RS8228

ATM Convergence Interface Software
Software Programming Guide

500126A
June 2002

CONEXANT™

Revision History

Revision	Level	Date	Description
A	X	September 2000	Created.
B	XX	June 2001	Finished.
C	XX	June 2002	Added Ch. 7.5 re: Rcv FIFO Overflow workaround. Deleted DRV_ACTIVE_NO_INT value from all DRV_MONITOR_ parameters, as it is unsupported in software. Non-Tech Pubs document release.

Reader Response: Conexant strives to produce quality documentation and welcomes your feedback. Please send comments and suggestions to conexant.tech.pubs@conexant.com. For technical questions, contact your local Conexant sales office or field applications engineer.

Table of Contents

List of Figures

List of Tables

1.0 Introduction

RS8228 interface software product is a Mindspeed Technologies' ANSI C language device driver.

The interface software provides a simplified application programming interface (API) that is both operating system and processor independent, allowing portability across target environments. The application uses a limited set of API functions to control and monitor the device via logical parameters. The interface software is distributed as commented source code and includes a PDF format software manual. The software provides facility failure integration and performance monitoring that is compliant with applicable telecom standards.

The software includes an embedded debug monitor that may optionally be compiled providing debug features that can be used to assist in initial hardware debugging.

1.1 Product Overview

Mindspeed Technologies's RS8228 driver interface software provides a parameter driven application programming interface (API). As illustrated in Figure 1-1, the API consists of a limited set of operating system independent ANSI C function calls, data structures, and parameters to simplify software control of the device.

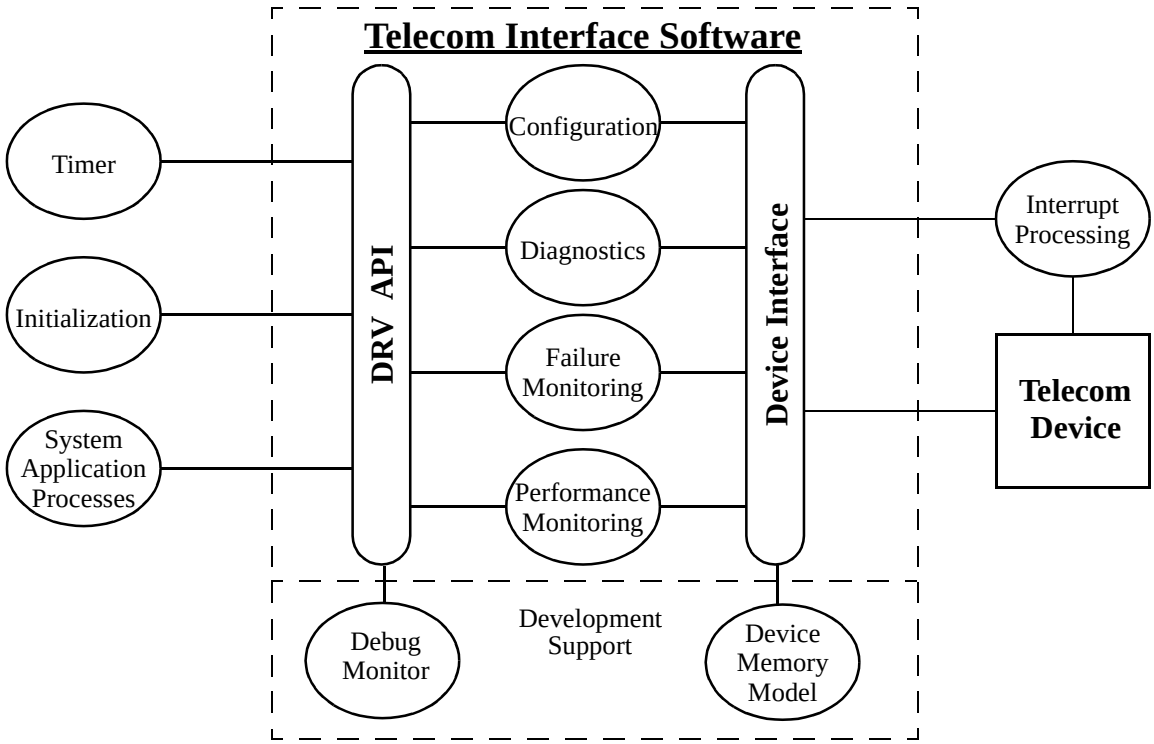
The software provides a polling function which performs the standards-compliant failure integration and performance monitoring features. The software also includes a function to process interrupts to reduce latency time for detecting errors, events, and defects. The software notifies the application of changes in the device state via an event callback function when changes to facility and device internal states are detected. The software includes an embedded monitor as a compiler option which can aid in hardware and software debugging.

All drivers in the telecom interface software family include a common set of core driver files that use an extensive table-driven architecture. Driver versions are available for various telecom devices and telecommunications standards.

The driver package includes device initialization code, configuration and diagnostic parameters, and support for failure and performance monitoring according to applicable telecom standards. Some packages also include advanced features (i.e. HDLC, FEAC, etc.).

The driver products are distributed as commented ANSI C source code and a PDF driver manual. The products are normally distributed on DOS-compatible floppy discs, 8 mm tape, or electronically via the Internet in a compressed and uuencoded tar file format. Other distribution options may be available upon special request.

Figure 1-1. Telecom Interface Software



1.1.1 Telecom Interface Software Features

- OS and processor independent ANSI C implementation.
- Initialization and diagnostic sequences.
- Commented source code.
- Meets applicable ANSI, Bellcore, ATM Forum, and ITU-T standards.
- Defect detection, failure monitoring and performance monitoring according to telecom standards.
- Device specific event/error monitoring.
- Embedded debug monitor program to assist with engineering activities.

