

PD69200M

**Dual Core Host / POE
MCU communication
protocol using shared
memory block**

Version 4.00

External Documents Update

1. **PD69200_UG_COMM_PROT** document was updated to revision 1.21.
 - a. New commands were added.
 - b. Save and Restore commands are marked as not valid for PD69200M.

Table of Content

| | | |
|------|---|----|
| 1 | Introduction: | 4 |
| 1.1 | Purpose | 4 |
| 1.2 | Scope: | 4 |
| 1.3 | References | 4 |
| 2 | Over all description | 5 |
| 2.1 | Block Diagram | 5 |
| 2.2 | Communication Method: | 6 |
| 2.3 | POE Manager communication module Architecture | 7 |
| 2.4 | Communication data exchange structure | 8 |
| 2.5 | Protocol Version | 10 |
| 2.6 | Communication Control Bytes: | 11 |
| 2.7 | Communication data exchange structure value settings after system power up | 13 |
| 2.8 | Communication data exchange structure value settings after Steady state (No new communication from Host) | 13 |
| 2.9 | Read / Write operation sequence description | 14 |
| 2.10 | General purpose Read / Write registers | 17 |
| 2.11 | Interrupt out to Host | 19 |
| 2.12 | Keep alive function | 21 |
| 2.13 | DN Frequency | 21 |
| 3 | Boot up flow (PowerUp) | 22 |
| 3.1 | Dragonite Reset from Host | 23 |
| 3.2 | Dragonite Self Reset | 23 |
| 3.3 | Dragonite Communication Error Counter | 24 |
| 4 | Debug Options | 24 |
| 4.1 | Dragonite | 24 |
| 4.2 | Host Debug Application | 29 |
| 5 | Code Delivery | 34 |
| 5.1 | Code Delivery | 34 |
| 5.2 | CRC Calculation | 34 |
| 5.3 | mssc tool | 35 |
| 5.4 | AC3x - Aldrin | 37 |
| | Appendix – A – Shared Memory Addresses @ POE DTCM | 39 |
| | Appendix – B – POE Reset influence on protocol flow | 44 |
| | Change Record | 51 |

1 Introduction:

1.1 Purpose

The purpose of this document is to describe the communication protocol between a Host CPU that controls and monitors POE Manager CPU, when both of them share a memory block for communication. In addition this document describes the necessary flows that need to be executed on each one of the CPUs to enable this protocol.

1.2 Scope:

This document is intended for the SW developer, the SW customers and software QA.

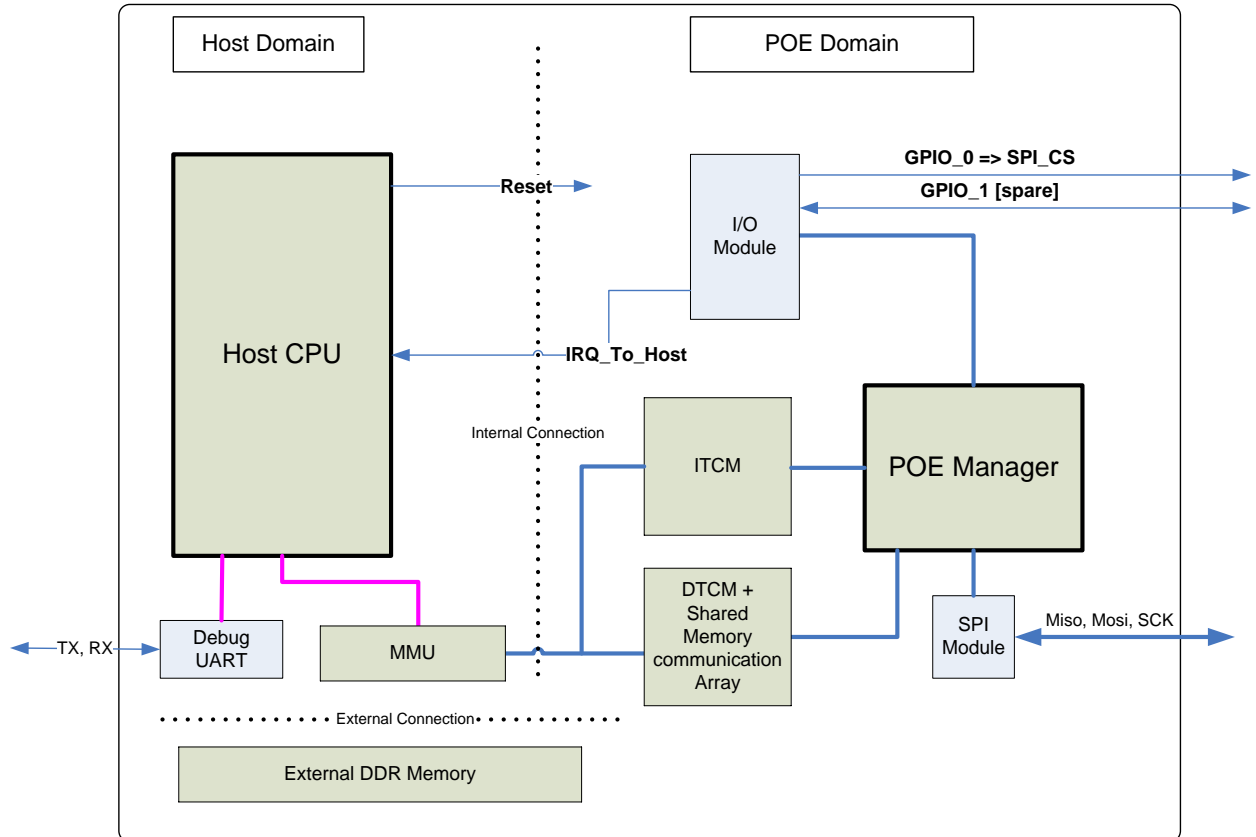
1.3 References

MSCC - PD69200 15 byte communication protocol.

2 Over all description

2.1 Block Diagram

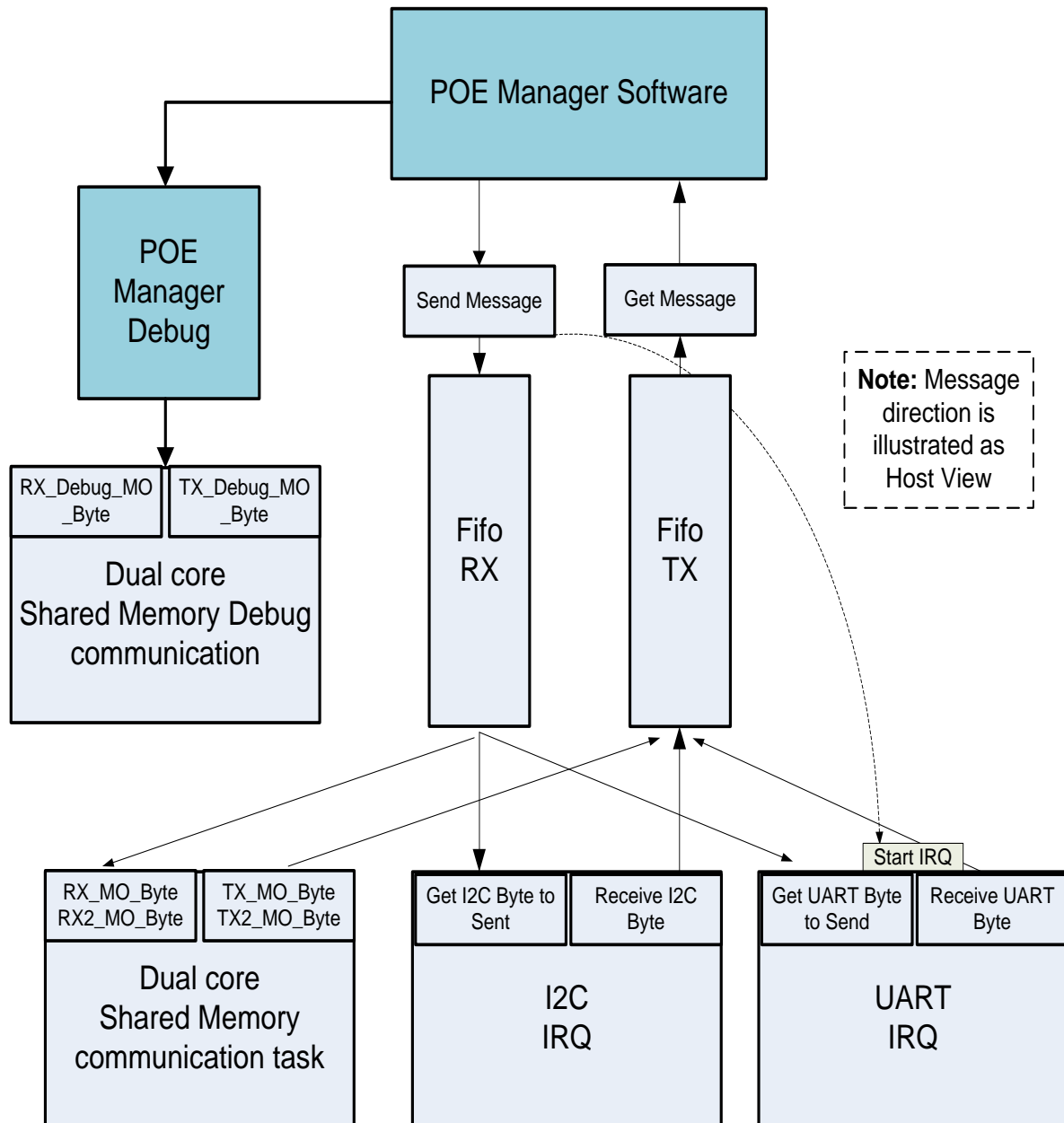
This block diagram describes the connection between the Host and the POE Manager.



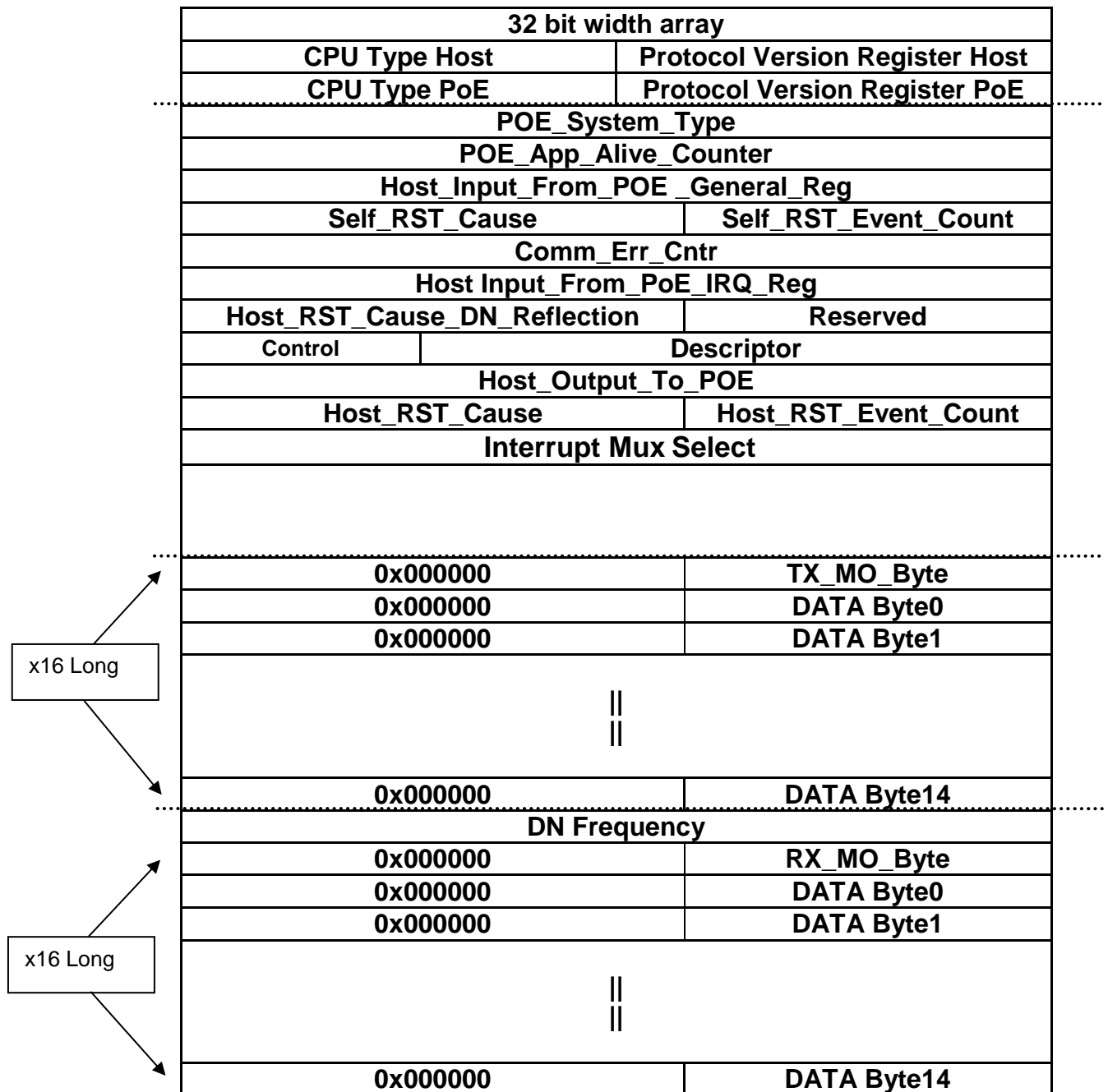
2.2 Communication Method:

