PM7350

S/UNI™-DUPLEX

DSLAM

REFERENCE DESIGN

LINE CARD SOFTWARE

RELEASED

ISSUE 2: NOVEMBER 2000

CONTENTS

1 INTRODUCTION

1.1 Scope

This document introduces the reader to the software components of the DSLAM Line card. It consists of two main sections: the Users' Guide and the Developers' Guide. The Users' Guide discusses setting up and running the software on the DSLAM Line card. The Developers' Guide provides details of the software to enable modification and further development.

1.2 Target Audience

This document has been prepared for all users of PMC-Sierra reference design boards that need to understand the software/firmware running on the DSLAM line cards. To learn more about the Line Card hardware, please refer to "DSLAM Reference Design: Line Card [4]". For an overview of the DSLAM System, please refer to "DSLAM Reference Design: System Design [5]".

1.3 Objectives

- To provide an overview conceptual understanding of the DSLAM line card software components.
- To document the setup required to run the software.
- To document the user interface of the software.
- To document the software design thereby allowing it to be maintained.
- To direct the user to other task-specific documents.

2 USERS' GUIDE

2.1 Overview

The following files are provided:

- The source code of the firmware. (See Developers' Guide section for complete list and description.)
- compile.bat – batch file for compiling and linking the source code.
- MAIN.OBJ – object file that can be loaded to a BDM (Background Debug Mode) emulator.
- main.s19 – Motorola S-record file that can be burned into a ROM.

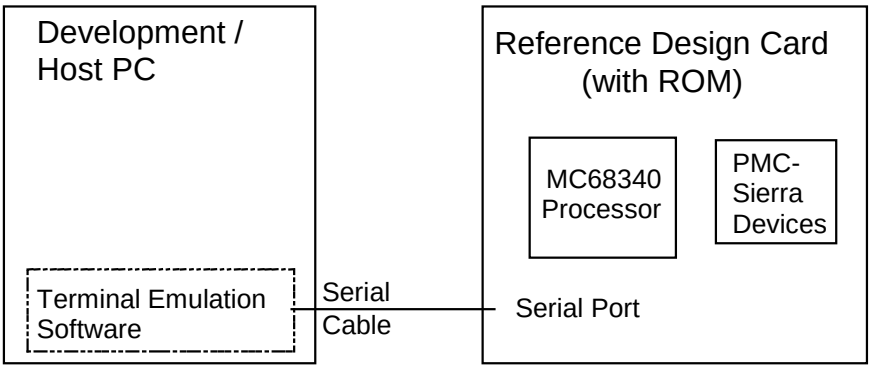
2.2 Setup

The DSLAM Line card software is designed to run under two different setups: ROM (Read Only Memory) and BDM (Background Debug Mode). If modification of the source code is not required, ROM setup allows easy setup with minimal hardware. For further development, the BDM setup is recommended. Both setups are described in detail below.

2.2.1 ROM

If modification of source code is not required, the ROM setup is used. Figure 1 illustrates a typical configuration for running the firmware in ROM setup. In this case the software is burned into the ROM and the user cannot modify the software. Jumper J25 (ROM/BDM SELECT) should be open for ROM setup.

Figure 1: Typical User Configuration



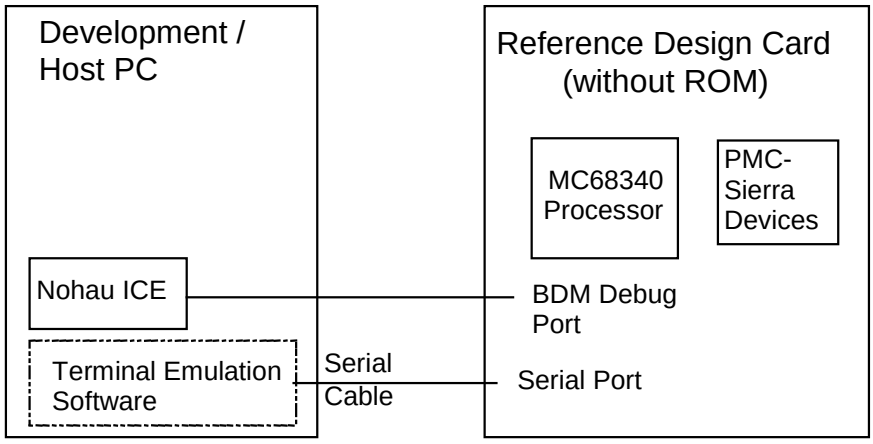
The Motorola S-record file provided, “main.s19” can be programmed to the ROM directly and no compiling is required.

2.2.2 BDM

Figure 2 illustrates an example test-bench required to test and modify the firmware. Jumper J25 (ROM/BDM SELECT) should be shorted for BDM setup. The firmware may need modification to add extra functionality or implement new services required by the user and user specific application software. The development PC should contain the following development tools:

- Nohau In-Circuit Emulator (or other Background Debug Mode emulator)
- Introl-C Compiler/Linker (or other suitable C compiler/linker)

Figure 2: Development Testbed



The next section describes how to compile the source code to load to the Nohau In-Circuit Emulator. Please refer to the Nohau documentation for details on how to load the compiled code.

2.2.3 Compilation

In order to compile the source code, Introl-C compiler/linker must be installed in the system and included in the search path. The batch file “compile.bat” is provided and executing it will produce the appropriate object files and Motorola S-record files for BDM and ROM setup.