1.1.2 Telecom Interface Software Benefits

- Common API across different PHY Layers promotes application code reuse.
- Internal design allows related telecom functions to be easily incorporated into DRV API.
- Common software architecture allows memory-efficient support for multiple devices.

2.0 References

American National Standards Institute (ANSI) Documents.

(1) ANSI T1.627-1993, Broadband ISDN - ATM Layer Functionality and Specification

(2) ANSI T1.646-1995, Broadband ISDN - Physical Layer Specification for User-Networke Interfaces Including DS1/ATM

ATM Forum Documents

(3) ATM Forum, ATM User-Network Interface Specification, V3.1, October 1995.

(4) ATM Forum, UTOPIA, An ATM PHY Interface Specification, Level 2, Version 1, June 1995.

(5) ATM Form, 94-0406R5, E3 (34,368 kbps_ Physical Layer Interface, December 21, 1994.

(6) ATM Forum, 95-1207R1, DS3 Physical Layer Interface, December 1995.

(7) ATM Forum, af-phy-0029.00, 6,312 Kbps UNI Specification, Version 1.0, June 1995.

Conexant Documents

(8) 100064B, March 2000, Conexant Systems, Inc., RS8228 Octal ATM Transmission Convergence PHY Device

3.0 Functional Descriptions

The driver functionality is logically divided into five subsystems: configuration (CF), diagnostics (DG), overhead processing (OH), failure monitoring (FM), and performance monitoring (PM). These subsystems are described in Section 3.1 - Section 3.5.

From the API programmer's perspective, there is no distinction between parameters in the various subsystems. The API get and set functions simply accept a parameter ID and return the associated value. Assigning parameter IDs to functional subsystems reduces the time required for parameter lookups. The application software uses the **DRV_get**()/**DRV_set()** and **DRV_trib_get**()/**DRV_trib_set()** functions described in Section 4.3.2 to access the parameter values. The set functions return an error code if called with an illegal parameter or value and does not affect the device. The driver will translate valid parameter values and write the appropriate device register(s).

Valid parameter values are stored in the driver's internal data structure. For read/write parameters, the get functions normally return the data from this internal structure and perform no device accesses.

This document groups the parameters by their assigned subsystem to provide some context for how the parameters are used. Often the subsystem a parameter is assigned to is unclear. For instance, a parameter that affects the generation of overhead bits may be assigned to the CF subsystem. This does not affect the API but affects the document section where the parameter appears.

3.1 Configuration (CF)

The CF subsystem controls the telecom device's operational state. All parameters assigned to the CF subsystem are read/write parameters and affect the device operation.

3.2 Diagnostics (DG)

The parameters assigned to the DG subsystem control testing features of the telecom device. These are read/write parameters that control things like signal loopbacks and forced error conditions. They are used during testing or maintenance conditions, and will affect the signal throughput. These

parameters should be used with care in equipment connected to the network as errors will be introduced into the network.

3.2.1 Diagnostic Tests

The DG subsystem optionally executes device test sequences when invoked by the user or during driver initialization to verify its operation. These tests are used primarily to assist in initial hardware development to help debug the device's processor interface. The specific tests and their behavior are described in the following sections. The API function **DRV_test**() is described in Section 4.3.4.

3.2.2 DRV_DEFAULT_TEST

The default diagnostic test performs a software device reset and compares the registers against their default values. This test can be optionally run during initialization. The test can also be requested by the application. Operation of the telecom device is disrupted during the execution of this test and register values are not saved or restored by the test.

3.2.3 DRV_DESTRUCT_REG_TEST

In this test, the writable registers within the device are written and read back with a number of different patterns. This test verifies the integrity of the hardware connection with the telecom device and some of the internal circuitry of the device. The destructive register test is run if the *destructive_reg_test* field in the driver initialization structure is set to DRV_ACTIVE. This test can also be requested by the application via the DRV_test() function.

When the destructive register test is performed, normal operation of the telecom device is disrupted (errors and alarms will be transmitted).

3.3 Overhead Processing (OH)

The OH subsystem is responsible for monitoring and configuring the facility and telecom device overhead parameters. Generally, the transmit parameters are readable and writable, while the receive parameters are read-only. The read-only parameters are updated directly from the device for each **DRV_{trib_}get()** function call.

3.4 Failure Monitoring (FM)

The FM subsystem monitors the telecom device for defects and anomalies and integrates the defects into failures according to applicable telecom standards.

In standard telecommunications terminology, the physical and logical elements comprising a telecommunications interface have defined failure states and conditions.

The driver detects changes in the state of the underlying defects and anomalies through a combination of interrupts and/or polling. If the interrupt capability is not used, the driver will poll the device obtain the latest state information. If interrupts are used, the driver will assume no change in state has occurred unless interrupted. For more information on the use of interrupts, see Section 7.7.2. Please note that some advanced features of certain devices (i.e. data link, FEAC, etc.) may require the use of interrupts to operate properly.

The user has control over which defect and failure indicators are monitored and the length of both the activation and decay times. Upon initialization the failure indications required by the various telecom standards are enabled and the activation and decay times are set at 2.5 and 10 seconds, respectively

3.5 Performance Monitoring (PM)

The PM subsystem is the set of functions and capabilities necessary for a network element (NE) to gather, store, and report performance data associated with its monitored digital transmission entities. In contrast with alarm/status indications, performance parameters are quantitative, not binary, in nature. These performance parameters are gathered over predetermined accumulation periods. The driver calculates these statistics over 15 minute and 24 hour intervals. The PM counts are available to the application individually or grouped in a structure encompassing one of the four accumulation sets: the current 15 minute interval, the current 24 hour day interval, the previous 15 minute interval, and the previous 24 hour day interval. The PM subsystem uses the FM anomaly, defect, and failure information, where applicable, to calculate its performance statistics.