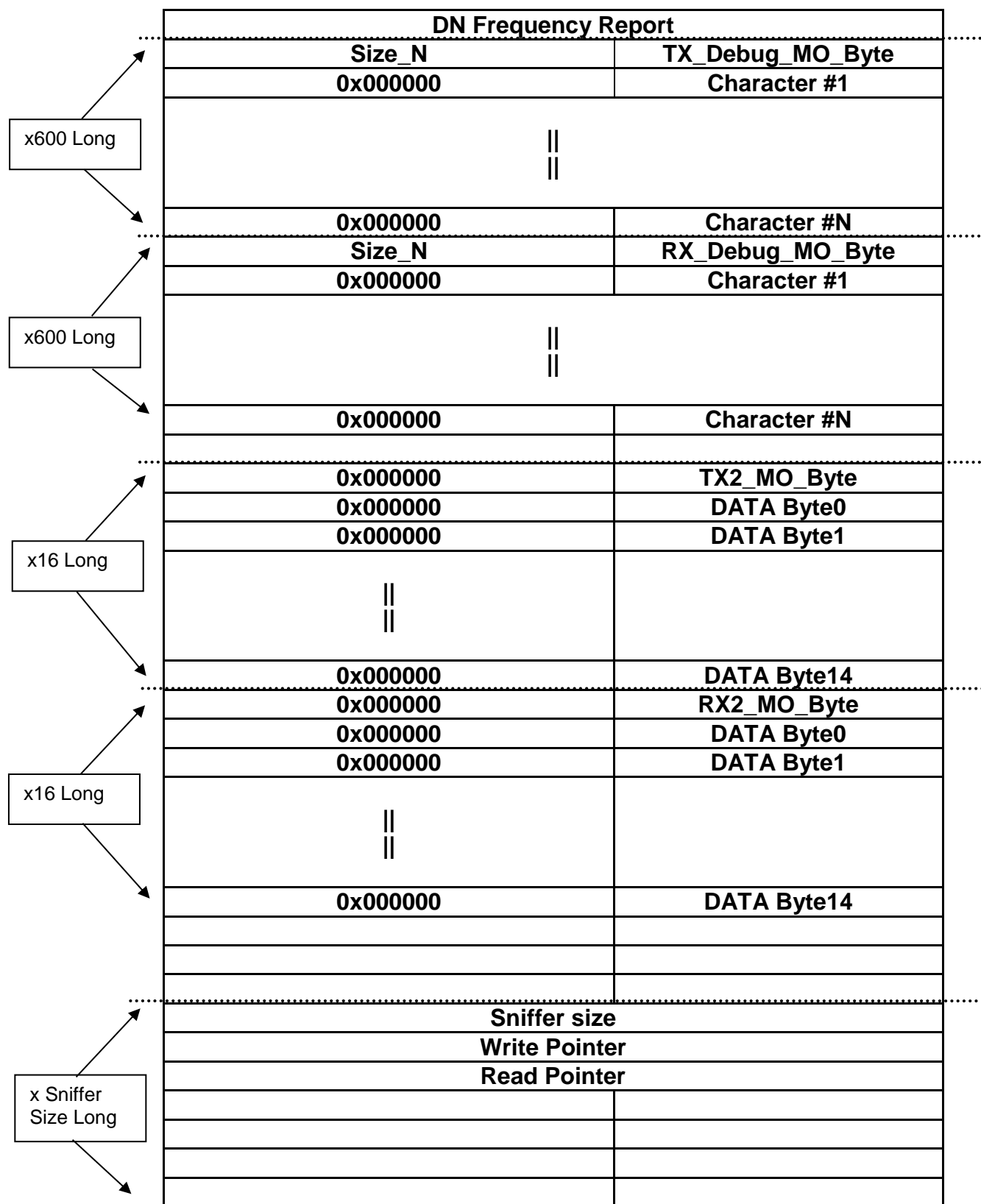
The communication is based on flags that direct the information exchange flow. The information inside the message is based on 15 byte protocol that is being used in PD69200 POE manager solution. The concept is that instead of receiving and transmitting information through UART using an RX/TX fifos of 15 byte, the 15 byte information is being copied directly (memcpy operation) by each one of the CPUs. The MO_Byte is used for data flow control, similar to the fifo information size check that is being used in UART solution. The protocol remains asynchronous half duplex. In addition to the 15byte protocol transfer, a special debug area was added to enable customer support debug. The debug area is split to 3 sections – Host / PoE messages sniffer, PoE dedbug prints, secondary PoE control. The debug will be described in a dedicated chapter.

2.3 POE Manager communication module Architecture



2.4 Communication data exchange structure





2.5 Protocol Version

Each side will deliver the protocol version that it supports.

Based on the register value, the Host can recognize version mismatch that will not allow using new communication commands/requests.

Protocol Version Register Host [15..0] – The Host should update this register with the supported protocol version number, prior to releasing the POE manager from Reset state. The Host can write and read it's protocol version. The POE side can only read this register.

CPU Type Host [31..16] – The Host should update this register with the expected DN type, prior to releasing the POE manager from Reset state. The Host can write and read this register value. The POE side can only read this register.

Existing Valid values:

| DN type | Register Value | Note |
|------------------------|------------------|-------------------------|
| ARMv5 | 0x0000 or 0xFFFF | the first DN AC and AC3 |
| ARMv5 with freq change | 0x0001 | AC3x |

How to test : Host write 0x0001 to the CPU Type Host register and should read back the same value from CPU Type PoE register. Next the Host should repeat the procedure with a different value.

Protocol Version Register PoE – The Host application can only read this register. This register is updated by the PoE Manager. The PoE Manager can read and write this register. The value of this register should be 0xFFFFFFFF after boot up (Set by the Host), prior to any PoE code execution.

| Protocol Version | Added features |
|------------------|--|
| 0x0001 | Initial protocol features |
| 0x0002 | Additional support for sniffer, secondary interface and Dragonite interrupt mask register (Control Dragonite IRQ to Host output) |
| 0x0003 | Adding descriptor |
| 0x0004 | Adding DN type support |

How to operate with MSCC tool: [file shard_mem.c --> function int Initshard_mem(void)]

2.6 Communication Control Bytes:

The data exchange between the two CPUs is based on polling and updating a communication control fields, located in the shared memory area. The data exchange structure is split into 2 sections. One for 15 byte data exchange and the 2nd for debug information exchange. The control fields are defined accordingly.

- TX_MO_Byte is used for transmitting data from the Host to the POE Manager. The allowed values of this byte are:
 - 0xA0 – Host Ownership.
 - Host can update the 15 byte information.
 - The information is not valid for POE manager.
 - 0x0A – POE Ownership.
 - Host sent a message to the POE manager and the information is valid for the POE Manager. Host is not allowed to change the data as long as the value of this byte is 0x0A.
 - 0xFF – Init value (Set by the Host as part of boot up sequence), will be updated by POE to 0xA0, after POE release from reset. The data structure information is not valid.

*How to test : The Host should set an unsupported value (0xBB). The PoE is expected to change the value to 0xA0 and increment the communication error counter.
The Host should verify the described PoE behavior.*

- RX_MO_Byte is used for receiving data from the POE Manager, by the Host. The allowed values of this byte are:
 - 0xB0 – Host Ownership.
 - Host can Read the 15 byte information.
 - The information is valid and was replied from the POE manager.
 - The POE manager cannot update the information as long as the value of this byte is 0xB0.
 - 0x0B – POE Ownership.
 - Host read the last message and the area is ready for new reply from the POE manager. The host is not allowed to read data as long as the byte value is 0x0B.
 - 0xFF – Init value (Set by the Host as part of boot up sequence), will be updated by POE to 0x0B, after POE release from reset. The data structure information is not valid.

*How to test : The Host should set an unsupported value (0xAA). The PoE is expected to change the value to 0x0B and increment the communication error counter.
The Host should verify the described PoE behavior.*



Note: During system power up, communication data exchange structure bytes must be configured by the Host to 0xFF, prior to releasing the POE manager from Reset.

How to operate with MSCC tool:

[file shard_mem.c --> function int Get_shard_mem_msg(char *buff, int size, long type, long *read_size)
and int Send_shard_mem_msg(char *buffer, long sizeofmsg, int type)]

2.7 Communication data exchange structure value settings after system power up

All data exchange structure area bytes must be filled with 0xFF by the host, after power up, prior to releasing the POE manager from reset.

When the POE manager is out of reset it will start executing its Application will update the communication control bytes and the relevant protocol registers.

As long as the ownership registers do not contain the exact host ownership value, the host is not allowed to use the structure data.

In addition, as long as POE Application is executing the watch dog counter

“POE_App_Alive_Count” will be advanced.

If the PoE application is not running from any reason, the “POE_App_Alive_Count” will stop counting.

2.8 Communication data exchange structure value settings after Steady state

(No new communication from Host)

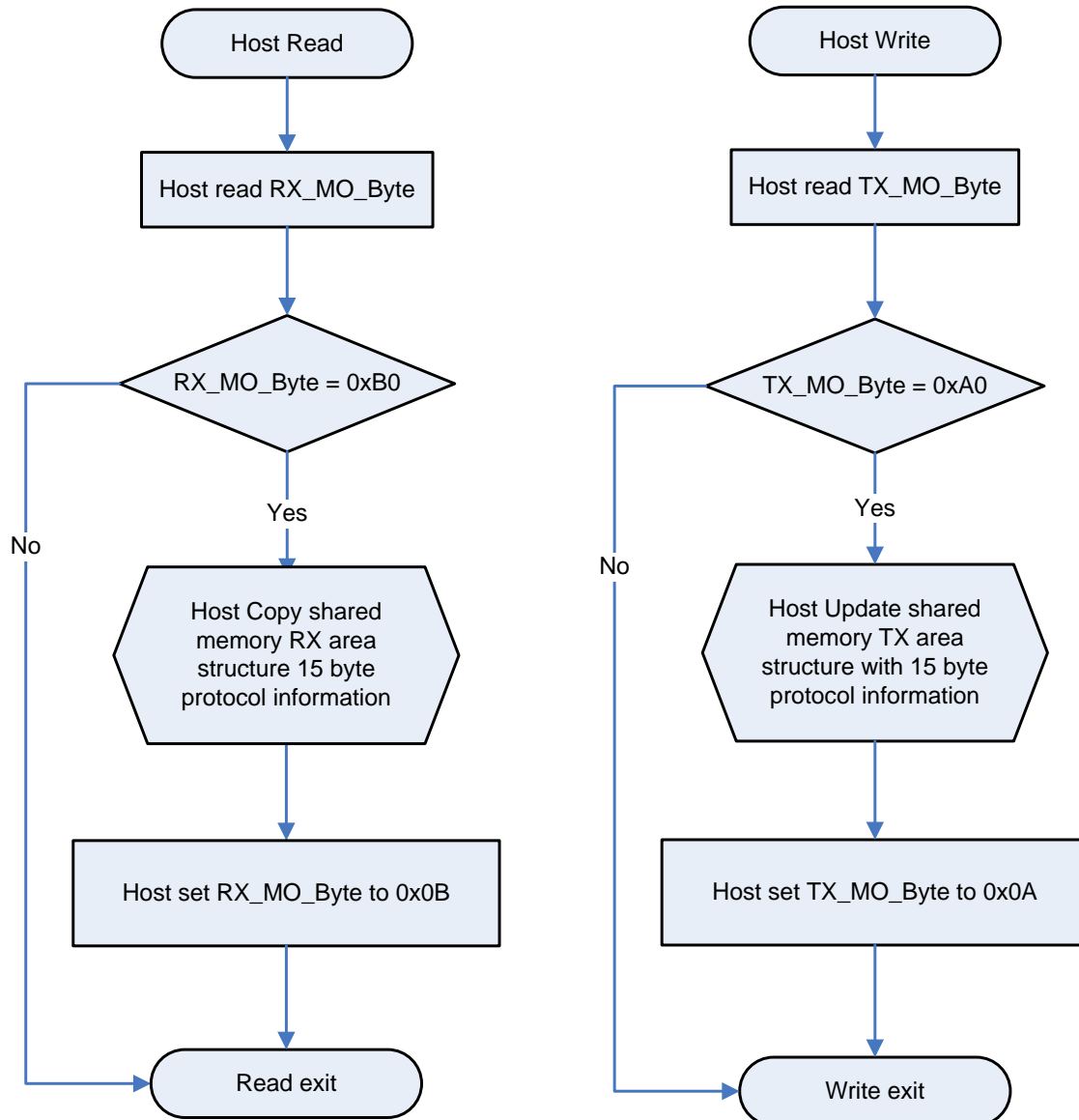
TX_MO_Byte is set to “Host Ownership” (Ready for new message from Host).