The linker command file “c68.ld” provided is targeted for BDM setup. In order to generate a Motorola S-record file for ROM setup after development with BDM setup, the file “c68.ld” must be modified. The line that reads:

```
let __RAMstart = 0xE000; /* address at which to map RAM */
```

needs to be modified to:

```
let __RAMstart = 0x100000; /* address at which to map RAM */
```

2.2.4 Serial Port connection

The user interface for the DSLAM Line Card software is provided by the serial port of the “DSLAM Line Card” reference design card. Typically this port is attached to a terminal via a serial cable. The terminal provides a keyboard interface to enter commands directly, or a terminal emulation program that allows a file of commands to be sent to the serial port. The serial port should be set up to 9600bps, 8 data bits, no parity and 1 stop bit.

The serial cable connects the serial port of the host computer to the RJ-45 connector of the Line Card and has the following pin out:

DB-9 pin #	signal name	RJ-45 pin #
1	Not Connected	
2	RXD	3
3	TXD	6
4	Not Connected	
5	GND	4
6	Not Connected	
7	RTS	8
8	CTS	1
9	Not Connected	

2.3 User Interface

The commands are text characters that can be created via a DOS or Windows editor such as notepad, or entered directly at the terminal. Each command must be delimited by the carriage return and line feed characters. These characters are inserted by typing the return key and are not visible within an editor.

Blank lines are valid and are ignored by the command parser.

Comments may be inserted into a script file by placing the asterisk character in the first column of each line that contains comments.

Fields of a command are delimited by an arbitrary number of blank characters (spaces) or tab characters. The fields of a command must be entered in the order specified and as described below:

- A field with no brackets implies that the text is to be typed exactly as shown.
- <> brackets around a field imply that the textual representation of a number or character string must be entered in the field.
- [ ] brackets around a field, or fields, imply that the field(s) is optional.
- { } braces denotes required parameters.
- | separates parameters that are mutually exclusive

2.3.1 Command Summary

The following list of commands are defined in the following sections:

- act
- clear_stat
- count <device>
- deact
- init [<device>] [<mode>]
- help
- loopback <device> <type> [{RxD1|RxD2} {enable|disable}]

- read <device> <register>
- reset [<device>]
- stat
- write <device> <register> <data>

2.3.2 Detailed Command List

The following pages provide a detailed summary for each of the commands.

2.3.2.1 act

The activate command activates the S/UNI-DUPLEX device for normal operation. The S/UNI-DUPLEX device must be initialized by the “init” command before activation. [Note: Comet devices are automatically activated at initialization.]

Syntax:

act

2.3.2.2 clear_stat

The clear statistic command is issued to clear the accumulated history of interrupts and other statistics of the S/UNI-DUPLEX device

Syntax:

clear_stat

2.3.2.3 count

The display counters command is used to display the accumulated count since the last time this command was issued or, if the command was never issued before, since the device was activated. The counters can be displayed for one specified device at a time.

Syntax:

count <device>

Parameter Description:

<device> selects the hardware device which is represented by typing one of the following character strings without the quotation marks.

Character String	Device
"duplex"	The S/UNI-DUPLEX chip onboard (S/UNI-DUPLEX device must be activated)
"comet1", "comet2"…"comet16"	One of the sixteen COMET chips

Output:

These are the counters that the command will display for the S/UNI-DUPLEX device:

- Serial Link 1 Cells Received Count
- Serial Link 2 Cells Received Count
- Serial Link 1 HCS Errors Count
- Serial Link 2 HCS Errors Count
- Serial Link Cells Transmitted Count

These are the counters that the command will display for a COMET device:

- Framing Bit Error Count
- Out of Frame / Change of Frame / Far End Block
- Bit Error / CRC Error
- LCV Count

Example:

```
* display accumulated counts for the 1st
* and 2nd comet devices
count comet1
count comet2
```

2.3.2.4 deact

The deactivate command deactivates the S/UNI-DUPLEX from normal operation.

Syntax:

deact

2.3.2.5 init

The initialize command configures the devices. [Note: All devices are automatically initialized and activated in default mode upon startup or full reset.]

Syntax:

init [<device>] [<mode>]

Parameter Description:

<device> selects the hardware device(s) which is represented by typing one of the following character strings without the quotation marks. When this option is not specified all devices are selected.

Character String	Device
"all"	All devices onboard
"duplex"	The S/UNI-DUPLEX chip onboard
"comet"	All the sixteen COMET chips onboard
"comet1", "comet2"…"comet16"	One of the sixteen COMET chips

<mode> selects the mode for the COMET devices which is represented by typing one of the following character strings without the quotation marks.. This argument is ignored for the S/UNI-DUPLEX device. The default mode is "T1_S0" if the parameter is omitted.

Character String	Device
"T1_S0"	T1 Short Haul (0-110 ft.)
"E1_120"	E1 120 Ohm

Example:

```
* initialize all devices

init

* initialize the third comet device
```

```
init comet3
```

2.3.2.6 help

Displays help information. The command will list all the commands available on the interface and provide a brief description of what each command does.

Syntax:

help

2.3.2.7 loopback

The loopback command enables/disables loopbacks for S/UNI-DUPLEX and COMET devices. Please refer to the S/UNI-DUPLEX data sheet[2] and COMET data sheet[1] for the functionality of the different kinds of loopback.