Upon initialization, the PM subsystem is disabled and no statistics are calculated.

4.0 Interface Descriptions

The following sections summarize the API of the telecom interface software family of device drivers. The various functional subsystems are discussed in Section 3.0. The following sections list the syntax of the functions with more precision. It is important to point out that the telecom interface software family of products can be configured to run in different operating environments.

Table 4-1 summarizes the API of the driver software. All functions require a pointer to the structure user allocates DRV_DEVICE structure. The additional parameters for each function are listed in Table 4-1.

Table 4-1. RS8228 API Functions

Class	Function	Additional Parameters (all functions require DRV_DEVICE *)
Initialization	DRV_init_default ()	Pointer to the initialization structure.
	DRV_init ()	Pointer to the initialization structure.
Interrupts	DRV_tick ()	none
	DRV_intr ()	Interrupt source (DRV_DEVICE_INTR).
Applications	DRV_read ()	Register offset to be read and a register offset to place the data read.
	DRV_write ()	Register offset to be written and the data to be written.
	DRV_set ()	Parameter ID to be configured and the parameter value.
	DRV_trib_set ()	Tributary parameter ID to be configured, tributary value, and the parameter value.
	DRV_get ()	Parameter ID to be retrieved and a pointer to receive the parameter value.
	DRV_trib_get()	Tributary parameter ID to be retrieved, tributary value, and a pointer to receive the parameter value.
	DRV_test ()	An identifier for which test to run.
Device FM Interface	RS8228_FM_status ()	Pointer to a RS8228_FM_STRUCTS data structure. This structure combines any physical layer FM states with device specific FM states in one structure.
Device PM Interface	DRV_PM_process()	No additional arguements.

Table 4-1. RS8228 API Functions

Class	Function	Additional Parameters (all functions require DRV_DEVICE *)
Device PM Interface	RS8228_ATM_PM_status ()	Identifier for the PM period and a pointer to the source of the PM state values.
	RS8228_ATM_PM_preset ()	Identifier for the PM period and a pointer to the source of the PM state values.
Monitor	DRV_mon()	A control parameter and a character string containing a monitor command.
User Defined (user writes function and supplies a pointer to function)	USER_event()	A pointer to a data structure describing the event generated by the RS8228 device.

4.1 Operating Environment Support

4.1.1 Initialization

The first call to the driver software should be to set all fields of the initialization structure to default values. The DRV_INIT structure is used to pass any special parameters needed by the driver software during initialization.

4.1.1.1 DRV_init_default

Description: Initializes the fields of the driver initialization structure to default values. This function should be the first driver function called. The use of this function insures new fields added to the initialization structure are initialized thus maintaining backward compatibility between the driver and application code not using the new fields. The fields of the initialization structure may then be configured by the application software, setting the required addresses and parameter values before passing the initialization structure to the DRV_init() function.

Prototype:
```
void DRV_init_default(DRV_DEVICE *dev, DRV_INIT *init);
```

Parameters: **dev**—pointer used to uniquely associate an instance of driver software with a specific device.

init—pointer to the initialization structure containing pointers to the driver structures: DRV_DEVICE, DRV_DEV_CONFIG, and RS8228_DEV_CONFIG. This structure also contains configurations of parameters needed by the DRV_init function. DRV_INIT is defined in the header file “drv.h”.

Return Codes: None.

4.1.1.2 DRV_init

Description: This function initializes the software driver data structures and the device, placing the device into a default operational mode. The device is configured with default parameter values that may differ from the device register default values. Optionally, the communication with the device is verified and diagnostic tests are executed. If this function returns successfully, the driver is now ready to accept additional setup and control information via the **DRV_set**() function.

If further setup is desired, function calls to **DRV_set()** should be made to before calling **DRV_tick()** or introducing device interrupts to avoid generating undesired errors.

Prototype:
```
DRV_RETURN DRV_init(DRV_DEVICE *dev, DRV_INIT *init);
```

Parameters: **dev**—pointer used to uniquely associate an instance of driver software with a specific device.

init—pointer to a structure that contains pointers to the driver structures: DRV_DEVICE, DRV_DEV_CONFIG, and RS8228_DEV_CONFIG. This structure also contains configurations of parameters needed by the DRV_init function. DRV_INIT is defined in the header file “drv.h”.

Return Codes: **DRV_SUCCESS**—No errors detected

DRV_TEST_FAIL—An initialization test failed

DRV_BAD_PARM—Invalid parameter initialization

4.1.2 Timer

A known time interval must be supplied to the driver via the **DRV_tick**() function for proper operation of the FM and PM subsystems. This function must be called even if operating the device in an interrupt-driven environment. Although certain device defects, etc. will be detected immediately via interrupts, **DRV_tick**() must be executed at regular intervals to properly integrate FM failures and determine PM status consistent with applicable standards.

4.1.2.1 DRV_tick

Description: This function must be called at a regular intervals to ensure proper operation of the driver FM and PM subsystems. It polls the device registers, detecting and integrating FM failures and calculating PM statistics according to telecom standards. The period of this interval must be stored in the **DRV_TIMER_TICK_INTERVAL** parameter for the driver to accurately calculate its statistics. The default value is 250 milliseconds at initialization.

Prototype:
```
DRV_RETURN DRV_tick(DRV_DEVICE *dev);
```

Parameters: **dev**—pointer used to uniquely associate an instance of driver software with a specific device.

Return Codes: **DRV_SUCCESS**—No errors detected

DRV_NOT_INIT—driver has not been initialized by DRV_init()

DRV_TRANSITION—A FM state parameter transition has occurred

DRV_TEST_FAIL—A background audit test failed

DRV_TRACE_ERR—Error communicating with optional Path Trace memory

4.2 Device Interrupts and User Defined Functions

The following driver interface functions are optional. The functions named **DRV_xxxx()** are function calls to the driver. The functions named **USER_xxxx()** are user-defined functions called by the driver. The pointers to the user defined functions are set during initialization via fields in the initialization structure. These pointer may also be set after initialization using the **DRV_set()** function.