RX_MO_Byte is set to “POE Ownership” (Ready for new reply from POE).

| | |
|---|----------------------|
| Protocol Version Register Host: != 0xFFFFFFFF | |
| Protocol Version Register PoE: != 0xFFFFFFFF | |
| POE_System_Type | |
| POE_App_Alive_Counter: = Counting | |
| Host_Input_From_POE_General_Reg | |
| Self_RST_Cause: = Last | Self_RST_Event_Count |
| Comm_Err_Cntr | |
| Host Input_From_PoE_IRQ_Reg | |
| Host_RST_Cause_DN_Reflection | Reserved |
| Control | Descriptor |
| Host_Output_To_POE | |
| Host_RST_Cause: = Last | Host_RST_Event_Count |
| Interrupt Mux Select | |

2.9 Read / Write operation sequence description

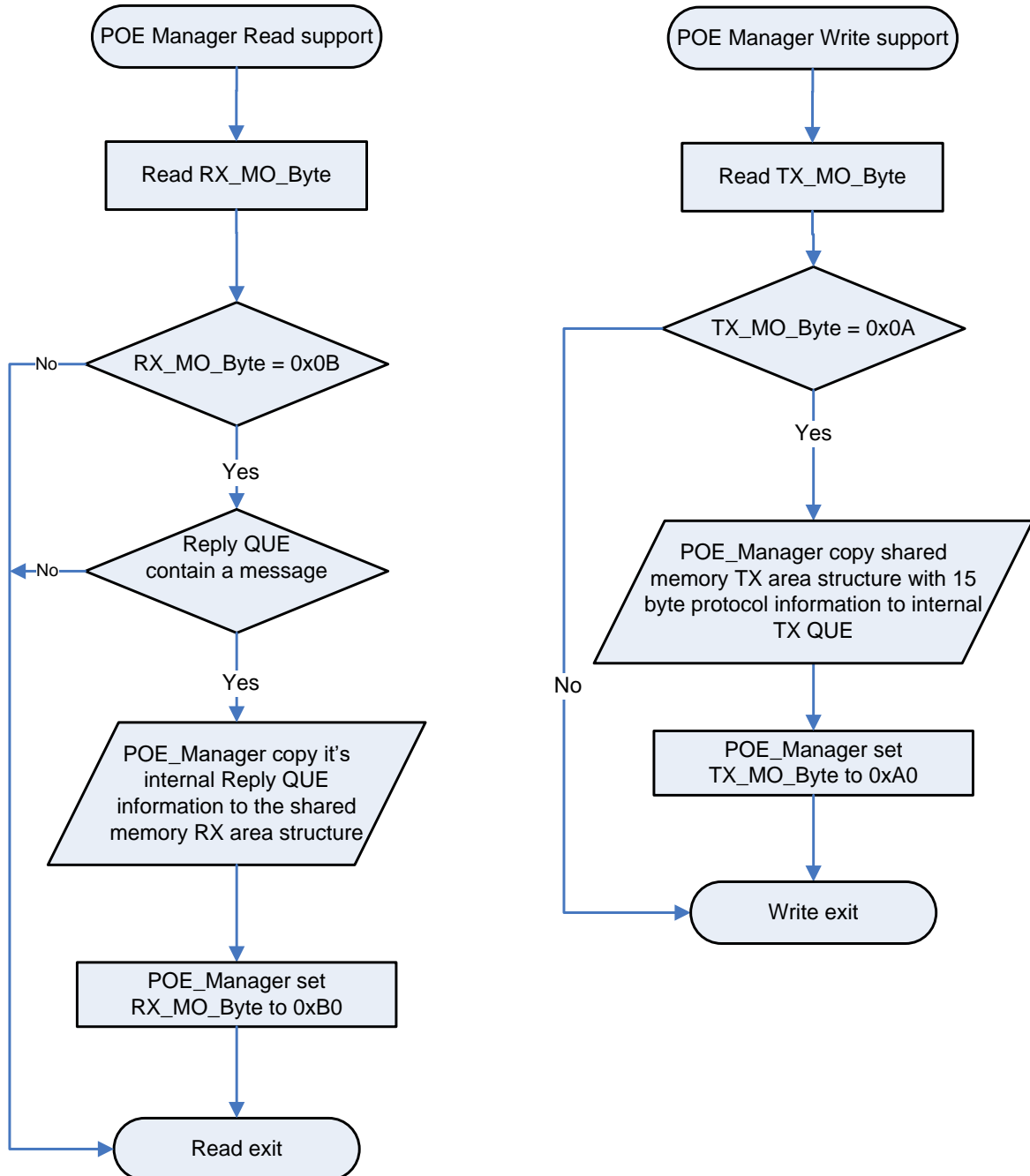
2.9.1 Host Read/Write flow



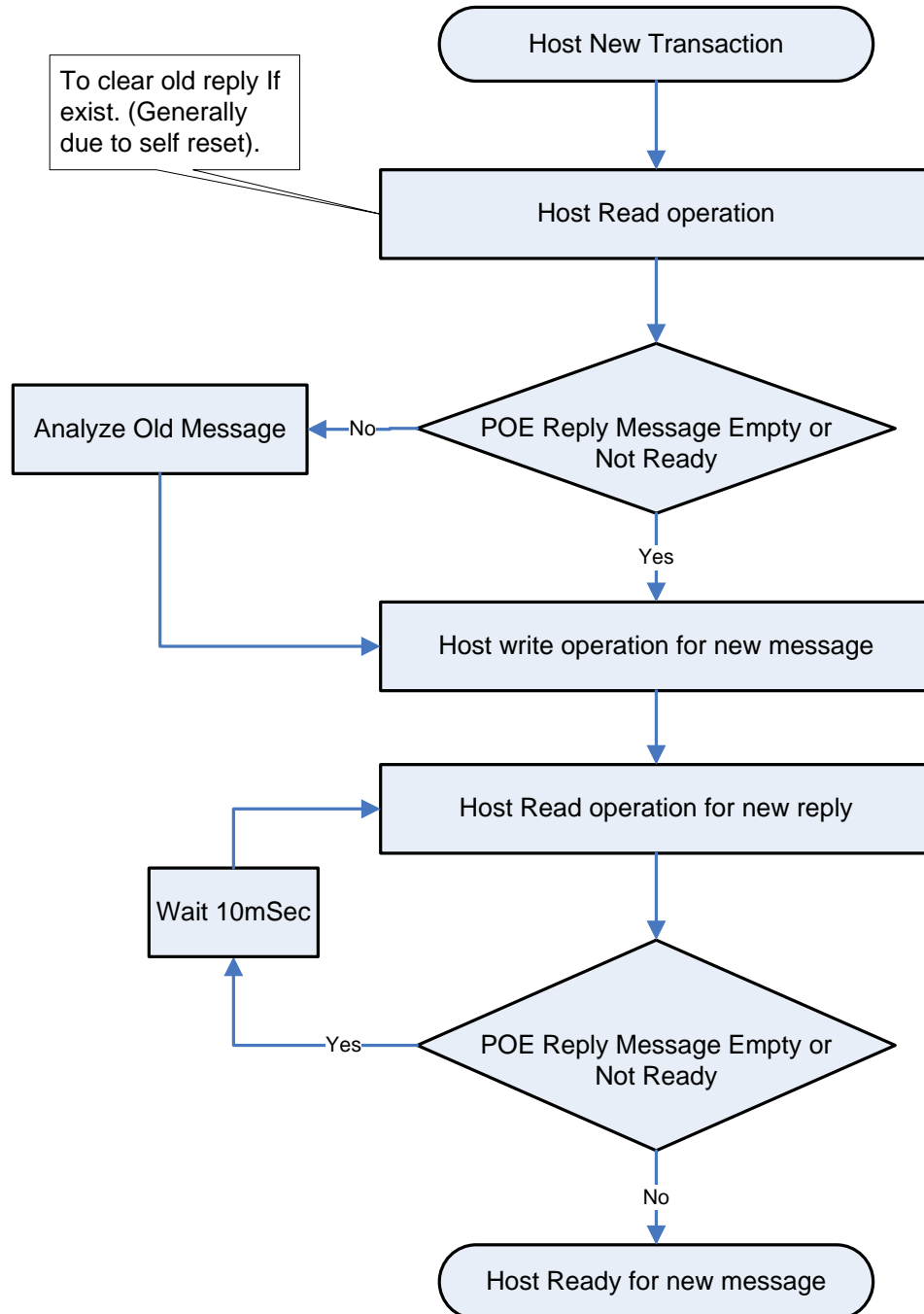
How to operate with MSCC tool:

[file shard_mem.c --> function int Get_shard_mem_msg(char *buff, int size, long type, long *read_size) and int Send_shard_mem_msg(char *buffer, long sizeofmsg, int type)]

2.9.2 POE_Manager Read/Write flow



2.9.3 Host Full Transaction Flow with POE Manager



Note: Please refer to 15 byte communication protocol description document

2.10 General purpose Read / Write registers

There are 3 x 32 bit registers for general purpose use, simulating I/O controls and reflecting POE_IRQ register.

1. **Host_Output_To_POE** – Read/Write register from host to POE manager. The POE manager will poll the value of this register. This register is a read only by POE.

Bit 0 of this register will be used for Disable ports operation, instead of the existing input xDisable_Ports in the generic PD69200.

How to test : Clear the Bit ('0'), the PoE ports are OFF, Set Bit to '1', PoE ports are ON.

Host_Input_From_POE_General_Reg - Read only register by Host from POE manager. The Host will poll the value of this register. The POE can perform Read/Write operations on this register.

Bit 0 – SystemOK_Pin:

This register bit is used for reflecting the inverted value of xSystem_status pin, instead of the existing output in the generic PD69200.

This bit will be set to 1 when the PD69200 is ready for communication and Vmain is inside the defined valid range.

Bit 1 – Message Ready:

This register bit is used for reflecting Message Ready, instead of the existing I/O in the generic PD69200.

When a reply message is written to the shared memory and it is valid for Host, this bit will be set. The bit will be cleared when MO flag returns to the Dragonite.

How to test : test by sending a Message and not read the answer. Read this bit, then read the Message and read this bit.

Bit 2 Fan_Control: This register bit is used for Fan Control, instead of the existing output in the generic PD69200. This bit will be set to '1' when the temperature of one of the PD69208 devices in the system will go higher than thermal alarm threshold.

How to test : Set thermal alarm threshold to 10degC while the product in room temperature by using relevant 15 byte protocol command. Verify that the bit is set. Return to 125degC and verify that the bit is cleared.

Bit 3 Int_Out:

This register bit is used to reflect Int_Out (When interrupt event is reported and enabled), instead of the existing output (xInt_Out) in the generic PD69200.

How to test : testing this bit by enable the interrupts using relevant 15 byte protocol command.

*generate PoE event like disconnecta port.
Verify the the bit is set.
Read the 15 byte system status and verify that the bit is cleared.*

2. Host Input_From_PoE_IRQ_Reg

The Host will poll the value of this register. The POE can perform Read/Write operations on this register.

The value is a reflection of the IRQ register which defined at the 15 BYTE protocol document.

2.11 Interrupt out to Host

2.11.1 Interrupt out I/O to host

The dragonite can generate an Interrupt to the Host (See block Diagram).

The interrupt is a logical '1' pulse generated by the DN software with time duration of about 25nSec.

The interrupt cause can be read in Host_Input_From_POE_General_Reg.

When one of the bits in this register is set and "Interrupt Select Mux" was set to the relevant option, an IRQ pulse will be generated by the Dragonite.