Syntax:

loopback <device> <type> [{RxD1|RxD2} {enable|disable}]

Parameter Description:

<device> selects the hardware device which is represented by typing one of the following character strings without the quotation marks.

Character String	Device
“duplex”	The S/UNI-DUPLEX chip onboard
“comet”	All the sixteen COMET devices
“comet1”, “comet2”…”comet16”	One of the sixteen COMET devices

<type> Selects the type of loopback which is represented by typing one of the following character strings without the quotation marks.
Note: Comet loopback modes are mutually exclusive. Setting a COMET to one mode will disable the other loopback modes.

Character String	Device
“line”	Line loopback
“diag”	Diagnostic loopback
“payload”	Payload loopback (valid for COMET devices only)
“disable”	Disable loopback (valid for COMET devices only)

{RxD1|RxD2} Selects the serial link (required for S/UNI-DUPLEX; not valid for COMET).

{enable|disable} Enables or disable loopback for the specified serial link (required for S/UNI-DUPLEX; not valid for COMET).

Example:

```
* enable line loopback on serial link1 of the duplex device

loopback duplex line RxD1 enable

* disable diagnostic loopback on serial link2 of

*    duplex device

loopback duplex diag RxD2 disable

* enable payload loopback on comet3

loopback comet3 payload

* disable all loopback on comet4

loopback comet4 disable
```

2.3.2.8 read

The read command causes the MC68340 to issue a read to the external device register. The 8-bit value read from the external device is displayed.

Syntax:

read <device><register>

Parameter Description:

<device> selects one of the hardware devices which is represented by typing one of the following character strings without the quotation marks.

Character String	Device
"duplex"	The S/UNI-DUPLEX chip onboard
"comet1", "comet2"…"comet16"	One of the sixteen COMET chips
"fpga"	The FPGA onboard
"abs"	No device. <register> provides the absolute address Note: The user must ensure that the address corresponds to RAM/ROM or a device.

<register> is a hex value representing the register offset.

Example:

```
* read register 0x1B of the comet16 device

* and register 0x06 of the duplex device

read comet16 1B

read duplex 6
```

2.3.2.9 reset

The reset command does a software reset on the selected device(s). When all devices are selected (by omitting the device parameter), software on the DSLAM Line Card is also re-started. A device will have to be initialized manually after a device specific reset before further use. [Note: All devices are automatically initialized and activated in default mode upon startup or full reset.]

Syntax:

reset [<device>]

Parameter Description:

<device> selects the hardware device(s) which is represented by typing one of the following character strings without the quotation marks. When this option is not specified all devices are selected.

Character String	Device
“duplex”	The S/UNI-DUPLEX chip onboard
“comet”	All the sixteen COMET chips onboard
“comet1”, “comet2”…”comet16”	One of the sixteen COMET chips

Example:

```
* reset all devices and re-start software

reset

* reset the third comet device

reset comet3
```

2.3.2.10 stat

The “stat” command displays the accumulated statistical counts and interrupts since the last time it was cleared using the “clear_stat” command. The information displayed is related to the S/UNI-DUPLEX device only.

Syntax:

stat

Output:

The following statistics of the S/UNI-DUPLEX device will be displayed:

- Serial Link Transmit Overflow Count Status
- Serial Link Transmit Updated
- Receive Logical Channel FIFO Overflow
- Transmit Logical Channel FIFO Overflow
- Phy Input Cell Transferred
- Invalid SOC Sequence
- Phy Input Parity Error
- Phy Output Error
- Microprocessor Insert FIFO Ready
- Microprocessor Insert FIFO Full
- Extract Cell CRC Error
- Clock Lock Fail

- Receive Serial Channel Out of Delineation (for each Channel)
- Receive Serial Channel In Delineation (for each Channel)
- Receive Serial Channel FIFO Overrun (for each Channel)
- Receive Serial Channel HCS Error (for each Channel)
- Receive Serial Channel Out of Sync (for each Channel)
- Receive Serial Channel In Sync (for each Channel)
- Receive Serial Link Extract FIFO Overflow (for both links)
- Receive Serial Link Loss of Signal (for both links)
- Receive Serial Link Signal Detected (for both links)
- Receive Serial Link Out of Delineation (for both links)
- Receive Serial Link In Delineation (for both links)
- Receive Serial Link Active (for both links)
- Receive Serial Link Not Active (for both links)
- Receive Serial Link Out of Sync (for both links)
- Receive Serial Link In Sync (for both links)
- Receive Serial Link CRC8 Error (for both links)
- Receive Serial Link HCS Error (for both links)
- Receive Serial Link Counter Updated (for both links)
- Receive Serial Link Counter Overflow (for both links)
- Receive Serial Link Valid BOC (for both links)
- Extract FIFO ready
- Number of Interrupts occurred

2.3.2.11 write

The write command causes the MC68340 to write an 8-bit value to a register of an external device.

Syntax:

write <device> <register> <value>

Parameter Description:

<device> selects one of the hardware devices which is represented by typing one of the following character strings without the quotation marks.

Character String	Device
“duplex”	The S/UNI-DUPLEX chip onboard
“comet1”, “comet2”…”comet16”	One of the sixteen COMET chips
“fpga”	The FPGA onboard
“abs”	No device. <register> provides the absolute address Note: The user must ensure that the address corresponds to RAM/ROM or a device.

<register> is the register offset within the device, a hex value.

<value> is the 8-bit hex value written to the external device.