4.2.1 Interrupts

Once the driver has been initialized with **DRV_init()**, the FM subsystem of the driver software has control over the device interrupts. Prior to initialization, the host system must control the device interrupts to prevent spurious interrupts. When the target environment detects that the telecom device has raised its interrupt line, the driver should be notified via the interrupt function call.

4.2.1.1 DRV_intr

Description: This function must be called when the telecom device interrupt line has been asserted. This function determines what interrupt sources within the device have generated the interrupt and performs the appropriate processing.

If this function is called before the driver has been initialized the return value of DRV_NOT_INIT is returned.

Prototype:
```
DRV_RETURN DRV_intr(DRV_DEVICE *dev, DRV_PARM intr_source);
```

Parameters: **dev**—pointer used to uniquely associate an instance of driver software with a specific device.

intr_source—Definition of the interrupt source (DRV_DEVICE_INTR)

Return Codes: **DRV_SUCCESS**—No errors detected

DRV_NOT_INIT—Driver has not been initialized by DRV_init()

DRV_TRANSITION—A FM state parameter transition has occurred

4.3 Application Interfaces

4.3.1 Low Level

Direct register access to the telecom device is provided to the user for debugging purposes. The ability to read and write device registers is provided. These accesses bypass the parameter mapping mechanism within the driver and may cause discrepancies between the driver data structures and the device registers.

4.3.1.1 DRV_read

Description: This function provides a direct interface to the telecom device registers. The function bypasses the major subsystems of the driver software and directly accesses the device registers. Care must be taken when reading failure and performance monitoring registers that are cleared when read. The driver blocks accesses to the device reserved and undefined registers. This function is primarily used for debugging.

Prototype:

```
DRV_RETURN DRV_read
  (DRV_DEVICE      *dev,
   DRV_REG         reg_offset,
   DRV_REG_DATA    *reg_data);
```

Parameters: **dev**—pointer used to uniquely associate an instance of driver software with a specific device.

reg_offset—device register offset (relative to the device base address) to be read. The register offset definitions in the header file xxxx_h.h file should be used, where xxxx is the device prefix.

reg_data—pointer to the storage for the value read from the device register

Return Codes: **DRV_SUCCESS**—No errors detected

DRV_NOT_INIT—Driver has not been initialized by DRV_init()

DRV_BAD_PARM—Input address invalid (reserved or illegal address offset)

4.3.1.2 DRV_write

Description: This function provides a direct interface to the telcom device registers. This function bypasses the major subsystems of the driver and may cause discrepancies between the driver data structures and the device registers. Care must be taken when writing control information which may disrupt the integrity of the other subsystems. The driver blocks accesses to device reserved and undefined registers. This primary purpose of this function is for debugging.

Prototype:

```
DRV_RETURN DRV_write
  (DRV_DEVICE          *dev,
   DRV_REG             reg_offset,
   const DRV_REG_DATA  reg_data);
```

Parameters:

dev—pointer used to uniquely associate an instance of driver software with a specific device.

reg_offset—device register offset (from the device base address) to be written. The register offset definitions in the header file xxxx_h.h should be used, where xxxx is the device prefix.

reg_data—The value to be written to the register

Return Codes:

DRV_SUCCESS—No errors detected

DRV_NOT_INIT—Driver has not been initialized by DRV_init()

DRV_BAD_PARM—Input address invalid (reserved or illegal address offset)

4.3.2 Basic DRV-RS8228 Parameter Interface

Although the driver parameters are divided into subsystems, the user can access most of the parameters through four API functions. One pair of functions, **`DRV_set()`** and **`DRV_get()`** are used to access scalar, or non-tributary, parameters in the driver. Many of the RS8228 parameters are indexed by a tributary value, and these parameters are accessed using **`DRV_trib_set()`** and **`DRV_trib_get()`**.

4.3.2.1 DRV_set

Description: This function is used to set driver non-tributary parameters. The list of valid parameters is found in Section 6.0.

Prototype:
```
DRV_RETURN DRV_set
  (DRV_DEVICE       *dev,
   DRV_PARM_ID      id,
   const DRV_PARM   src);
```

Parameters: **dev**—pointer used to uniquely associate an instance of driver software with a specific device.

id—Identifier of driver parameter to be configured

src—The value being set for the parameter

Return Codes: **DRV_SUCCESS**—No errors detected

DRV_NOT_INIT—Driver has not been initialized by DRV_init()

DRV_BAD_PARM—Invalid parameter ID or parameter value

4.3.2.2 DRV_get

Description: This function is used to read the driver non-tributary parameters. The list of valid parameters is found in Section 6.0.

Prototype:
```
DRV_RETURN DRV_get
  (DRV_DEVICE          *dev,
   DRV_PARM_ID         id,
   DRV_PARM            *dest);
```

Parameters: **dev**—pointer used to uniquely associate an instance of driver software with a specific device.

id—Identifier of driver parameter to be retrieved

dest—Pointer to value being retrieved for the parameter

Return Codes: **DRV_SUCCESS**—No errors detected

DRV_NOT_INIT—Driver has not been initialized by DRV_init()

DRV_BAD_PARM—Invalid parameter ID or parameter value

4.3.2.3 DRV_trib_set

Description: This function is used to set driver non-tributary parameters. The list of valid parameters is found in Section 6.0.