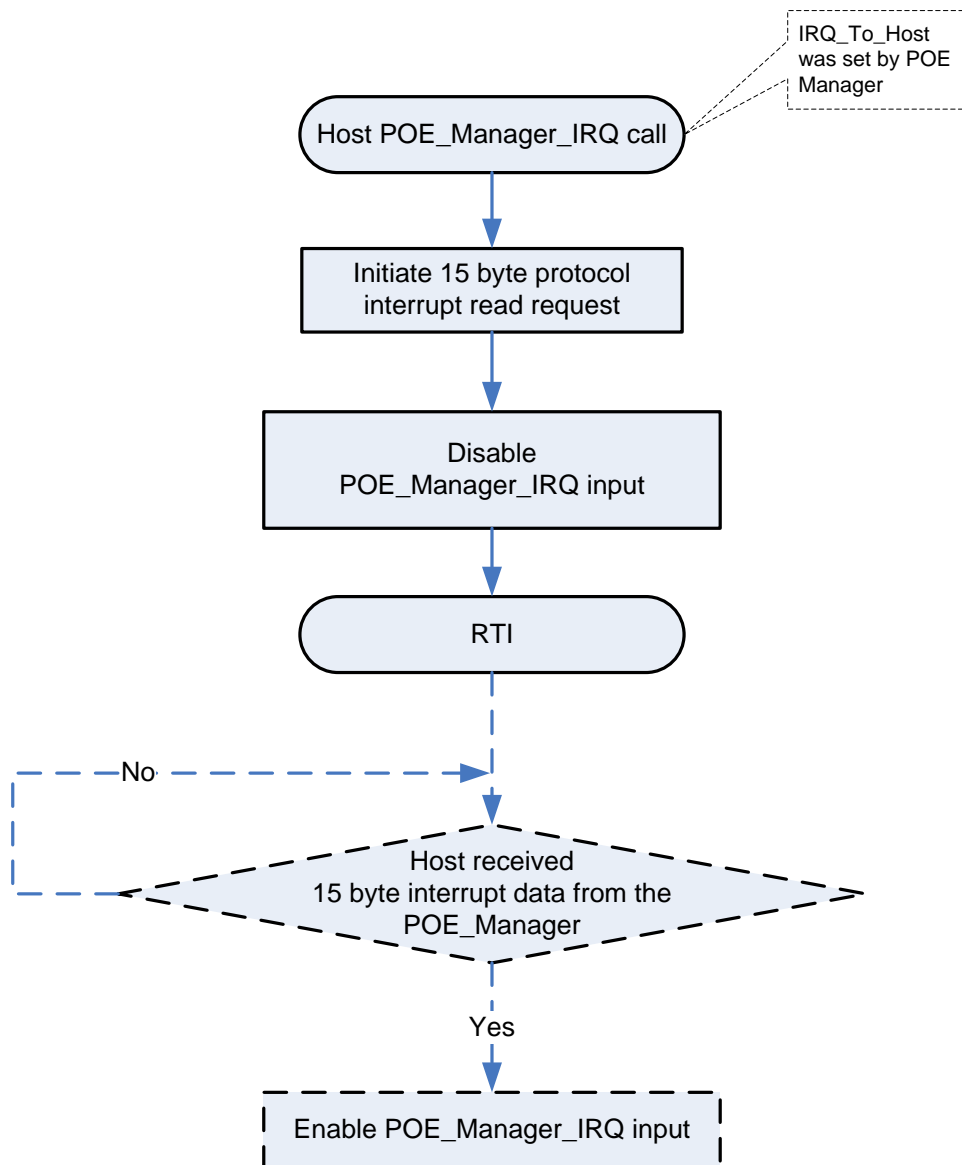
2.11.2 Interrupt Mux Select

Read/Write register enables the host to select one of the causes that will toggle IRQ_To_Host signal.

| Value | Selection | Note |
|--------------|---------------|---------------------------------|
| 0x00000000 | Disable | Output is not active always '0' |
| 0x00000001 | SystemOK_Pin | Output = High pulse |
| 0x00000002 | Message Ready | Output = High pulse |
| 0x00000003 | Fan_Control | Output = High pulse |
| 0x00000004 | Int_Out | Output = High pulse |
| 0xFFFFFFFF | Disable | Output is not active always '0' |
| Other Values | Reserved | Do not use |

Note: The IRQ will be operate only if the Protocol version is set to 2 or higher.

2.11.3 Host interrupt flow



2.12 Keep alive function

The Host low level driver should check that the POE manager is alive, recommended in case of communication timeout, by monitoring POE application alive counter.

PoE_App_Alive_Counter

The counter is incremented by the Dragonite FW.

If the counter is halted for minimum 100mSec, it means that the POE manager was stopped working properly and a Dragonite Reset is required.

How to test :

In regular DN operation verify that the counter value is incremented.

hold the DN Reset and verify that the counter is halted on the same value .

2.13 DN Frequency

AC3x device supports DN frequency change.

The DN module do not have a frequency configuration register or any register information about it's working frequency.

Previous AC3 devices where set up with fix operating frequency of 200MHz.

To support the new AC3x with variable frequency, new registers were added to the shared memory register map.

DN Frequency – The Host shell write to the DN its operating frequency. If the value doesn't match the actual DN frequency that was set by H.W the PoE SPI frequency will be wrong and may lead to communication failures with the PD69208 devices.

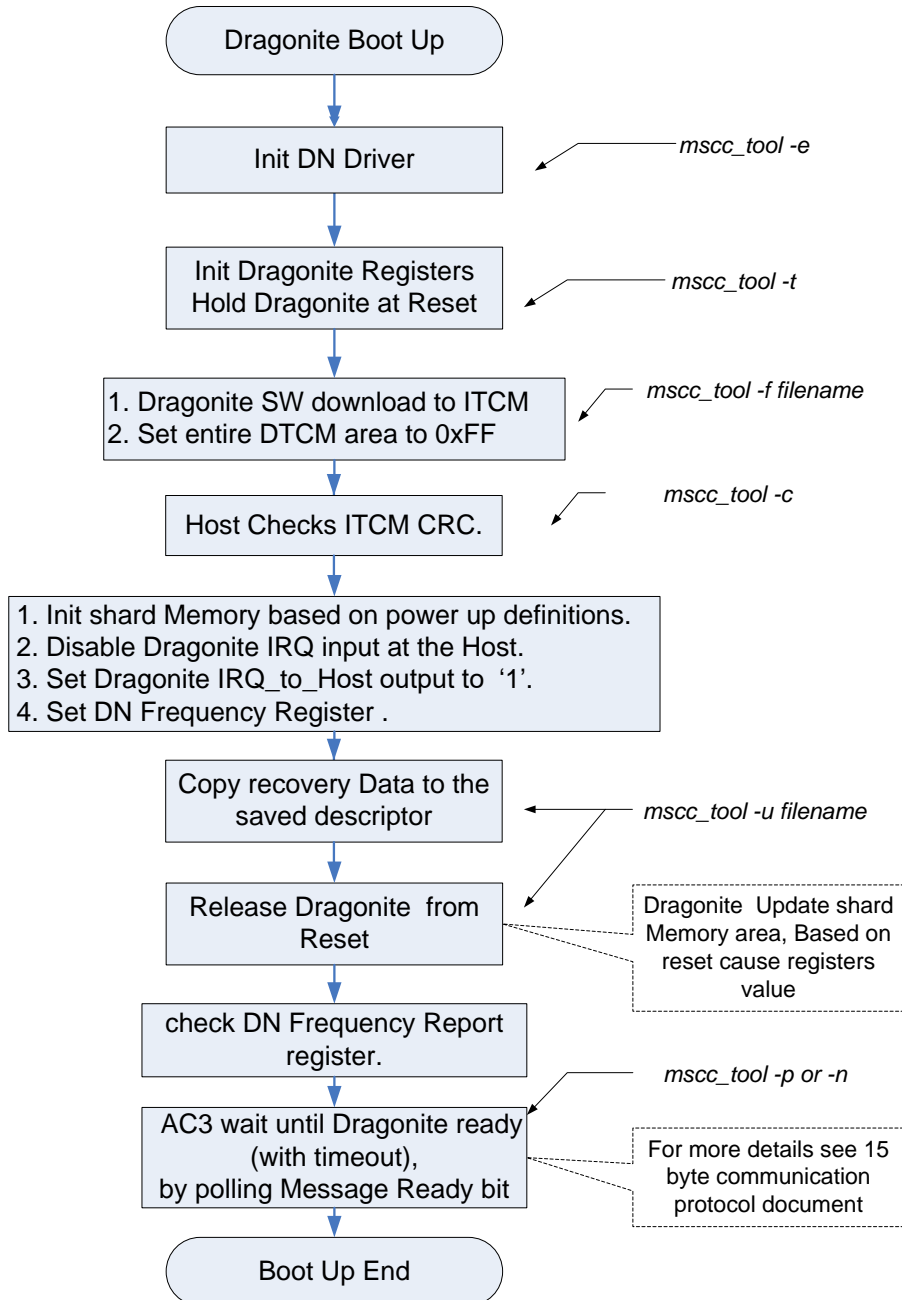
DN Frequency Report - The DN report to the Host which frequency the DN choose to work.

If both registers values do not match, it indicates that the DN FW did not find any suitable SPI frequency divider to work with. The Host must change the DN frequency and restart the DN.

Note: The registers value is in Hz.

Example: 250MHz = 250000000 = 0x0EE6B280

3 Boot up flow (PowerUp)



3.1 Dragonite Reset from Host

| Host_RST_Cause | Host_RST_Event_Count |
|----------------|----------------------|
|----------------|----------------------|

This register set is used to inform the POE software about the reason that the host decided to reset the POE MCU. The event counter is 16 bit a free running incremental counter that needs to be incremented one step on each reset event. After POR the event counter value should be set by the host to 0xFFFF.

The Host_RST_Cause is 16 bit defined by specific number for each event type.

- POR – 0xFFFF
- POE Not responding – 0x0001
- Application request – 0x0002
- Communication errors – 0x0003
- Host Application Reboot – 0x0004
- Undefined cause – 0x1000
- ETC ...

The value of this register should be updated by the Host on each reset event before Dragonite is taken out of reset. The Dragonite will clear the value of this register after read. The host is not allowed to write any value to this register while the Dragonite is not held in reset state.

3.2 Dragonite Self Reset

| Self_RST_Cause | Self_RST_Event_Count |
|----------------|----------------------|
|----------------|----------------------|

This register set is used to inform the Host about the reason that the POE decided to generate a self-reset. The event counter is 16 bit a free running incremental counter that needs to be incremented one step on each reset event. After POR the event counter value should be set by the host to 0xFFFF.

The Self_RST_Cause is 16 bit defined by specific number for each event type.

- POR – 0xFFFF (Set by host as part of memory init after POR)
- Reserved – 0x0001
- Internal Watch Dog reset – 0x0002
- Other reason described in communication protocol – 0x0003
- External reset – 0x0004
- Undefined cause – 0x1000
- ETC ...

How to test : testing by generate DN reset. Release the reset, read 0x0004 and verify that the counter value was increment . Repeat several times.

3.3 Dragonite Communication Error Counter

| |
|----------------------|
| Comm_Err_Cntr |
|----------------------|

This Register counts up whenever an unknown ownership value is recognized. When such error is being detected the Dragonite update the ownership value based on reset values.

How to test : Testing by generate communication error from host like wrong RX_MO_BYTE etc, this value need to be increment

How to operate with MSCC tool:

*[file shard_mem.c --> function long save_recover_to_file(char *filename, int backup)]*

4 Debug Options

The shared memory protocol contains additional areas to enable PoE debug. This functionality is necessary since the Dragonite is part of the Host silicon and there is no option to connect external equipment to analyze Host and PoE data exchange or to add external debug control. In addition there is no UART at the Dragonite. Secondary UART is available only at the Host (See block diagram 2.1) For proper debug, sections 4.1, 4.2 and 4.4 must be implemented in both sides (Host and Dragonite).

The debug is designed in such way that messages will go through the Host secondary UART to the Dragonite shared memory. The Host must enable this UART for debug and implement a Debug application that will use it.

How to operate with MSCC tool:

[file rx.c --> function int check_message_to_gui(void)]

[file tx.c --> function int check_message_from_gui(void)]

4.1 Dragonite

4.1.1 Sniffer

The sniffer enables to see messages transfer between the Host and DN. The implementation imitates the operation of sniffing equipment for communication lines.

Note: The sniffer is not working for debug interfaces

How to operate with MSCC tool:

[file rx.c --> function void run_sniffer(void)]

4.1.1.1 Sniffer FIFO

The FIFO contains a Header and in addition can contain 15 byte TX/RX message information or Descriptor information for more complex data reports or both types. The FIFO Header is used to inform the capabilities (By Version) and monitor the FIFO activity (By size and counters).