Example:

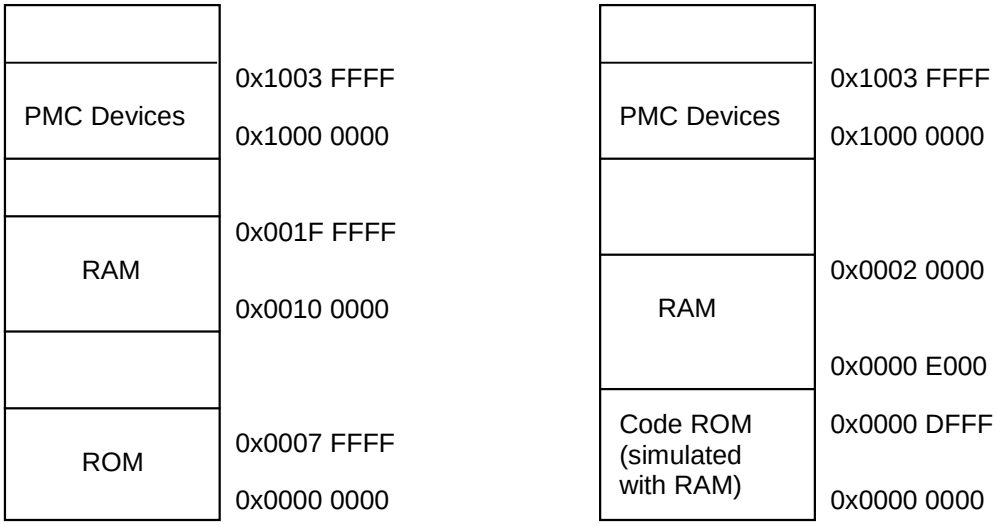
```
* write 0xFF to register 0x08 of the duplex device

* and register 0x01 of the FPGA

write duplex 8 FF

write fpga 1 FF
```

3 DEVELOPERS' GUIDE

3.1 Memory

Figure 3 depicts the memory map for both the ROM Setup and the BDM setup.

Figure 3: Memory Map



The following are the base addresses of each of the devices in the PMC Devices region:

COMET0	0x1000 0000
COMET1	0x1000 1000
COMET2	0x1000 2000
COMET3	0x1000 3000
COMET4	0x1000 4000
COMET5	0x1000 5000
COMET6	0x1000 6000
COMET7	0x1000 7000
COMET8	0x1000 8000
COMET9	0x1000 9000
COMET10	0x1000 A000
COMET11	0x1000 B000
COMET12	0x1000 C000
COMET13	0x1000 D000
COMET14	0x1000 E000

COMET15	0x1000 F000
S/UNI-DUPLEX	0x1001 0000
FPGA	0x1001 8000

If new features are added, the RAM and ROM requirements of the new firmware might increase. If such requirements exceed the ranges provided, the new firmware will fail to execute. Please check the memory map file "main.m68" which is created at linking, to make sure code and data is mapped to the provided ranges.

3.2 Source Code Description

3.2.1 Start Up Main Module

The file "__main.c" has only one function that simply receives a call from start.s and calls the main function in "main.c"

3.2.1.1 __main: Main Module Caller

Call the main module.

Prototype `void __PASCAL __main(void)`

Input None

Output None

Return Codes None

3.2.2 System Functions

The files "_system.c" and "_system.h" provide functions to initialize the system and manage interrupts.

3.2.2.1 DisableExternalInerrupts: Disable Interrupts

Disable interrupts so execution of the program will not be stopped.

Prototype `void DisableExternalInterrupts(void)`

Input None

Output None

Return Codes None

3.2.2.2 EnableExternalInerrupts: Enable Interrupts

Enable Interrupts.

Prototype `void EnableExternalInterrupts(void)`

Input None

Output None

Return Codes None

3.2.2.3 ReenableInerrupts: Re-Enable Interrupts

Re-enable interrupts if it has been disabled before.

Prototype `void ReenableInterrupts(void)`

Input None

Output None

Return Codes None

3.2.2.4 sysInitialize: Initialize System

Initialize the system by disabling timer and enabling interrupts.

Prototype `void sysInitialize(void)`

Input None

Output None

Return Codes None

3.2.3 Linker Input Command File

The file “c68.ld” configures the linker to generate the executable image. The files that need to be linked are listed in this file.

3.2.4 Line Card Command Parameter Handler

The files “com.c” and “com.h” handle the parameter of commands specific to the line card.

3.2.4.1 strtolower: Convert String to Lower Case

Convert any upper case characters in a string to lower case.

Prototype `void strtolower(char *strTmp)`

Input `strTmp`: The string to be converted. This string is overwritten as it is used for output as well.

Output `strTmp`: The string after conversion.

Return Codes None

3.2.4.2 findArg: Find Command Argument

Call the appropriate function based on the arguments passed in the command line.

Prototype `void findArg(char **szArg, ARG_ENTRY *aArgs)`

Input `szArgs`: Array of arguments of the command.

`aArgs`: List of acceptable arguments and the appropriate function to call.

Output None

Return Codes None

3.2.4.3 notifyError: Argument Error Notification

Notifies the user that the arguments in the command have error(s). This is done by sending an error message through the serial port.

Prototype `void notifyError(void)`

Input None

Output None

Return Codes None

3.2.4.4 cmdActivate: Activate Command Handler

Handle the activate command by calling the Duplex driver API to activate the S/UNI-DUPLEX device.