Prototype:
```
DRV_RETURN DRV_trib_set
  (DRV_DEVICE          *dev,
   DRV_PARM_ID         id,
   const DRV_PARM      src,
   DRV_TRIB            trib);
```

Parameters: **dev**—pointer used to uniquely associate an instance of driver software with a specific device.

id—Identifier of driver tributary parameter to be configured

src—The value being set for the parameter

trib—The tributary of the parameter to be configured

Return Codes: **DRV_SUCCESS**—No errors detected

DRV_NOT_INIT—Driver has not been initialized by DRV_init()

DRV_BAD_PARM—Invalid parameter ID or parameter value

DRV_BAD_TRIB—Invalid tributary for this parameter

4.3.2.4 DRV_trib_get

Description: This function is used to read the driver non-tributary parameters. The list of valid parameters is found in Section 6.0.

Prototype:
```
DRV_RETURN DRV_trib_get
  (DRV_DEVICE     *dev,
   DRV_PARM_ID    id,
   DRV_PARM       *dest,
   DRV_TRIB       trib);
```

Parameters: **dev**—pointer used to uniquely associate an instance of driver software with a specific device.

id—Identifier of driver tributary parameter to be retrieved

dest—Pointer to value being retrieved for the parameter

trib—Tributary of the parameter to be retrieved

Return Codes: **DRV_SUCCESS**—No errors detected

DRV_NOT_INIT—Driver has not been initialized by DRV_init()

DRV_BAD_PARM—Invalid parameter ID or parameter value

DRV_BAD_TRIB—Invalid tributary for this parameter

4.3.3 Overhead Trace Access

If the optional external overhead trace feature is enabled, three overhead trace message strings are passed between the host application and the driver software for each overhead trace layer. These strings are the expected receive overhead trace message, the actual receive overhead trace message, and the transmit overhead trace message. These ASCII character arrays are accessed via the following functions:

4.3.3.1 DRV_trace_set

Description: This function sets the expected receive trace message and the transmit trace message.

Prototype:
```
DRV_RETURN DRV_trace_set
  (DRV_DEVICE          *dev,
   DRV_PARM_ID         id,
   const DRV_TRACE     src);
```

Parameters: **dev**—pointer used to uniquely associate an instance of driver software with a specific device.

id—Identifier of overhead trace parameter to be configured

src—Array holding an overhead trace message

Return Codes: **DRV_SUCCESS**—No errors detected

DRV_NOT_INIT—Driver has not been initialized by DRV_init()

DRV_BAD_PARM—Invalid parameter ID

DRV_TRACE_ERR—Error communicating with optional overhead trace memory

4.3.3.2 DRV_trace_get

Description: This function retrieves the actual receive, expected receive, and the transmit trace messages.

Prototype:
```
DRV_RETURN DRV_trace_get
  (DRV_DEVICE     *dev,
   DRV_PARM_ID    id,
   DRV_TRACE      dest);
```

Parameters: **dev**—pointer used to uniquely associate an instance of driver software with a specific device.

id—Identifier of overhead trace parameter to be retrieved

dest—Array to place the retrieved overhead trace message

Return Codes: **DRV_SUCCESS**—No errors detected

DRV_NOT_INIT—Driver has not been initialized by DRV_init()

DRV_BAD_PARM—Invalid parameter ID

DRV_TRACE_ERR—Error communicating with optional the trace memory

4.3.4 Diagnostic Testing (DG)

The DG subsystem performs functional tests on the device and its environment. The DG subsystem executes diagnostic sequences to determine the proper operation of the device. These tests optionally execute during initialization or by the user. These tests will interfere with the normal operation of the device and will introduce errors into the network of connected equipment.

4.3.4.1 DRV_test

Description: This function executes a specified device diagnostic test. Some of these tests will disrupt normal data transmission through the device. Care should be exercised when running these tests. Available test sequences are:

- Verify device configuration with software data structures (background)
- Run device default register values test (post initialization)
- Run device register access tests

Prototype:
```
DRV_RETURN DRV_test(DRV_DEVICE *dev, DRV_PARM_ID test_id);
```

Parameters: **dev**—pointer used to uniquely associate an instance of driver software with a specific device.

test_id—Identifier for desired diagnostic test. See the definitions list for DRV_TEST_CODES enumeration in the header file “drv.h”.

Return Codes: **DRV_SUCCESS**—Diagnostic test passed successfully

DRV_NOT_INIT—Driver has not been initialized by DRV_init()

DRV_BAD_PARM—Invalid test requested

DRV_TEST_FAIL—Diagnostic test failed. An event (see DRV_event()) is generated with a specific error code for each error.

4.3.5 Failure Monitoring Status (FM)

The FM subsystem monitors device operating states. As discussed in Section 4.3.2, the **DRV_{trib_}set()** and **DRV_{trib_}get()** functions are used to access the FM parameters. The user can also access the status parameters with a special function designed to retrieve all the status parameters with a single function call. Events are generated with every state transition if the associated parameter monitoring has been enabled.

4.3.5.1 RS8228_FM_status

Description: This function retrieves the current state of the parameters monitored by the Facility Monitoring subsystem for all physical layers pertaining to the designated device. Parameters that are not being monitored will return **DRV_NOT_MONITORED**.

Prototype:

```
DRV_RETURN RS8228_FM_status
  (DRV_DEVICE          *dev,
   RS8228_FM_STRUCTS   *dest);
```

Parameters: **dev**—pointer used to uniquely associate an instance of driver software with a specific device.

dest —pointer to a FM status data structure that incorporates all relevant physical layer structures and the device specific FM data structure

Return Codes: **DRV_SUCCESS**—No errors detected

DRV_NOT_INIT—DRV-device has not been initialized by DRV_init()

DRV_BAD_PARM—Input parameter invalid

4.3.6 Performance Monitoring (PM) Status

The PM subsystem provides the telecom standards based operating statistics. These performance measures are aggregated over 15 minute and 24 hour intervals. As discussed in Section 4.3.2, the **DRV_set()** and **DRV_get()** functions are used to access the PM parameters. The user may optionally access the PM status parameters with special functions designed to retrieve all PM status parameters related to a specific physical layer with one function call.