The FIFO has Write counter and Read counter.

The FIFO size is predefined by sniffer size register (defined by the Dragonite).

The write counter (WC) will be R/W from the DN and Read only from the Host.

The Read counter (RC) will be R/W from the Host and Read only from the DN.

If the difference between RC and WC will be equal to sniffer size it indicates an over run condition.

| FIFO Header | | | |
|-------------|--------------|--------------------|--------------------|
| Info Type | Upper 16 bit | Lower 16 bit | Host R/W Direction |
| FIFO Header | Version | Sniffer Size | RO |
| | | Write counter (WC) | RO |
| | | Read counter (RC) | R/W |

| FIFO Information | | | | |
|------------------|--|---|-----------------|--------------------|
| Info Type | Control Byte [31..24] | Information [23..0] | | Host R/W Direction |
| 15 Byte Data | See Following description bit 7 = 0 | Time stamp (16 bit) | DATA (8 bit) | RO |
| Descriptor | See Following description bit 7 = 1 | Descriptor Data Pointer (24 bit) | | RO |
| Details | Bit 0 – FIFO Overflow Bits [5..1] - Reserved Bit 6 – Descriptor is unavailable Bit 7 – Information Type 0 – 15 byte data 1 – Descriptor pointer | <u>Time stamp</u> : Dragonite timing in steps of 1mSec. <u>Data</u> : A single byte from a 15 byte protocol message. <u>Descriptor</u> : A pointer to Dragonite internal RAM that stores variable length information. | | RO |

4.1.1.2 FIFO management and messages order

The Dragonite is responsible for FIFO writes. When the Dragonite get a 15 byte received message from the Host ,it writes to the FIFO. When the Dragonite send a 15 byte message it writes to the FIFO, together with message ready signal to the Host. The message order in the FIFO will be: Host message 1st, Dragonite reply 2nd. All message bytes will contain time stamping.

The data can also contain a descriptor. A descriptor will appear in the FIFO when pre-defined debug scenario will occur generating this message.

The Host should contain a special PoE debug application that can read the FIFO messages and print them to a Debug UART, including time stamping.

At each FIFO write, the Dragonite will increment the Write Counter value. At each Host read from the FIFO, the Host will increment the Read Counter value.

In case of descriptor in the FIFO, the descriptor is a pointer to a memory with more data. The host should read the information in the memory according to 4.2.2. 4) and print the descriptor information.

The memory structure is:

| | | |
|--------|---|---|
| Header | Type (16 bit) – For MSCC use | Size (16bit) – The number of Data longs |
| | Time stamp (32 bit) – Dragonite timing | |
| Data | DATA (32 bit) | |
| | DATA (32 bit) | |
| | DATA (32 bit) | |
| | ... | |

4.1.1.3 FIFO Manage

The Dragonite is responsible to manage the FIFO, check for FIFO errors and write the information to it. The Host is responsible to read the information from the FIFO and update the Read Counter. After Dragonite is out of reset it will set write counter and the read counter to zero (0x0). In addition the Dragonite will set the sniffer size register (This value defines the FIFO size).

The Sniffer Size range can be 0x1E (30) till 0x1000.

Each value at the FIFO is long.

Before initiating a host read from the FIFO, the Host must verify that write counter value is bigger than the read counter value by at least 30 Bytes. In addition, it is recommended to check that a message ready signal was received by the Host.

When the FIFO is in use, the Host should read the FIFO information first and then increment the read counter.

The Dragonite should write the data first and then increment the write counter.

The write counter & read counter are cyclic up to Sniffer Size.

The Dragonite will set the OVR bit when the distance between the write and read counters are equal or above the Sniffer size.

4.1.2 Secondary 15 byte control and telemetry

The shared memory contains a main TX/RX 15 byte protocol communication area between the Host and the Dragonite. Exact additional TX2/RX2 15 byte protocol

area exists for external control that can bypass the Host or work in parallel to the Host.

This area is used to check PoE functionality by using external PC applications, connected through the debug UART.

To enable this functionality the Host must have a special command that can halt any PoE activity and any Dragonite monitoring. In addition the Host must contain debug tunneling function that can transmit / receive messages from the debug UART to the shared memory TX2/RX2 area.

How to operate with MSCC tool:

```
[ file rx.c --> function int run_rx_for_gui(void) ]  
[ file tx.c --> function int handle_tx_15byte_from_gui(char *line) ]
```

4.1.3 PoE manager debug prints

The shared memory contains special string information area, used for debug prints. It is described in section 0.

To use it, a special Dragonite debug version is required. The debug version will print out relevant information to enable proper debug.

How to operate with MSCC tool:

```
[ file rx.c --> function int run_rx_debug_text_for_gui(void) ]  
[ file tx.c --> function int HandelDbgmsg_from_gui(char *line) ]
```

Debug String Print control bytes:

- TX_Debug_MO_Byte is used for transmitting debug control from the Host to the POE Manager. The allowed values of this byte are:
 - 0xC0 – Host Ownership.
 - Host can update the data information.
 - The information is not valid for POE manager.
 - 0x0C – POE Ownership.
 - Host sent a message to the POE manager and the information is valid for the POE Manager. Host is not allowed to change the data as long as the value of this byte is 0x0C. (Size_N field must be > 0)
 - 0xFF – Init value (Set by the Host as part of boot up sequence), will be updated by POE to 0xC0, after POE release from reset.
 - The data structure information is not valid.

*How to test : The Host should set an unsupported value (0xBB). The PoE is expected to change the value to 0xC0 and increment the communication error counter.
The Host should verify the described PoE behavior.*

- RX_Debug_MO_Byte is used for receiving debug information from the POE Manager, by the Host. The allowed values of this byte are:
 - 0xD0 – Host Ownership.
 - Host can Read the debug information.
 - The information is valid and was sent from the POE manager.
 - The Host need to print the information to the debug UART.
 - The POE manager cannot update the information as long as the value of this byte is 0xD0. (Size_N field must be > 0)
 - 0x0D – POE Ownership.
 - Host read the last message and the area is ready for new reply from the POE manager. The host is not allowed to read data as as the byte value is 0x0D.
 - 0xFF – Init value (Set by the Host as part of boot up sequence), will be updated by POE to 0x0D, after POE release from reset.
 - The data structure information is not valid.
- Size_N – defines the number of ASCII characters that exist in the debug message. (Size_N=0 means no debug info. Size > 1536 is ignored).

*How to test : The Host should set an unsupported value (0xAA). The PoE is expected to change the value to 0x0D and increment the communication error counter.
The Host should verify the described PoE behavior.*

Note: During system power up, communication data exchange structure bytes must be configured by the Host to 0xFF, prior to releasing the POE manager from Reset.

4.2 Host Debug Application

4.2.1 General view

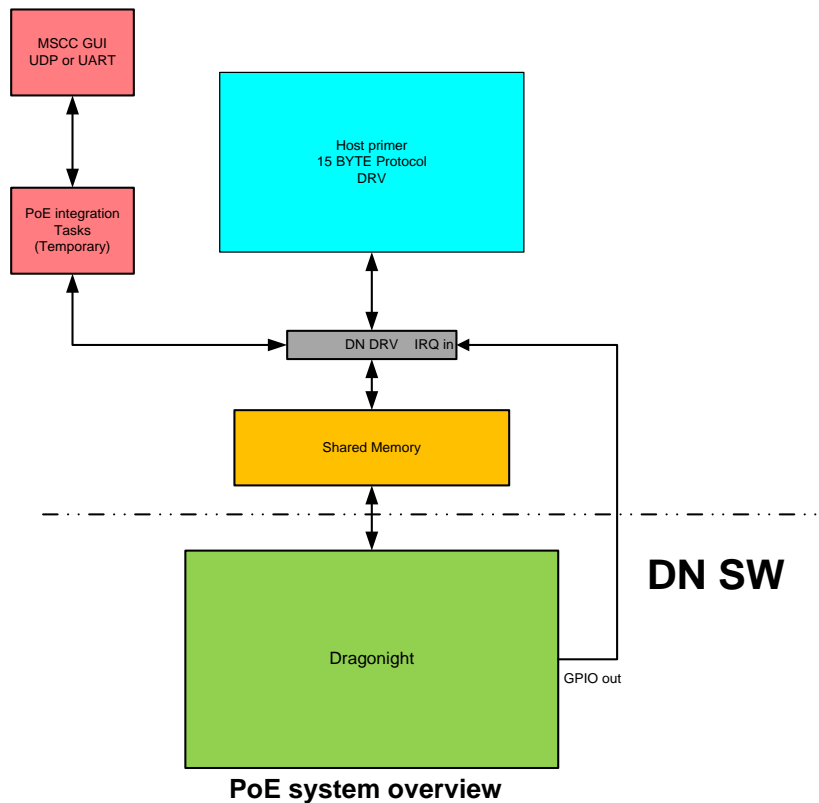
The debug relies on host debug application that will reflect internal data exchange between the Host and the Dragonite, since the Dragonite is not externally accessible.

Debug need to be enabled when necessary on the host side. (Dragonite side debug is constantly enabled).

If JTAG debugger is required there is a need for hardware modifications.
(Relevant documentation need to be supplied by Marvell).

Debug requirements:

1. "text debug" like console.
2. Access to Dragonite FW using secondary TX/RX interface.
3. Access to shared memory.
4. Connect MSCC GUI to support 1 to 3.
5. Print out the Sniffing information from the shared memory. (The sniffing information is the 15 byte information at the primary TX/RX interface).
6. Host should be able to log the information from 5.

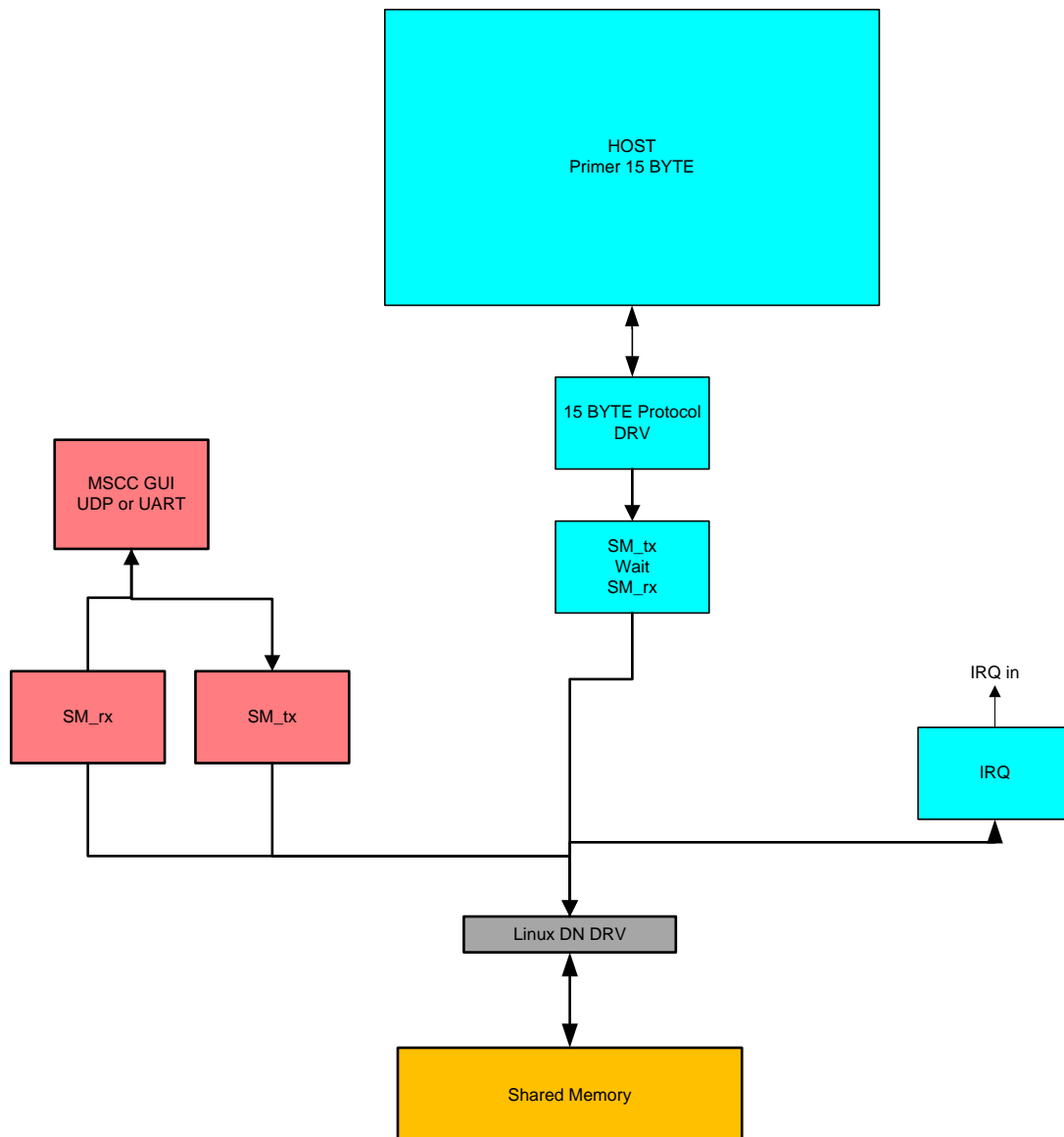


The Host need to run a Thread/Task/process that will communicate with the PC based GUI and the shared memory.