Prototype `void cmdActivate(char** szArgs)`

Input `szArgs`: Array of arguments of the command.

Output None

Return Codes None

3.2.4.5 hlpActivate: Activate Help Information

Display a short description of what the activate command does.

Prototype `void hlpComment(void)`

Input None

Output None

Return Codes None

3.2.4.6 cmdClearStat: Clear Statistics Command Handler

Handle the clear statistics command by calling the Duplex driver API to reset all statistical counts of the S/UNI-DUPLEX device.

Prototype `void cmdClearStat(char** szArgs)`

Input `szArgs`: Array of arguments of the command.

Output None

Return Codes None

3.2.4.7 hlpClearStat: Clear Statistics Help Information

Display a short description of what the clear statistics command does.

Prototype `void hlpClearStat(void)`

Input None

Output None

Return Codes None

3.2.4.8 cmdCounters: Count Command Handler

Handle the count command.

Prototype `void cmdCounters(char** szArgs)`

Input `szArgs`: Array of arguments of the command.

Output None

Return Codes None

3.2.4.9 hlpCounters: Count Help Information

Displays a short description of what the count command does.

Prototype `void hlpCounters(void)`

Input None

Output None

Return Codes None

3.2.4.10 CountersDuplex: Duplex Device Counters

Call the Duplex driver API to obtain counters stored in the S/UNI-DUPLEX device registers and display them.

Prototype `void CountersDuplex(char** szArgs)`

Input `szArgs`: Array of arguments of the command.

Output None

Return Codes None

3.2.4.11 CountersComet1: Comet Device Counters

Display the counters in a Comet Device. This is done by calling `CountersCometAny` with the appropriate parameters. The number at the end of the function specifies the COMET number (1-16) that the function is for.

Prototype `void CountersComet1(char** szArgs)`

Input `szArgs`: Array of arguments of the command.

Output None

Return Codes None

3.2.4.12 CountersCometAny: Comet Device Counters

Call the Comet driver to obtain the values of the counters from the register and displays them.

Prototype `void CountersCometAny(int intCometIndex, char** szArgs)`

Input `intCometIndex`: Index for the COMET device (0-15).

`szArgs`: Array of arguments of the command.

Output None

Return Codes None

3.2.4.13 cmdDeactivate: Deactivate Command Handler

Handle the deactivate command by calling the Duplex driver API to deactivate the S/UNI-DUPLEX device.

Prototype `void cmdDeactivate(char** szArgs)`

Input `szArgs`: Array of arguments of the command.

Output None

Return Codes None

3.2.4.14 hlpDeactivate: Deactivate Help Information

Display a short description of what the deactivate command does.

Prototype `void hlpDeactivate (void)`

Input None

Output None

Return Codes None

3.2.4.15 cmdInit: Initialize Command Handler

Initialize command handler.

Prototype `void cmdInit(char** szArgs)`

Input `szArgs`: Array of arguments of the command.

Output None

Return Codes None

3.2.4.16 hlpInit: Initialize Command Help Information

Display a short description of what the initialize command does.

Prototype `void hlpInit(void)`

Input None

Output None

Return Codes None

3.2.4.17 InitAll: Initialize All Devices

Initialize all the S/UNI-DUPLEX and COMET devices.

Prototype `void InitAll(char** szArgs)`

Input `szArgs`: Array of arguments of the command.

Output None

Return Codes None

3.2.4.18 InitDuplex: Initialize Duplex Device

Calls the Duplex driver API to initialize the S/UNI-DUPLEX device.

Prototype `void InitDuplex(char** szArgs)`

Input `szArgs`: Array of arguments of the command.

Output None

Return Codes None

3.2.4.19 InitComet: Initialize All Comet Devices

Initialize all the COMET devices.

Prototype `void InitComet(char** szArgs)`

Input `szArgs`: Array of arguments of the command.

Output None

Return Codes None

3.2.4.20 InitComet1: Initialize Comet Device

Initialize one Comet Device. This is done by calling `InitCometAny` with the appropriate parameters. The number at the end of the function specifies the COMET number (1-16) that the function is for.

Prototype `void InitComet1(char** szArgs)`

Input `szArgs`: Array of arguments of the command.

Output None

Return Codes None

3.2.4.21 InitCometAny: Initialize Comet Device Helper

Call the Comet driver to initialize a specified COMET device.

Prototype `void InitCometAny(int intCometIndex, char** szArgs)`

Input `intCometIndex`: Index for the COMET device (0-15).

`szArgs`: Array of arguments of the command.

Output None

Return Codes None

3.2.4.22 cmdLoopback: Loopback Command Handler

Handle the loopback command.

Prototype `void cmdLoopback(char** szArgs)`

Input `szArgs`: Array of arguments of the command.

Output None

Return Codes None

3.2.4.23 hlpLoopback: Loopback Command Help Information

Display a short description of what the loopback command does.

Prototype `void hlpLoopback(void)`

Input None

Output None

Return Codes None

3.2.4.24 LoopbackDuplex: Duplex Device Loopback

Call the Duplex driver API to enable/disable loopback for the S/UNI-DUPLEX device.

Prototype `void LoopbackDuplex(char** szArgs)`

Input `szArgs`: Array of arguments of the command.

Output None

Return Codes None

3.2.4.25 LoopbackComet: All Comet Devices Loopback

Enable/disable loopback for all COMET devices.