4.3.6.1 RS8228_ATM_PM_status

Description: This function retrieves the states (from either the current or previous 15 minute or 24 hour accumulation interval) of the parameters monitored by the PM subsystem.

Prototype:

```
DRV_RETURN RS8228_ATM_PM_status
  (DRV_DEVICE          *dev,
   DRV_PARM_ID         interval,
   DRV_ATM_PM_STATE    *dest,
   DRV_TRIB            trib);
```

Parameters: **device**—pointer used to uniquely associate an instance of driver software with a specific device.

interval—indicates current or past interval and the accumulation period. Currently defined values include:

- DRV_PM_CURRENT—current 15 minute interval
- DRV_PM_PREVIOUS—previous 15 minute interval
- DRV_PM_CURRENT_D—current 24 hour day interval
- DRV_PM_PREVIOUS_D—current 24 hour day interval

dest —pointer to layer specific Performance Monitoring status data structure

trib —tributary number

Return Codes: **DRV_SUCCESS**—No errors detected

DRV_NOT_INIT—DRV-device has not been initialized by DRV_init()

DRV_BAD_PARM—Invalid parameter ID

4.3.6.2 RS8228_ATM_PM_preset

Description: This function allows the user to initialize the PM statistics for the current interval of either the 15 minute or 24 hour accumulation period to arbitrary values. This capability can be useful in context switching applications such as redundancy or protection switching implementations.

Prototype:

```
DRV_RETURN RS8228_ATM_PM_preset
  (DRV_DEVICE          *dev,
   DRV_PARM_ID         interval,
   DRV_ATM_PM_STATE    *src,
   DRV_TRIB            trib);
```

Parameters:

dev—pointer used to uniquely associate an instance of driver software with a specific device.

interval—indicates current or past interval and the accumulation period. Currently defined values include:

- DRV_PM_CURRENT—current 15 minute interval
- DRV_PM_CURRENT_D—current 24 hour day interval

src —pointer to a Performance Monitoring status data structure

trib —tributary number

Return Codes:

DRV_SUCCESS—No errors detected

DRV_NOT_INIT—Driver has not been initialized by DRV_init()

DRV_BAD_PARM—Invalid interval time value

4.3.7 DRV-RS8228 Events

The driver calls the function **USER_event()** when certain conditions occur. The DG subsystem will generate an event whenever there is a change in the result of a background diagnostic. The FM and PM subsystems will generate events for changes to the states of monitored parameters. The user can customize the immediate response to these various events by completing the body of the function **USER_event()**. If no action is taken upon any of the events, this function simply returns immediately.

4.3.7.1 USER_event

Description: This is an application defined function that is called with asynchronous event messages from the driver software. The **DRV_ASYNCH_EVENT_FUNC_PTR** parameter id should be set in the *asynch_event_func_ptr* field of the driver initialization structure.

The DRV_EVENT structure passed to the application contains three fields to convey event information: the event ID, the event parameter value, and the optional pointer to the event specific structure. Whenever possible, an event ID will be a parameter ID or test ID that is also used by the other driver API functions.

The DG subsystem generates events when a diagnostic test fails or when a background audit mismatch occurs. For diagnostic tests, the event ID is the test ID. For a register test, the event parameter is the register offset that caused the failure. For a loopback test, the event parameter is the parameter ID of the FM failure state under test. Background audit tests are performed for the CF, DG, and FM monitor parameters. An asynchronous event is generated when a mismatch between the software data structures and the hardware registers is detected. The event ID passed to the application is the parameter ID that was found to be invalid. The FM monitor audit checks that the interrupt status of the device is properly configured. A single FM monitor parameter can control more than one interrupt status bit, so the event parameter field will also contain the parameter ID of the FM state parameter that had an improper interrupt status configuration.

The FM subsystem will generate an event when an FM state parameter changes state. The event ID contains the corresponding FM state parameter ID and the event parameter contains the current state (DRV_ACTIVE, DRV_INACTIVE, or DRV_NOT_MONITORED).

The PM subsystem will generate an event when an accumulation period has ended and all data from the current period is transferred to the previous period. The event ID contains the DRV_EV_NEW_PM_DATA event code.

Prototype:
```
void USER_event(DRV_DEVICE *device, const DRV_EVENT *ev);
```

Parameters: **dev**—pointer used to uniquely associate an instance of driver software with a specific device.

ev—pointer to the driver event data structure.

Return Codes: None.

5.0 Embedded Monitor

The embedded support for hardware and software debugging sends a stream of ASCII characters to an external terminal interface under control of a simple command line monitor function. This feature, although included with the driver software, is a compile time option and need not be resident on the final system. It is essentially a shell program around the main subsystems of the driver software and uses the services of these subsystems for the raw data controlled and monitored through the debugger. When enabled, the user sends a simple command (such as read address 0 of the device, abbreviated "`r 0`") that results in either a single or multi-line response. The operating environment should provide access to both an input and output port. The interface to the driver is ASCII character strings, with the output using VT100 compatible "escape sequences" for screen highlighting purposes.

The functions provided by the monitor can be separated into two classes: hardware debugging and software debugging.

Hardware Access Debugging	In this mode, the monitor bypasses the functions provided by the main subsystems of the driver and communicates directly with the device. The user has the ability to read and write registers and to display a complete status of the registers. The monitor provides the ability to scan a given register (continue reading a specified register once a second) and to display the register status with the fields continuously updated at every monitor timer interrupt (typically once a second).
Software Access Debugging	In this mode, the monitor displays the parameters (both input and output), of the CF, DG, OH, FM, and PM subsystems. Through this mode, the user can both display and change any of the driver soft parameters.

