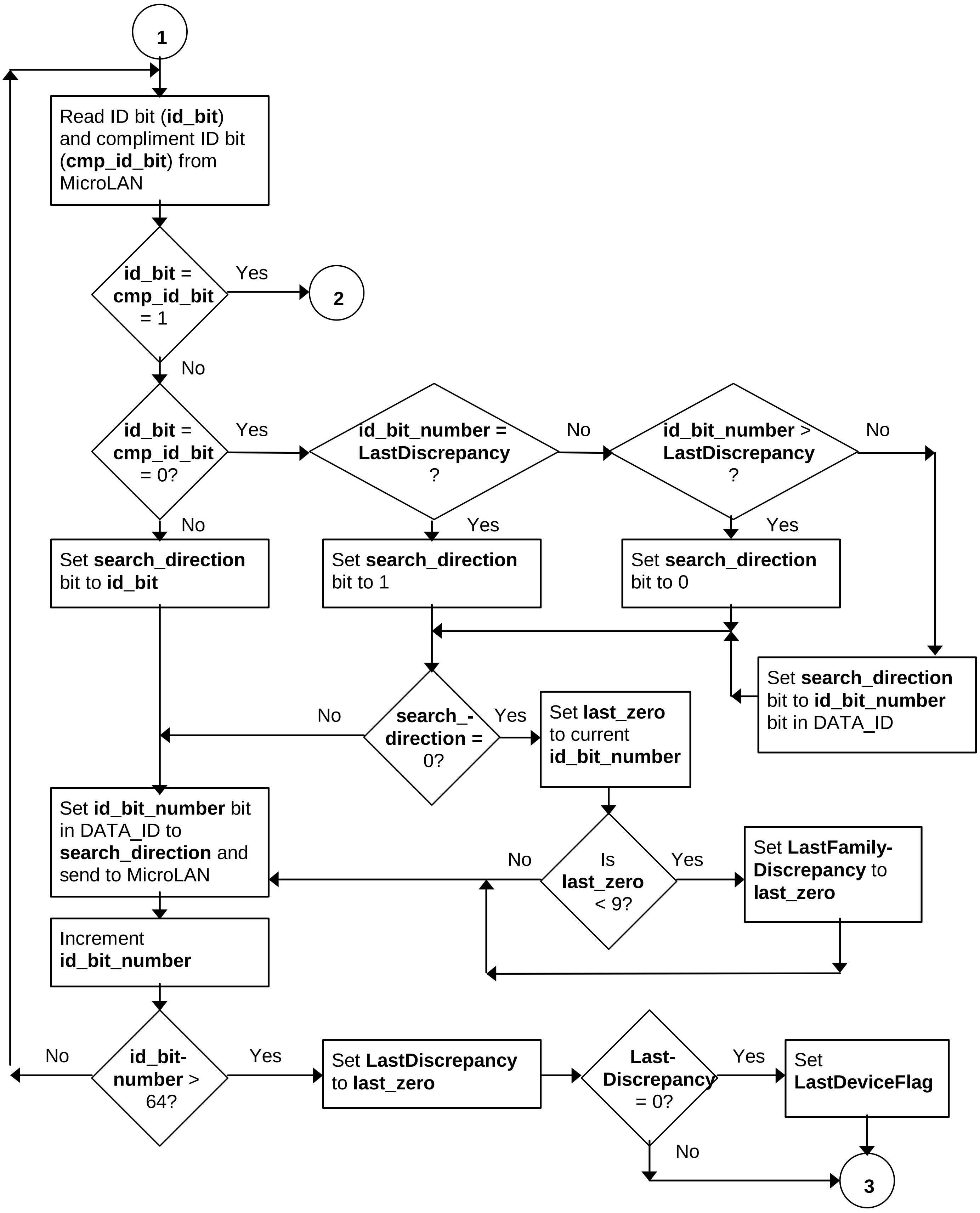
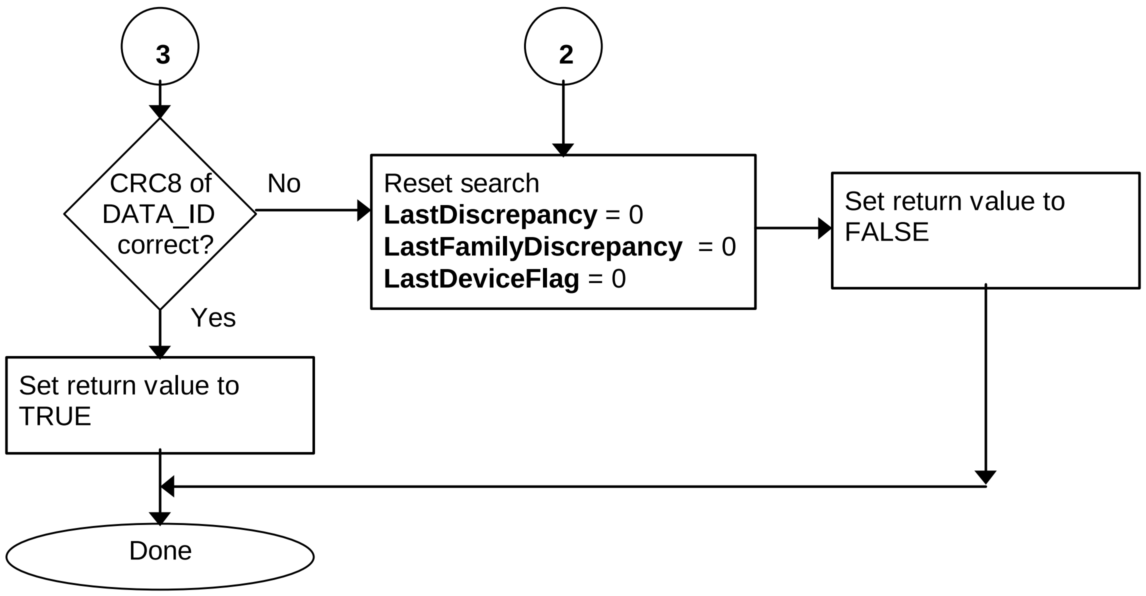

1
Read ID bit (**id_bit**) and compliment ID bit (**cmp_id_bit**) from MicroLAN
id_bit = **cmp_id_bit** = 1
Yes
2
No
id_bit = **cmp_id_bit** = 0?
Yes
id_bit_number = **LastDiscrepancy** ?
No
id_bit_number > **LastDiscrepancy** ?
No
No
Yes
Yes
Set **search_direction** bit to **id_bit**
Set **search_direction** bit to 1
Set **search_direction** bit to 0
Set **search_direction** bit to **id_bit_number** bit in DATA_ID
No
search_-direction = 0?
Yes
Set **last_zero** to current **id_bit_number**
Set **id_bit_number** bit in DATA_ID to **search_direction** and send to MicroLAN
No
Is **last_zero** < 9?
Yes
Set **LastFamily-Discrepancy** to **last_zero**
Increment **id_bit_number**
No
id_bit-number > 64?
Yes
Set **LastDiscrepancy** to **last_zero**
Last-Discrepancy = 0?
Yes
Set **LastDeviceFlag**
No
3

Figure A1. (continued)


3
2
CRC8 of DATA_ID correct?
No
Yes
Reset search
LastDiscrepancy = 0
LastFamilyDiscrepancy = 0
LastDeviceFlag = 0
Set return value to FALSE
Set return value to TRUE
Done