Prototype `void LoopbackComet(char** szArgs)`

Input `szArgs`: Array of arguments of the command.

Output None

Return Codes None

3.2.4.26 LoopbackComet1: Comet Device Loopback

Enable/disable loopback for one COMET device. This is done by calling `LoopbackCometAny` with the appropriate parameters. The number at the end of the function specifies the COMET number (1-16) that the function is for.

Prototype `void LoopbackComet1(char** szArgs)`

Input `szArgs`: Array of arguments of the command.

Output None

Return Codes None

3.2.4.27 LoopbackCometAny: Comet Device Loopback Helper

Call the Comet driver to enable/disable loopback on a specified COMET device.

Prototype `void LoopbackCometAny(int intCometIndex, char** szArgs)`

Input `intCometIndex`: Index for the COMET device (0-15).

`szArgs`: Array of arguments of the command.

Output None

Return Codes None

3.2.4.28 cmdRead: Read Command Handler

Read Command Handler.

Prototype `void cmdRead(char** szArgs)`

Input `szArgs`: Array of arguments of the command.

Output None

Return Codes None

3.2.4.29 hlpRead: Read Command Help Information

Display a short description of what the read command does.

Prototype `void hlpRead(void)`

Input None

Output None

Return Codes None

3.2.4.30 ReadDuplex: Duplex Device Read Register

Call the Duplex driver API to read a register from the S/UNI-DUPLEX device.

Prototype `void ReadDuplex(char** szArgs)`

Input `szArgs`: Array of arguments of the command.

Output None

Return Codes None

3.2.4.31 ReadComet1: Comet Device Read Register

Read a register from a COMET device. This is done by calling `ReadCometAny` with the appropriate parameters. The number at the end of the function name specifies the COMET number (1-16) that the function is for.

Prototype `void ReadComet1(char** szArgs)`

Input `szArgs`: Array of arguments of the command.

Output None

Return Codes None

3.2.4.32 ReadCometAny: Comet Device Read Register Helper

Call the Comet driver to read a register from a Comet device.

Prototype
```
void ReadCometAny(int intCometIndex, char** szArgs)
```

Input `intCometIndex`: Index for the COMET device (0-15).

`szArgs`: Array of arguments of the command.

Output None

Return Codes None

3.2.4.33 ReadFPGA: FPGA Read Register

Read a register from the FPGA.

Prototype
```
void ReadFPGA(char** szArgs)
```

Input `szArgs`: Array of arguments of the command.

Output None

Return Codes None

3.2.4.34 ReadAbsolute: Absolute Address Read

Read from an absolute address.

Prototype
```
void ReadAbsolute(char** szArgs)
```

Input `szArgs`: Array of arguments of the command.

Output None

Return Codes None

3.2.4.35 cmdReset: Reset Command Handler

Handle the reset command.

Prototype `void cmdReset(char** szArgs)`

Input `szArgs`: Array of arguments of the command.

Output None

Return Codes None

3.2.4.36 hlpReset: Reset Command Help Information

Display a short description of what the reset command does.

Prototype `void hlpReset(void)`

Input None

Output None

Return Codes None

3.2.4.37 ResetAll: Reset All Devices

Reset all the devices and re-start the firmware.

Prototype `void ResetAll (char** szArgs)`

Input `szArgs`: Array of arguments of the command.

Output None

Return Codes None

3.2.4.38 ResetDuplex: Reset Duplex Device

Call the Duplex driver API to do a software reset on the S/UNI-DUPLEX device.

Prototype `void ResetDuplex(char** szArgs)`

Input `szArgs`: Array of arguments of the command.

Output None

Return Codes None

3.2.4.39 ResetComet: Reset All Comet Devices

Call the Comet driver to reset all the COMET devices.

Prototype `void ResetComet(char** szArgs)`

Input `szArgs`: Array of arguments of the command.

Output None

Return Codes None

3.2.4.40 ResetComet1: Reset Comet Device

Call the Comet driver to reset a Comet Device. The number at the end of the function specifies the Comet number (1-16) that the function is for.

Prototype `void ResetComet1(char** szArgs)`

Input `szArgs`: Array of arguments of the command.

Output None

Return Codes None

3.2.4.41 cmdStat: Statistics Command Handler

Handle the statistics command by calling the Duplex driver API to return all statistical counts of the S/UNI-DUPLEX device and displaying them.

Prototype `void cmdStat(char** szArgs)`

Input `szArgs`: Array of arguments of the command.

Output None

Return Codes None

3.2.4.42 hlpStat: Statistics Help Information

Display a short description of what the statistics command does.

Prototype `void hlpStat(void)`

Input None

Output None

Return Codes None

3.2.4.43 cmdWrite: Write Command Handler

Handle the write command.

Prototype `void cmdWrite(char** szArgs)`

Input `szArgs`: Array of arguments of the command.

Output None

Return Codes None

3.2.4.44 hlpWrite: Write Command Help Information

Displays a short description of what the write command does.

Prototype `void hlpWrite(void)`

Input None

Output None

Return Codes None

3.2.4.45 WriteDuplex: Duplex Device Write

Call the Duplex driver API to write to a register of the S/UNI-DUPLEX device.

Prototype `void WriteDuplex(char** szArgs)`

Input `szArgs`: Array of arguments of the command.