5.1 Driver Debugger API

The monitor requires the operating environment to provide access to input/output data sources that are typically associated with a computer display terminal. In addition, some of the features of the monitor require a periodic timer interrupt for screen refresh purposes.

5.1.1 DRV_mon

Description: This function is called to control the debugger. It allows the debugger to be initialized, configured, and terminated. The monitor program uses C standard I/O library functions (e.g., fgets, fputs, fprintf) to receive keystrokes and output screen data, so the standard input and standard output file pointers must be configured to properly access the debugger terminal.

Prototype:

```
DRV_RETURN DRV_mon
  (DRV_DEVICE          *dev,
   DRV_PARM_ID         id,
   char                *mon_text);
```

Parameters:

dev—pointer used to uniquely associate an instance of driver software with a specific device.

id—Identifier for the desired monitor/debugger action. Possible values are:

- DRV_MON_INIT—initialize the monitor/debugger
- DRV_MON_USER_DATA—process a user input string
- DRV_MON_TICK—initiates a screen update for repeat mode
- DRV_MON_TERM—terminate the monitor/debugger

mon_text—NULL terminated string used with the following monitor/debugger identifiers:

- DRV_MON_INIT - custom prompt string
- DRV_MON_USER_DATA - command string received from a terminal interface.

Return Codes:

DRV_SUCCESS—No error, monitor program active.

DRV_MON_QUIT—User requested termination of monitor.

DRV_NOT_INIT—The driver software has not been initialized.

5.1.2 Debugger Event Handler

The debugger function, **DRV_mon_event**(), is a custom application function that is provided for printing events sent to the asynchronous event handler. This function will decode the parameters of the driver event structure and display the corresponding parameter names. The **DRV_mon_event**() function has the same function prototype as the application's asynchronous event handler function, described in "DRV-RS8228 Events" on page 3-16, so the **DRV_mon_event**() address can be used to directly set the asynchronous event handler function parameter, DRV_ASYNCH_EVENT_FUNC_PTR.

5.2 User Interface Commands

Each of the following commands are entered from the user's terminal as ASCII character strings terminated with a carriage return or line feed. The commands are shown using lower case characters but the monitor is case insensitive. Most numerical entries use the hex data format except for PM parameters which are in decimal form.

5.2.1 Debugger Control

Upon completion of monitor initialization, the “help” menu is displayed. This is a list of the available commands. The “help” menu can be displayed at any time with the following commands:

```
      Command:    h or?

 H or? Displays help menu
 OH  Displays DRV Overhead Parameters
 MS Displays DRV FM and PM Statistics
 S Set a DRV parameter   ex: s DRV_TIMING_SOURCE DRV_FACILITY_TMG
 G Get a DRV parameter   ex: g DRV_TIMING_SOURCE
 SC Send DRV parameter Configuration
 R Device Read   ex: r 03     read device register 0x03
                      r 03 04 read device registers 0x03 and 0x04
                      r 03:05 read device registers 0x03-0x05
 W Device Write ex: w 03 a5     write 0xa5 to device register 0x03
                      w 03:05 a5 write 0xa5 to device registers
0x03-0x05
 FW Simulator Force Device Write
DMB Display Memory Byte(s)
     ex: dmb b0000 0a display ten bytes starting at 0xb0000
MMB Modify Memory Byte
     ex: mmb b0075 55 write the value 0x55 to the address 0xb0075
 Q Stops execution of monitor
! Repetitive execution (stop by typing <CR>)
     ex:!r 03 Read device register 0x03
RS8228>
```

The monitor has a common repeat command prefix.

```
      Command:    !<monitor_command_mnemonic>
```

The “!” prefix causes a repetitive execution of the monitor command. An example is the repetitive reading of a given register. This register is read once a second continually until the user enters another command. The syntax for the repetitive reading of register 0x12 is “`!r 12`”.

Execution of the monitor can be ceased under user control

```
      Command:    q
```

This command stops the execution of the monitor.

The monitor also has a list of common error messages.

```
      Response:   Improper command syntax.
      Response:   Invalid RS8228 address
```

5.2.2 Device Register Access

The registers within the device can be read and written via the monitor. Single or multiple registers can be accessed with a single command. The addresses and data are in hex format. Repetitive execution of this command is allowed and error messages are output for invalid command entries.

```
Command:    r <addr_1> ... <addr_n>
Response:   r(<addr_1>)=<data_1> ... r(<addr_n>)=<data_n>
```

This form of the "R" command reads 1 to n ($1 \leq n \leq 8$) RS8228 registers. The response lists the data read from each 8228 register.

```
Command:    r <addr_1>:<addr_2>
Response:   r(<addr_1>:<addr_2>)= <data_1>... <data_2>
```

This form of the "R" command reads a sequential bank of 8228 registers. The output displays the data read from the device eight (8) registers at a time.

```
Command:    r
Response:   r(<addr_1>:<addr_2>)= <data_1>... <data_2>
            ...
            r(<addr_y>:<addr_z>)= <data_y> ... <data_z>
```

This form of the "R" command reads all the 8228 registers. The output displays the data read from the device eight (8) registers at a time.

Use of the "R" command to access device registers that contain bits that clear after being read will not affect operation of the driver software because the debugger reads a buffered register value. The buffer containing the values of registers that clear after being read is updated during driver "tick" processing and during device interrupt processing. Interrupts associated with the "clear after read" bits are processed immediately after updating the buffers.

```
Command:    w <addr_1> <data_1>
Response:   w(addr_1)=<data_1>  r(<addr_1)=<data_1>
```

This form of the "W" command writes a single value to a specified 8228 register. The register is then read back and that value is displayed.

```
Command:    w <addr_1>:<addr_2> <data>
Response:   w(<addr_1>:<addr_2>)=<data>
            r(<addr_1>:<addr_2>)= <data_1> ... <data_2>
```

This form of the "W" command writes the same data value to a sequential bank of 8228 registers. These registers are then read back and the values are displayed.