The GUI is text over UART/UDP.

The access to the shared memory must be atomic operation.

Note: Two tasks access to the same resource without atomic operation is not allowed. A read/write operation flowed “lseek” cause errors.



More detailed PoE system overview

4.2.2 The GUI text protocol flow:

- 1) Sending 15 BYTE from PC and back(UART/UDP).
The PC send a text data which need to be rip from the prefix ,convert from text to binary ,send it to the shared memory and vice versa from Host to PC
 - a. PC in sending to the Host
msg15: 02 00 07 3D 4E 4E 4E 4E 4E 4E 4E 4E 03 04 \r\n
 - i. the Host need to remove the "msg15: " .
 - ii. convert the HEX Text to a byte.
 - iii. The Host need to calculate checksum (if fail drop this msg)
 - iv. Send the 15 byte to the shared memory via TX2_MO ..
 - b. the Host sending to the PC (answer)
 - i. if the host get a 15 BYTE MSG on the RX2_MO take it.
 - ii. Start msg with "msg15: "
 - iii. Convert the bytes to a Text mgs (msg15: 03 00 00 3D 4E 4E 4E 4E 4E 4E 4E 03 04\r\n).
 - iv. Add CRLF.
 - v. Send it to the PC.
- 2) Sending a Debug text message from PC and Back:
the Idea is send/receive regular text message but add to it a prefix "dbgmssc:
"
 - a. dbgmssc: some text message \r\n
 - b. when Getting a Debug text the host need to rip the "dbgmssc: " and send it to the shared memory Debug MSG.
 - c. when there is a Debug MSG at the shared memory – Grab the text ,add the prefix "dbgmssc: " , add suffix CRLF and send to the PC.
- 3) Access to specific shared memory starting address location in the shared memory, from external PC.
The necessity of having access from the PC to the shared memory space is to debug the shared memory information flow between the Host and the Dragonite.
The debug is based on accessing to a starting address and stream of data from that starting point. The information is represented in ASCII characters, delimited by space. A parameter between 2 spaces represents a 32bit value.
 - a. Write example:
dbgmssc_sim: W 3A8 0B1E 03\r\n
Explanation:
 - i. Operation – W (Write) + Space delimiter

- ii. Address – 3A8 = 0x000003A8
- iii. Data in 32bit – 0B1E = 0x00000B1E (Will be located in address 0x000003A8)
- iv. Data in 32bit – 03 = 0x00000003 (Will be located in address 0x000003AC)
- v. `\r\n` = End of message

the host need acknowledge the operation by returning ack message to the PC

```
dbgmscc_sim: write addr-3a8: 00000b1e 00000003
.. \r\n
```

b. Read example:

```
dbgmscc_sim: R 3A8 0C\r\n
```

Explanation:

- i. Operation – R (Read) + Space delimiter
- ii. Address – 3A8 = 0x000003A8
- iii. Data length value – 0C = How many 32bit values to read, in this case 12 values.
- iv. `\r\n` = End of message

The host need to return to the PC the read data from the shared memory according to the below foremat:

```
dbgmscc_sim: read addr-3A8: ffffffff ffffffff
ffffffff ffffffff ffffffff 000000a0 00000000
00000000 00000000 00000000 00000000
00000000\r\n
```

4) Sniffer:

The sniffer usage is to debug the 15byte protocol messages exchange between the host and the Dragonite.

The host should read the data from sniffer fifo and send it / save it with the following format. Each extracted value from the FIFO should be tagged using free running 32bit counter that will be incremented after every read. This will enable sorting 15 byte messages information after sending it through UDP to a PC.

In 15Byte case:

sniffmscc: fifo data CNT: counter value data: 8 bit
data TS: time stamp Overflow? NO/YES \r\n

example :

sniffmscc: 0x00012003 CNT:0x0235f004 data: 0x03
TS: 0x0120 Overflow? NO \r\n

In descriptor case:

sniffmscc: descriptor value CNT:counter value HDR:
header value TS: time stamp value DATA: data value
\r\n

example :

sniffmscc: 0x800030d8 CNT:0x0235f002 HDR:
0x00010060 TS: 0x047568af DATA: 0x00000000
0x00000020 \r\n

5 Code Delivery

5.1 Code Delivery

The POE Application code file will be delivered in Binary format.

The POE application code will be downloaded by the Host and the Host is responsible to check download success.

In addition, prior to the download, the Host must check that the file is valid and suitable for the system.

To perform file verification, special pointers for CRC, sys_Type, Product information and creation date&time where added to the Binary code, at fix locations from the file end.

| Information Type | Information Location [Bytes] | Information Structure |
|------------------|------------------------------|--|
| CRC | File_End - 4 | CRC value |
| System Type | File_End - 8 | Pointer to : customer_info_data_struct *pcustomer_info_data_struct; |

How to test : The CRC functionality can be tested by applying 2 different valid DN code images and 2 different Invalid or damaged DN codes.

```
typedef struct
{
    ms_u32 version_number; - struct version
    ms_u32 struct_size; - struct sizeof
    //All vars below are pointers
    ms_s8 *sign_date; - date of generate the version null terminate
    ms_s8 *sign_time; - time of generate the version null terminate
    ms_s8 *chr_array_ptr; - TBD string null terminate
    ms_u16 *software_pn; - this value is the same as in the 15 BYTE communication protocol doc
    ms_u8 *software_build; - this value is the same as in the 15 BYTE communication protocol doc
    ms_u8 *product_number; - this value is the same as in the 15 BYTE communication protocol doc
    ms_u16 *kernel_version; - this value is the same as in the 15 BYTE communication protocol doc
    ms_u8 *salad_param_code; - this value is the same as in the 15 BYTE communication protocol doc
}customer_info_data_struct
```

5.2 CRC Calculation

```
ms_u8 CheckKereneValid(void)
{
    KernelInfo_t *pKernelInfo;
    ms_u32      crcval=0;
    ms_u32 CRCsize;

    const ms_u32 zero=0;

    crcval = slow_crc32(crcval, (ms_u8 *)0, 65533);
    crcval = slow_crc32(crcval, (unsigned char *)&zero, sizeof(zero));

    if (crcval!=*(ms_u32 *) (0x10000-0x4))
```

```

        return FALSE;
    return TRUE;
}

#define uint32_t unsigned long
#define POLY 0x4C11DB7

unsigned long slow_crc32(unsigned long sum, unsigned char *p, unsigned
long len);

unsigned long slow_crc32(unsigned long sum, unsigned char *p, unsigned
long len)
{
    while (len--)
    {
        int i;
        unsigned char byte = *p++;

        for (i = 0; i < 8; ++i)
        {
            unsigned long osum = sum;
            sum <<= 1;
            if (byte & 0x80)
                sum |= 1;
            if (osum & 0x80000000)
                sum ^= POLY;
            byte <<= 1;
        }
    }
    return sum;
}

```

5.3 *mscc tool*

The msc tool implement this document specially section 4.

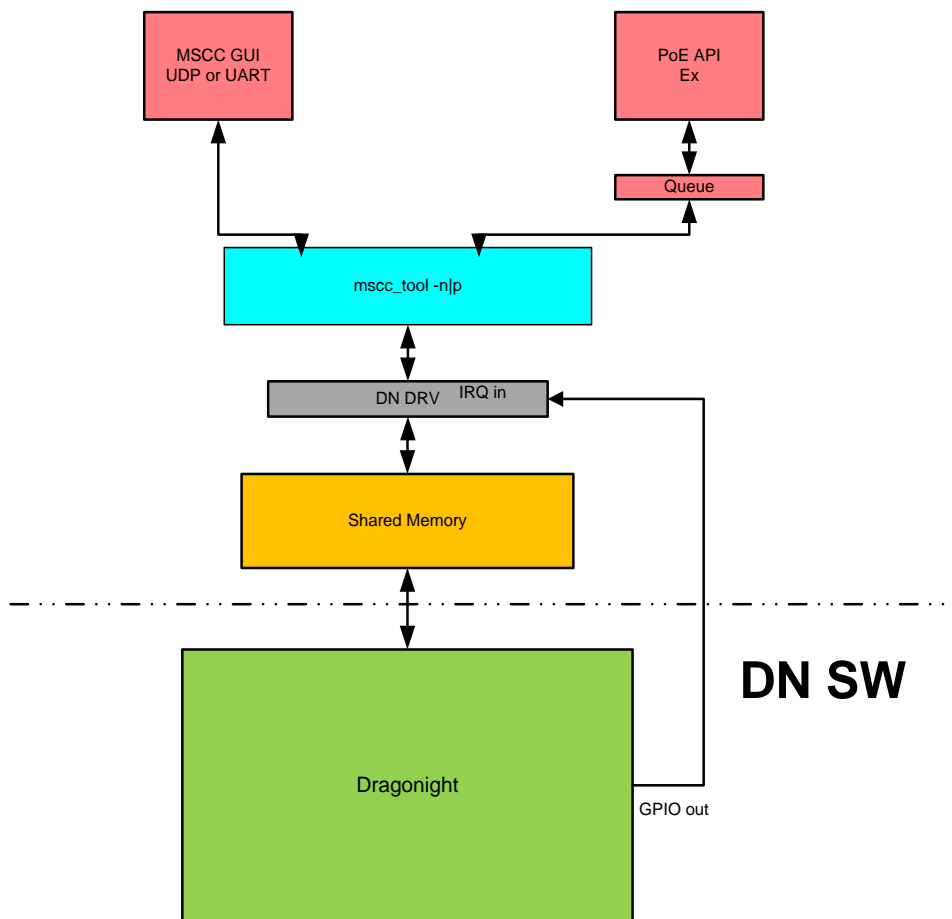
The tool has to three interfaces

- 1) communicate with DN with or without Linux kernel driver.
- 2) with the GUI – over UDP (OOB port not from packet professor) or with UART
- 3) with Queue - using the API example code

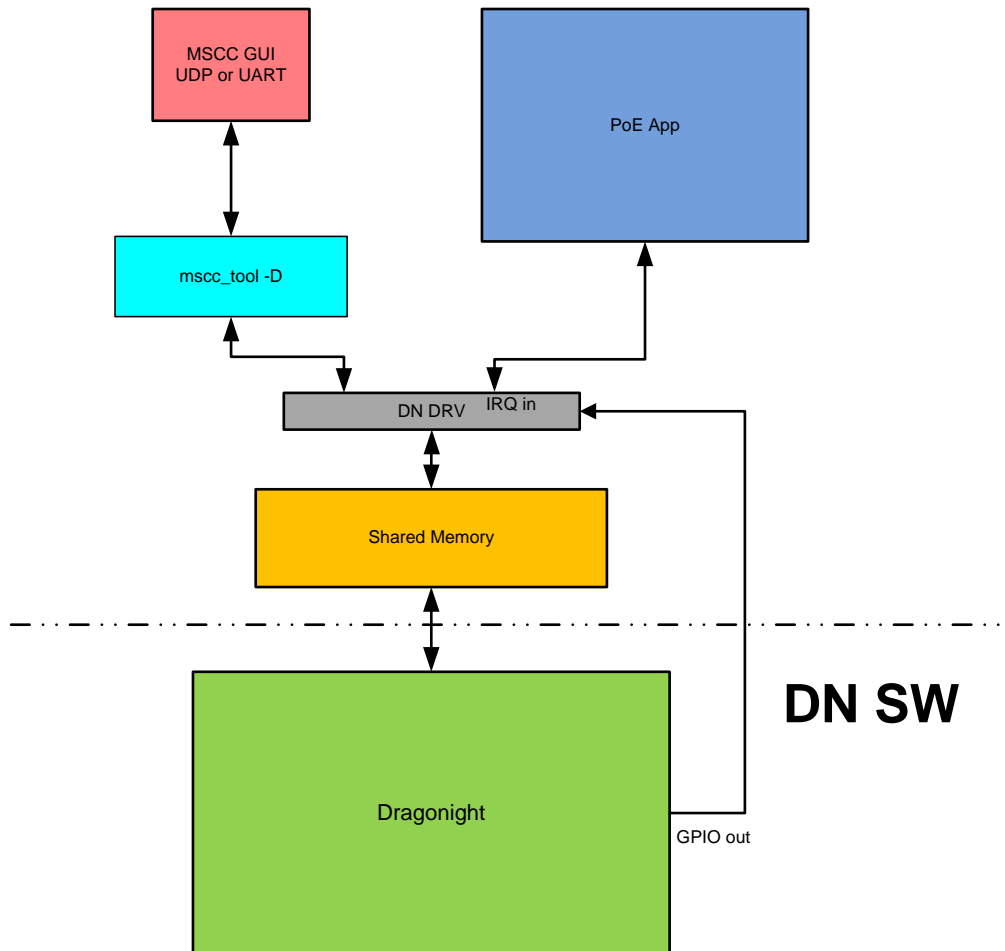
mscc tool has several arguments options

| Option | Description |
|----------------------------|---|
| -t, --reset | init and put dragonite into reset |
| -u, --unreset | un-reset dragonite |
| -s, --sendirq | send irq to dragonite |
| -o, --irqpoll | infinite poll for dragonite irq |
| -r, --read f_off m_size | read m_size from dragonite(ITCM or DTCM) from f_off offset |