Output None

Return Codes None

3.2.4.46 WriteComet1: Comet Device Write

Write to a register of a COMET device. This is done by calling `WriteCometAny` with the appropriate parameters. The number at the end of the function name specifies the COMET number (1-16) that the function is for.

Prototype `void WriteComet1(char** szArgs)`

Input `szArgs`: Array of arguments of the command.

Output None

Return Codes None

3.2.4.47 WriteCometAny: Comet Device Write Helper

Call the Comet driver to write to a register of a Comet device.

Prototype `void WriteCometAny(int intCometIndex, char** szArgs)`

Input `intCometIndex`: Index for the COMET device (0-15).

`szArgs`: Array of arguments of the command.

Output None

Return Codes None

3.2.4.48 WriteFPGA: FPGA Write

Write to a register of the FPGA.

Prototype `void WriteFPGA(char** szArgs)`

Input `szArgs`: Array of arguments of the command.

Output None

Return Codes None

3.2.4.49 WriteAbsolute: Absolute Address Write

Write to an absolute address.

Prototype `void WriteAbsolute(char** szArgs)`

Input `szArgs`: Array of arguments of the command.

Output None

Return Codes None

3.2.5 Comet Drivers

The files “comet.c” and “comet.h” provide a simple driver for the COMET devices. The functions provided are discussed below.

3.2.5.1 cometInit: Initializing Comet Devices

This function initializes the device based on the initialization mode passed in. It configures the device registers accordingly.

Prototype `int cometInit (COMET pComet, int mode)`

Input `pComet`: Base address of the COMET device to be initialized.

mode: Identifies the mode that the COMET chip will operate, which can be one of:

- `COMET_T1_S0`
- `COMET_E1_120`

Output None

Return Codes `COMET_SUCCESS` (Function was completed successfully)

`COMET_ERROR` (There was an error)

3.2.5.2 cometLoopback: Setup/Disable Loopback Modes

This function enables a loopback mode or disables loopback based on the loopback type passed in.

Prototype `int cometLoopback(COMET comet, unsigned char LoopbackType)`

Input pComet: Base address of the COMET device.

LoopbackType: Identifies the mode of loopback, which can be one of:

- COMET_LPBK_LINE
- COMET_LPBK_PAYLOAD
- COMET_LPBK_DIAG
- COMET_LPBK_DISABLE

Output None

Return Codes COMET_SUCCESS (Function was completed successfully)

COMET_ERROR (There was an error)

3.2.5.3 cometRead: Reading from Comet Registers

This function reads the contents of a register of a COMET device.

Prototype
```
cometRead(COMET comet, unsigned int uiRegId,
unsigned char *value)
```

Input pComet: Base address of the COMET device.

uiRegId: Address offset of the register

Output value: Register value

Return Codes COMET_SUCCESS (Function was completed successfully)

COMET_ERROR (There was an error)

3.2.5.4 cometReset: Resetting Comet Devices

This function applies a software reset to the COMET device.

Prototype
```
int cometReset(COMET pComet)
```

Input pComet: Base address of the COMET device to be reset.

Output None

Return Codes COMET_SUCCESS (Function was completed successfully)

COMET_ERROR (There was an error)

3.2.5.5 cometWrite: Write to Comet Registers

This function writes a value to a register of a COMET device.

Prototype `int cometWrite(COMET comet, unsigned int uiRegId, unsigned char value)`

Input pComet: Base address of the COMET device to be initialized.

uiRegId: Address offset of the register

value: Register value

Output None

Return Codes COMET_SUCCESS (Function was completed successfully)

COMET_ERROR (There was an error)

3.2.6 Duplex Drivers

The following files are part of the Duplex driver. The generic Duplex drivers were ported to work under this specific environment. Please refer to the Duplex Device Driver Design Specification[3] for detail explanation.

dpx.c
dpx.h
dpx_api.c
dpx_api.h
dpx_err.h
dpx_hw.c
dpx_hw.h
dpx_rtos.c
dpx_rtos.h
dpx_test.c
dpx_test.h
sysPci.h

3.2.7 Event Manager Module

The files “events.c” and “events.h” implements the event manager module that is used to handle incoming commands.

3.2.7.1 evAddRecordToList: Add Event Record

Places an event record, whose fields have been appropriately filled in, at the tail of the queue of events to be processed.

Prototype `int evAddRecordToList(EVENT_RECORD *pRecord)`

Input `pRecord`: Address of the event record to be queued.

Output None

Return Either 0 (Failure) or 1 (Success).

3.2.7.2 evInitialize: Event Module Initialization

Initializes event manager data structures. Must be called before any other event manager routine.

Prototype `void evInitialize(void)`

Input None

Output None

Return None

3.2.7.3 evGetFreeRecord: Get Next Free Event

Returns address of an empty event record and removes that record from the pool of free event records.

Prototype `EVENT_RECORD *evGetFreeRecord(void)`

Input None

Output None

Return address of next free event record

3.2.7.4 evGetNext: Get Next Event

Returns address of next event record to be processed.

Prototype `EVENT_RECORD *evGetNext(void)`

Input None

Output None

Return address of next event record to be processed

3.2.7.5 evReturnFreeRecord: Return Free Event Record

Returns an event record which is no longer required to the pool of free event records.

Prototype `int evReturnFreeRecord(EVENT_RECORD *pRecord)`

Input `pRecord`: Address of the event record to be freed.

Output None

Return Either 0 (Failure) or 1 (Success).

3.2.8 Global Variables and Definitions

The files “globals.c” and “globals.h” provides the global variables and definitions used by other modules. There is no function that can be called in this module.

3.2.9 Line Card Initialization

The files “linecard.c” and “linecard.h” provides functions related to the setup of the line card.

3.2.9.1 lineDuplexInit: Setup Duplex Drivers

Sets up the Duplex driver. Also, initializes and activates all devices.

Prototype `int lineDuplexInit(void)`

Input None

Output None

Return Codes DPX_SUCCESS (0) or an error value.

3.2.10 Main Module

The file “main.c” is the main module of the firmware.

3.2.10.1 main: Main module

This is the main module of the firmware. It calls the required initialization routines and starts main event loop of the command parser.

Prototype `main (int argc, char **argv)`

Input `argc`: This parameter is ignored.

`argv`: This parameter is ignored.

Output None

Return Codes None

3.2.11 Serial Port Interface

The files “serial.c” and “serial.h” handles serial communication with the host system.

3.2.11.1 sioInit: Serial Port Initialization

Initializes the serial port and related variables.

Prototype `int sioInit(int baud)`

Inputs `baud`: baud rate (bits per second) at which to activate the serial port

Output None

Return Codes Either 0 (Success) or 1 (Failure).

3.2.11.2 sioEchoControl: Terminal Echo Control

Enables/disables echoing of incoming characters (duplexing).

Prototype `int sioEchoControl (int iFlag)`

Inputs `iFlag`: 1 to turn echoing on; 0 to turn echoing off

Output None

Return Codes Either 0 (Success) or 1 (Failure).

3.2.11.3 sioSendChars: Send Characters

Send a specified number of characters through the serial port.

Prototype `int sioSendChars(char *pBuffer, int iSize)`

Inputs `szBuffer`: pointer to characters to be send

`iSize`: number of characters to send

Output None

Return Codes Number of characters which were sent through the serial port.

3.2.11.4 sioSendString: Send String

Send a string through the serial port. This is done by calling `sioSendChars` with the correct parameters.

Prototype `int sioSendString(char* szBuffer)`

Inputs `szBuffer`: pointer to characters to be send

Output None

Return Codes Number of characters which were sent through the serial port.

3.2.12 Command Parser

The files “sioMain.c” and “sioMain.h” provides functions to parse commands and handle generic commands not specific to the line card.

3.2.12.1 sioMain: Main Event Loop

Infinite loop that waits for commands and calls the user command handler to handle them.

Prototype `void sioMain (void)`

Input None

Output None

Return Codes None

3.2.12.2 UserCommand: User Command Handler

This function receives an event (command) and executes the appropriate function that corresponds to that command.

Prototype `void UserCommand(EVENT_RECORD *pRec)`

Input `pRec`: Event record containing the command

Output None

Return Codes None

3.2.12.3 cmdComment: Comment Command Handler

An empty function used to ignore all comment lines.

Prototype `void cmdComment(char **szArgs)`

Input `szArgs`: Array of arguments of the command.

Output None

Return Codes None

3.2.12.4 cmdHelp: Help Command Handler

Handles the help command.

Prototype `void cmdHelp(char **szArgs)`

Input `szArgs`: Array of arguments of the command.

Output None

Return Codes None

3.2.12.5 hlpComment: Comment Help Information

Displays a short description of how to write comments.

Prototype `void hlpComment(void)`

Input None

Output None

Return Codes None

3.2.12.6 hlpHelp: Help Command Help Information

Displays a short description of the help command.

Prototype `void hlpHelp(void)`

Input None

Output None

Return Codes None

3.2.13 Microprocessor Initialization

The assembly file “start.s” contains the list of interrupt vectors. It also initializes the microprocessor (Motorola 68340) and sets up chip selects. This code is given control when the 68340 resets.

3.2.14 ANSI Library Update

The file “strtoul.c” replaces the `strtoul` function provided by the standard library. Early versions of the IntrolC compiler came with a very old implementation of the ANSI library which provides less functionality so this file is used instead. The `strtoul` function provided is the same as the current ANSI implementation.

4 REFERENCES

1. PMC-Sierra Inc., PMC-1970624, “COMET - Combined E1/T1 Transceiver/Framer Long Form Data Sheet”, July 1999, Issue 8.

2. PMC-Sierra Inc., PMC-1980581, “S/UNI-DUPLEX Dual Serial Link PHY Multiplexer Data Sheet”, April 2000, Issue 5.

3. PMC-Sierra Inc., PMC-1990799, “S/UNI-DUPLEX Device Driver Manual”, July 1999, Issue 1.

4. PMC-Sierra Inc., PMC-1990832, “DSLAM Reference Design: System Design”, September 2000, Issue 3.

5. PMC-Sierra Inc., PMC-1990354, “DSLAM Reference Design: Line Card”, September 2000, Issue 3.

CONTACTING PMC-SIERRA, INC.

PMC-Sierra, Inc.
105-8555 Baxter Place Burnaby, BC
Canada V5A 4V7

Tel: (604) 415-6000

Fax: (604) 415-6200

Document Information: document@pmc-sierra.com
Corporate Information: info@pmc-sierra.com
Application Information: apps@pmc-sierra.com
(604) 415-4533
Web Site: http://www.pmc-sierra.com

PMC-2000504 (R2) Issue date: November 2000