| | |
|----------------|--|
| -w, --write | write 32bit to DN memory - address , data , ITCM DTCM (0/1) write f_size bytes from f_name to dragonite(ITCM or DTCM) in f_off offset |
| -d, --info | reads 3 longs from customer info structure. The pointer to this structure is in ITCM\ |
| -c, --crc | execute crc check on the dragonite ITCM code |
| -p, --run | run the bridge program over stty |
| -n, --net | run the bridge program over UDP |
| -D, --DBUGE | run the bridge program over UDP .checking only sniffer and secondary interface, other msc tool check bypassed |
| -e, --enable | enable DN on NO_KM mode |
| -f, --download | download a FW to the DN |
| -v | print version |
| -? | print help |



mscc_tool -p|n



mscc_tool -D

5.4 AC3x - Aldrin

In the Aldrin the `mscc_tool` need more information to work.

The information is regarding the PCI

need to run a script `p-pci.sh`

the script generate file name `mscc_tool_conf` with the parameters for the `mscc_tool`

```
/root # cat p-pci.sh
#!/bin/sh -x
grep -i -q msys /proc/cpuinfo
if [ $? = 0 ]; then
    devId=ff:ff:ff
else
```



```
devld=`lspci|sed -n 's/^(.*) Class 0200: 11ab:.*\1/p'|head -n 1`  
if [ "x${devld}x" = "xx" ]; then  
    devld=ff:ff:ff  
fi  
fi  
  
echo pex: $devld  
echo pex: $devld > msc_tool_conf
```

Appendix – A – Shared Memory Addresses @ POE DTCM

| 32 bit width array | | Address Start DTCM BASE + offset | Host R/W Direction |
|---------------------------------|-----------------------------------|--|-----------------------|
| CPU Type Host | Protocol Version Register Host | 0x0000 | R/W |
| CPU Type PoE | Protocol Version Register PoE | 0x0004 | RO |
| POE_System_Type | | 0x0008 | RO |
| POE_App_Alive_Counter | | 0x000C | RO |
| Host_Input_From_POE_General_Reg | | 0x0010 | RO |
| Self_RST_Cause | Self_RST_Event_Count | 0x0014 | RO |
| Comm_Err_Cntr | | 0x0018 | RO |
| Host Input_From_PoE_IRQ_Reg | | 0x001C | RO |
| Host_RST_Cause_DN_Reflection | Reserved | 0x0020 | RO |
| Control | N.A | 0x0024 | RO |
| Host_Output_To_POE | | 0x0030 | R/W |
| Host_RST_Cause | Host_RST_Event_Count | 0x0034 | R/W |
| Interrupt Mux Select | | 0x0038 | R/W |
| | | | |
| 0x000000 | TX_MO_Byte | 0x0050 | R/W |
| 0x000000 | DATA Byte0 | 0x0054 | R/W |
| 0x000000 | DATA Byte1 | 0x0058 | R/W |
| 0x000000 | DATA Byte2 | 0x005c | R/W |
| 0x000000 | DATA Byte3 | 0x0060 | R/W |
| 0x000000 | DATA Byte4 | 0x0064 | R/W |
| 0x000000 | DATA Byte5 | 0x0068 | R/W |
| 0x000000 | DATA Byte6 | 0x006C | R/W |
| 0x000000 | DATA Byte7 | 0x0070 | R/W |
| 0x000000 | DATA Byte8 | 0x0074 | R/W |
| 0x000000 | DATA Byte9 | 0x0078 | R/W |
| 0x000000 | DATA Byte10 | 0x007c | R/W |
| 0x000000 | DATA Byte11 | 0x0080 | R/W |
| 0x000000 | DATA Byte12 | 0x0084 | R/W |

| | | | |
|---------------------|-------------|---------------|-----|
| 0x000000 | DATA Byte13 | 0x0088 | R/W |
| 0x000000 | DATA Byte14 | 0x008C | R/W |
| DN Frequency | | 0x0090 | R/W |
| must be 0xffff_ffff | | 0x0094-0x009C | |
| | | | |

Note: The structure order is "Little Endian".

| 32 bit width array | | Address Start DTCM BASE + offset | Host R/W Direction |
|-------------------------|------------------|-------------------------------------|--------------------------|
| 0x000000 | RX_MO_Byte | 0x0100 | R/W |
| 0x000000 | DATA Byte0 | 0x0104 | RO |
| 0x000000 | DATA Byte1 | 0x0108 | RO |
| 0x000000 | DATA Byte2 | 0x010C | RO |
| 0x000000 | DATA Byte3 | 0x0110 | RO |
| 0x000000 | DATA Byte4 | 0x0114 | RO |
| 0x000000 | DATA Byte5 | 0x0118 | RO |
| 0x000000 | DATA Byte6 | 0x011C | RO |
| 0x000000 | DATA Byte7 | 0x0120 | RO |
| 0x000000 | DATA Byte8 | 0x0124 | RO |
| 0x000000 | DATA Byte9 | 0x0128 | RO |
| 0x000000 | DATA Byte10 | 0x012C | RO |
| 0x000000 | DATA Byte11 | 0x0130 | RO |
| 0x000000 | DATA Byte12 | 0x0134 | RO |
| 0x000000 | DATA Byte13 | 0x0138 | RO |
| 0x000000 | DATA Byte14 | 0x013C | RO |
| DN Frequency Report | | 0x0140 | RO |
| FW will put 0xffff_ffff | | 0x0144-0x014C | RO |
| Size_N | TX_Debug_MO_Byte | 0x0150 | R/W |
| 0x000000 | Character #1 | 0x0154 | R/W |
| size 0x4b0 [Byte] | | | R/W |
| | | | |
| | | | R/W |
| 0x000000 | Character #N | | R/W |

| Size_N | RX_Debug_MO_Byte | 0x650 | R/W |
|--------------------|------------------|-------------------------------------|--------------------------|
| 0x000000 | Character #1 | | RO |
| size 0x4b0 [Byte] | | | RO |
| | | | |
| | | | RO |
| 0x000000 | Character #N | | RO |
| 32 bit width array | | Address Start DTCM BASE + offset | Host R/W Direction |
| | | | |
| 0x000000 | TX2_MO_Byte | 0x0b04 | R/W |
| 0x000000 | DATA Byte0 | 0x0b08 | R/W |
| 0x000000 | DATA Byte1 | 0x0b0c | R/W |
| 0x000000 | DATA Byte2 | 0x0b10 | R/W |
| 0x000000 | DATA Byte3 | 0x0b14 | R/W |
| 0x000000 | DATA Byte4 | 0x0b18 | R/W |
| 0x000000 | DATA Byte5 | 0x0b1C | R/W |
| 0x000000 | DATA Byte6 | 0x0b20 | R/W |
| 0x000000 | DATA Byte7 | 0x0b24 | R/W |
| 0x000000 | DATA Byte8 | 0x0b28 | R/W |
| 0x000000 | DATA Byte9 | 0x0b2c | R/W |
| 0x000000 | DATA Byte10 | 0x0b30 | R/W |
| 0x000000 | DATA Byte11 | 0x0b34 | R/W |
| 0x000000 | DATA Byte12 | 0x0b38 | R/W |
| 0x000000 | DATA Byte13 | 0x0b3C | R/W |
| 0x000000 | DATA Byte14 | 0x0b40 | R/W |

Note: The structure order is "Little Endean".

| 32 bit width array | | Address Start DTCM BASE + offset | Host R/W Direction |
|--------------------|-------------|-------------------------------------|--------------------------|
| 0x000000 | RX2_MO_Byte | 0x0b44 | R/W |
| 0x000000 | DATA Byte0 | 0x0b48 | RO |
| 0x000000 | DATA Byte1 | 0x0b4C | RO |
| 0x000000 | DATA Byte2 | 0x0b50 | RO |
| 0x000000 | DATA Byte3 | 0x0b54 | RO |
| 0x000000 | DATA Byte4 | 0x0b58 | RO |
| 0x000000 | DATA Byte5 | 0x0b5C | RO |
| 0x000000 | DATA Byte6 | 0x0b60 | RO |
| 0x000000 | DATA Byte7 | 0x0b64 | RO |
| 0x000000 | DATA Byte8 | 0x0b68 | RO |
| 0x000000 | DATA Byte9 | 0x0b6C | RO |
| 0x000000 | DATA Byte10 | 0x0b70 | RO |
| 0x000000 | DATA Byte11 | 0x0b74 | RO |
| 0x000000 | DATA Byte12 | 0x0b78 | RO |
| 0x000000 | DATA Byte13 | 0x0b7C | RO |
| 0x000000 | DATA Byte14 | 0x0b80 | RO |

Note: The structure order is "Little Endian".

| 32 bit width array | | | Address Start DTCM BASE + offset | Host R/W Direction |
|--------------------|-------------------------|------------|-------------------------------------|--------------------------|
| version | Sniffer Size 16 bit | | 0x0B84 | Ro |
| | Write Counter 16 bit | | 0x0B88 | RO |
| | Read Counter 16 bit | | 0x0B8C | R/W |
| Control | Time stamp (16 bit) | DATA 8 bit | 0x0B90 | RO |
| Control | Time stamp (16 bit) | DATA 8 bit | | RO |
| Control | Descriptor | | | RO |
| Control | Time stamp (16 bit) | DATA | | RO |
| Control | Descriptor | | | RO |
| Control | Time stamp (16 bit) | DATA | | RO |
| Control | Time stamp (16 bit) | DATA | | RO |
| Control | Time stamp (16 bit) | DATA | | RO |
| Control | Time stamp (16 bit) | DATA | | RO |
| Control | Time stamp (16 bit) | DATA | | RO |
| Control | Time stamp (16 bit) | DATA | | RO |
| Control | Time stamp (16 bit) | DATA | | RO |
| | | | | |

Note: The structure order is "Little Endean".

Appendix – B – POE Reset influence on protocol flow

| Normal Operation | |
|---|--|
| Expected Init conditions before new message from Host: TX_MO_Byte = Host Ownership RX_MO_Byte = POE Ownership | |
| Host | POE |
| Sample RX_MO_Byte: | Sample TX_MO_Byte |
| If (Host Ownership) | |
| { Host Reads RX Shared Memory OLD message, Set RX_MO_Byte to POE Ownership } | Sample TX_MO_Byte |
| If (POE Ownership) {continue following steps} | |
| Sample TX_MO_Byte | |
| Host copy data to TX Shared Memory | Sample TX_MO_Byte |
| TX_MO_Byte = POE Ownership (By Host) | |
| | Sample TX_MO_Byte |
| Sample RX_MO_Byte | POE Manager copy TX Shared Memory to POE TX Fifo |
| | Set TX_MO_Byte to Host Ownership |
| | POE Manager prepares message reply |
| | POE Message is copied into the POE TX fifo. |
| Sample RX_MO_Byte | Sample RX_MO_Byte |
| | POE RX fifo copy to RX Shared Memory |
| | Set RX_MO_Byte to Host Ownership |
| Sample RX_MO_Byte | POE Ready for new message from Host |
| Host Reads RX Shared Memory message | |
| Set RX_MO_Byte to POE Ownership | |
| Host Ready for new message | |

| POE Reset during Host write | |
|--|--|
| <u>Expected Init conditions before new message from Host:</u> TX_MO_Byte = Host Ownership RX_MO_Byte = POE Ownership | |
| Host | POE |
| Sample RX_MO_Byte: | Sample TX_MO_Byte |
| If (Host Ownership) | |
| { Host Reads RX Shared Memory OLD message, Set RX_MO_Byte to POE Ownership } | Sample TX_MO_Byte |
| If (POE Ownership) {continue following steps} | |
| Sample TX_MO_Byte | |
| Host copy data to TX Shared Memory | Sample TX_MO_Byte |
| TX_MO_Byte = POE Ownership (By Host) | POE Reset (Reset Cause != POR) |
| | Sample TX_MO_Byte |
| Sample RX_MO_Byte | POE Manager Ignores TX Shared Memory |
| | Set TX_MO_Byte to Host Ownership |
| | POE Manager prepares system status message reply |
| | POE Message is copied into the POE RX fifo. |
| Sample RX_MO_Byte | Sample RX_MO_Byte |
| | POE RX fifo copy to RX Shared Memory |
| | Set RX_MO_Byte to Host Ownership |
| Sample RX_MO_Byte | POE Ready for new message from Host |
| Host Reads RX Shared Memory message | |
| Set RX_MO_Byte to POE Ownership | |
| Host Ready for new message | |

| POE Reset during reply preparations | |
|--|--|
| <u>Expected Init conditions before new message from Host:</u> TX_MO_Byte = Host Ownership RX_MO_Byte = POE Ownership | |
| Host | POE |
| Sample RX_MO_Byte: | Sample TX_MO_Byte |
| If (Host Ownership) | |
| { Host Reads RX Shared Memory OLD message, Set RX_MO_Byte to POE Ownership } | Sample TX_MO_Byte |
| If (POE Ownership) {continue following steps} | |
| Sample TX_MO_Byte | |
| Host copy data to TX Shared Memory | Sample TX_MO_Byte |
| TX_MO_Byte = POE Ownership (By Host) | |
| | Sample TX_MO_Byte |
| Sample RX_MO_Byte | POE Manager copy TX Shared Memory to POE TX Fifo |
| | Set TX_MO_Byte to Host Ownership |
| | POE Manager prepares message reply |
| | POE Reset (Reset Cause != POR) |
| Sample RX_MO_Byte | Sample TX_MO_Byte |
| | POE Manager prepares system status message reply |
| | POE Message is copied into the POE RX fifo. |
| Sample RX_MO_Byte | Sample RX_MO_Byte |
| Host Reads RX Shared Memory message | POE RX fifo copy to RX Shared Memory |
| Set RX_MO_Byte to POE Ownership | Set RX_MO_Byte to Host Ownership |
| Host Ready for new message | POE Ready for new message from Host |

| POE Reset after reply preparations #1 | |
|--|--|
| <u>Expected Init conditions before new message from Host:</u> TX_MO_Byte = Host Ownership RX_MO_Byte = POE Ownership | |
| Host | POE |
| Sample RX_MO_Byte: | Sample TX_MO_Byte |
| If (Host Ownership) | |
| { Host Reads RX Shared Memory OLD message, Set RX_MO_Byte to POE Ownership } | Sample TX_MO_Byte |
| If (POE Ownership) {continue following steps} | |
| Sample TX_MO_Byte | |
| Host copy data to TX Shared Memory | Sample TX_MO_Byte |
| TX_MO_Byte = POE Ownership (By Host) | |
| | Sample TX_MO_Byte |
| Sample RX_MO_Byte | POE Manager copy TX Shared Memory to POE TX Fifo |
| | Set TX_MO_Byte to Host Ownership |
| | POE Manager prepares message reply |
| | POE Message is copied into the POE RX fifo. |
| Sample RX_MO_Byte | Sample RX_MO_Byte |
| | POE Reset (Reset Cause != POR) |
| | Sample TX_MO_Byte |
| Sample RX_MO_Byte | POE Manager prepares system status message reply |
| | POE Message is copied into the POE RX fifo. |
| | Sample RX_MO_Byte |
| Sample RX_MO_Byte | POE RX fifo copy to RX Shared Memory |
| | Set RX_MO_Byte to Host Ownership |
| | POE Ready for new message from Host |
| Sample RX_MO_Byte | |
| Host Reads RX Shared Memory message | |
| Set RX_MO_Byte to POE Ownership | |
| Host Ready for new message | |

| | |
|--|--|
| POE Reset after reply preparations #2 | |
| <u>Expected Init conditions before new message from Host:</u> TX_MO_Byte = Host Ownership RX_MO_Byte = POE Ownership | |
| Host | POE |
| Sample RX_MO_Byte: | Sample TX_MO_Byte |
| If (Host Ownership) | |
| { Host Reads RX Shared Memory OLD message, Set RX_MO_Byte to POE Ownership } | Sample TX_MO_Byte |
| If (POE Ownership) {continue following steps} | |
| Sample TX_MO_Byte | |
| Host copy data to TX Shared Memory | Sample TX_MO_Byte |
| TX_MO_Byte = POE Ownership (By Host) | |
| | Sample TX_MO_Byte |
| Sample RX_MO_Byte | POE Manager copy TX Shared Memory to POE TX Fifo |
| | Set TX_MO_Byte to Host Ownership |
| | POE Manager prepares message reply |
| | POE Message is copied into the POE TX fifo. |
| Sample RX_MO_Byte | Sample RX_MO_Byte |
| | POE RX fifo copy to RX Shared Memory |
| | POE Reset (Reset Cause != POR) |
| Sample RX_MO_Byte | Sample TX_MO_Byte |
| | POE Manager prepares system status message reply |
| | POE Message is copied into the POE RX fifo. |
| Sample RX_MO_Byte | Sample RX_MO_Byte |
| | POE RX fifo copy to RX Shared Memory |
| | Set RX_MO_Byte to Host Ownership |
| Sample RX_MO_Byte | POE Ready for new message from Host |
| Host Reads RX Shared Memory message | |
| Set RX_MO_Byte to POE Ownership | |
| Host Ready for new message | |

| | |
|---|--|
| POE Reset after reply preparations #3 <u>Expected Init conditions before new message</u> from Host: TX_MO_Byte = Host Ownership RX_MO_Byte = POE Ownership | |
| Host | POE |
| | |
| Sample RX_MO_Byte: | Sample TX_MO_Byte |
| If (Host Ownership) | |
| { Host Reads RX Shared Memory OLD message, Set RX_MO_Byte to POE Ownership } | Sample TX_MO_Byte |
| If (POE Ownership) {continue following steps} | |
| Sample TX_MO_Byte | |
| Host copy data to TX Shared Memory | Sample TX_MO_Byte |
| TX_MO_Byte = POE Ownership (By Host) | |
| | Sample TX_MO_Byte |
| Sample RX_MO_Byte | POE Manager copy TX Shared Memory to POE TX Fifo |
| | Set TX_MO_Byte to Host Ownership |
| | POE Manager prepares message reply |
| | POE Message is copied into the POE RX fifo. |
| Sample RX_MO_Byte | Sample RX_MO_Byte |
| | POE RX fifo copy to RX Shared Memory |
| | Set RX_MO_Byte to Host Ownership |
| Sample RX_MO_Byte | POE Reset (Reset Cause != POR) |
| Host activity is interrupted | Sample TX_MO_Byte |
| | POE Manager prepares system status message reply |
| | POE Message is copied into the POE RX fifo. |
| | Sample RX_MO_Byte |
| | |
| | |
| Host Reads RX Shared Memory message | Sample RX_MO_Byte |
| Set RX_MO_Byte to POE Ownership | |
| Host Ready for new message | |
| | Sample RX_MO_Byte |
| | POE RX fifo copy to RX Shared Memory |
| | Set RX_MO_Byte to Host Ownership |
| Shared memory contains Old message that was not read | POE Ready for new message from Host |

| | |
|--|--|
| POE Reset during Host read | |
| <u>Expected Init conditions before new message from Host:</u> TX_MO_Byte = Host Ownership RX_MO_Byte = POE Ownership | |
| Host | POE |
| | |
| Sample RX_MO_Byte: | Sample TX_MO_Byte |
| If (Host Ownership) | |
| { Host Reads RX Shared Memory OLD message, Set RX_MO_Byte to POE Ownership } | Sample TX_MO_Byte |
| If (POE Ownership) {continue following steps} | |
| Sample TX_MO_Byte | |
| Host copy data to TX Shared Memory | Sample TX_MO_Byte |
| TX_MO_Byte = POE Ownership (By Host) | |
| | Sample TX_MO_Byte |
| Sample RX_MO_Byte | POE Manager copy TX Shared Memory to POE TX Fifo |
| | Set TX_MO_Byte to Host Ownership |
| | POE Manager prepares message reply |
| | POE Message is copied into the POE RX fifo. |
| Sample RX_MO_Byte | Sample RX_MO_Byte |
| | POE RX fifo copy to RX Shared Memory |
| | Set RX_MO_Byte to Host Ownership |
| Sample RX_MO_Byte | POE Ready for new message from Host |
| Host Reads RX Shared Memory message | POE Reset (Reset Cause != POR) |
| Set RX_MO_Byte to POE Ownership | Sample TX_MO_Byte |
| Host Ready for new message | POE Manager prepares system status message reply |
| | POE Message is copied into the POE RX fifo. |
| | Sample RX_MO_Byte |
| | POE RX fifo copy to RX Shared Memory |
| | Set RX_MO_Byte to Host Ownership |
| Shared memory contains additional message that was not read | POE Ready for new message from Host |

Change Record

| Rev | Date | Responsible Person | Description of Change |
|------------|------------|--------------------|--|
| Draft 1.0 | 30/06/2013 | Alon F | Initial Release. |
| Draft 1.1 | 23/09/2013 | Alon F | Add Protocol version, Reset cause, I/O registers, POE Watch dog counter, Protocol control Reset scenarios, Add full host communication flow, update control bytes description. |
| Draft 1.2 | 27/01/2014 | Alon F | <ol style="list-style-type: none"> 1. Protocol description update Based on remarks from readers. 2. No POE Boot Code. Boot Up is Host responsibility. 3. POE_Boot_WD_Count was removed. 4. Keep alive explanation was updated. 5. Boot up flow update. 6. Appendix A table update. |
| Draft 1.3 | 10/07/2014 | Alon F | <ol style="list-style-type: none"> 1. Add Comm_Err_Cntr field in the shared memory protocol. 2. Add CRC algorithm. 3. Add pointers for version information. |
| Draft 1.4 | 10/12/2014 | Roni B. | <ol style="list-style-type: none"> 1. fix pointer version 2. add a testing method for the protocol |
| Draft 1.5 | 10/12/2014 | Roni B. | <ol style="list-style-type: none"> 1. fix pointer version |
| Draft 1.6 | 25/02/2015 | Roni B | <ol style="list-style-type: none"> 1. add IRQ 2. change IRQ to MSG ready |
| Draft 1.61 | 26/02/2015 | Alon F | Headre and footer where added. |
| Draft 1.7 | 8/6/2015 | Roni B | Adding sniffer Add secand interface of 15 Byte |
| Draft 1.8 | 8/7/2015 | Roni B | Adding internal WD timer stamp Update Self_RST_Cause |
| Draft 1.85 | 10/1/2016 | Roni B | Add Mask on the shared memory |
| Rel 2.00 | 17/02/2016 | Alon.F | <ol style="list-style-type: none"> 1. Add Debug. 2. Add host interrupt control. 3. Update descriptions with how to test explanation. 4. Increment protocol revision to 2. |

| | | | |
|----------|------------------------|---------------------|---|
| Rel 2.01 | 29/05/2016 | Roni B. | Add descriptor to the sniffer |
| Rel 2.02 | 6/6/2016 | Roni B | Fix in following subjects: 1. RX_MO_Byte – value 2. RX_Debug_MO_Byte – value 3. dbgmscc_sim – upgrade explanation 4. sniffer – adding host counter |
| Rel 3.00 | 12/12/2016 | Roni B | 1. Add descriptor for R feature. 2. Add 4 bit enter R mode request. 3. Add 4 bit enter R done reply. 4. Power up flowchart update. 5. DB structure section 2.13 was added. 6. Protocol version was incremented to 3. |
| Rel 3.01 | 22/1/2017 | Roni B | 1. Update General purpose Read / Write registers Bits 4-7 |
| Rel 3.02 | 6/2/2017 | Roni B | 1. Add enter R mode flow |
| Rel 3.03 | 15/3/2017 | Roni B | 1. Add mscc_tool |
| Rel 3.04 | 22/3/2017 | Roni B. | 1. Add Aldrin script |
| Rel 4.00 | 2/4/2017 14/08/2017 | Roni B Avi, Alon | 1. Add AC3x frequency 2. Description fixes |