5.2.3 DRV-RS8228 Parameter Access

The control and status parameters of each driver subsystem can be accessed individually through simple "set" and "get" functions.

```
Command:    g <DRV parameter ID> { t=<trib> }
Response:   <DRV parameter ID> = <parameter value> { t=<trib> }
```

The "G" command retrieves a single or multiple driver parameters and displays the current parameter state. An example of this command is "g DRV_TIMING_SOURCE" with the response "DRV_TIMING_SOURCE = DRV_FACILITY_TMG". The syntax on the command is case insensitive and the entire parameter ASCII name is not required to be completely entered. The driver program searches for a match between the command string and one or more of the driver parameters. It will match all the parameters that contain <DRV parameter string> and display the current state. For example, the command "g timing" will produce the same response. Entering no parameter string will display ALL parameters. Repetitive execution of this command is allowed.

If the parameter requires a tributary, it must be supplied as the last token on the command line. The monitor does not perform any tributary calculations, so the supplied trib must be a linear index of the proper value and in the range of 0 through 95.

```
Command:    s <DRV parameter ID> <parameter value> { t=<trib> }
Response:   set <DRV parameter ID> = <parameter value> { t=<trib>
}
```

The "S" command sets a single driver parameter to a particular value. The monitor matches part of the entered parameter ID with one of the driver parameters in the manner described for the "g" command. The parameter value may be a logical variable (such as DRV_ACTIVE) or may a numerical value. For logical variables, the monitor provides a character string match similar to that used for the parameter ID. For numerical values, the entry is performed in hex format except for PM variables which are entered in decimal.

If the parameter requires a tributary, it must be supplied as the last token on the command line. The monitor does not perform any tributary calculations, so the supplied trib must be a linear index of the proper value and in the range of 0 through 95.

```
Command:    sc
Response:   s <DRV parameter ID #1> <parameter value #1>
            ....                ....              ....
            ....                ....              ....
            s <DRV parameter ID #22> <parameter value #22>
```

The "SC" command prints the current state of all driver parameters that contain <DRV parameter string> so that the current system configuration can be saved to an external buffer of a terminal emulation program. Each line of output contains the command to set the parameter to the current state. The buffer containing the desired parameter states can then be sent to the debugger to quickly configure the desired parameters of the system. (Note: When transmitting a buffer containing multiple "set" parameter commands, the transmitting program should wait for a carriage return from the debugger before transmitting the next line of the buffer.)

5.2.4 Random Memory Access

The driver debugger allows generic memory access to aid in debugging the entire system that contains the driver software. The “Display Memory” and “Modify Memory” commands are provided for this purpose. The addresses and byte counts are in hex format. Commands ending in a “b” will access the memory using a byte at a time instead of a word at a time.

(WARNING: Use of these commands to access device registers will bypass all driver software and can affect its integrity because of the loss of active interrupt status bits and the clearing of error counters.)

```
Command:    dm(b) <base_addr> <byte_count>
Response:
<addr>      : 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F ................
<addr> + 10: 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F ................
<addr> + 20: 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F !”#$%&’()*+’-./
```

The “DM(B)” command reads a range of memory addresses from the `<base_addr>` address and displays the current memory value and any corresponding printable ASCII character. The range for `<byte_count>` is 0x00 to 0x100 and up to sixteen memory addresses are printed per line. Repetitive execution of this command is allowed. Error messages are output for invalid command entries.

```
Command:    mm(b) <addr> <value>
Response:   <addr>       : 40        @
```

The “MM(B)” command modifies a single byte of memory at the given address and then displays the new memory value and any corresponding printable ASCII character. Repetitive execution of this command is allowed. Error messages are output for invalid command entries.

6.0 RS8228 Parameters

The RS8228 parameter interface provides a flexible, scalable mapping of generic telecom variables to physical device addresses. There are eight (8) ports in the RS8228 device and our software and this document references a ports as a "trib."

6.1 Overview

6.1.1 Parameter Names and IDs

The parameter names and IDs are defined in enumerations in the header file “drv.h”. In general, the enumerations are grouped along the logical subsystems of Configuration, Diagnostics, Overhead Processing, Failure Monitoring (FM), and Performance Monitoring (PM). The parameter ID is a 32-bit value associated with a parameter name. Each enumeration grouping starts its numbering from a base offset value definition for the logical grouping of parameters. The base offset value definitions are found in the section called “DRV Parameter Offsets” in the file “drv.h” and can be modified if necessary.

6.1.2 Parameter Hardware Mappings

The parameter name/ID enumerations defined in “drv.h” and "drv_atm.h" are not bound to a single physical hardware mapping and the storage for the parameter value is not bound to a specific structure within storage. This mapping information is configured separately for each device. Therefore application code can be written using the generic parameter interface and an alternate hardware implementation can be accommodated by linking in the library containing the parameter mappings for the alternate device.

6.2 RS8228 Driver Parameters

6.2.1 General Parameters

Table 6-1 displays the general parameters for the RS8228. Unless otherwise indicated, all General parameters are Read/Write.

Table 6-1. General Parameters

General Parameter	DRV_ENABLE_INTERRUPTS	
Possible Values	DRV_INACTIVE DRV_ACTIVE	
Default Value	DRV_INACTIVE	
Description	Configures the telecom device to operate either in a “polled” or “interrupt driven” environment.	
General Parameter	DRV_SOFTWARE_REV_MAJOR	
Possible Values	Positive integer	
Default Value	Read from driver	
Description	Major revision level of the software driver.	
General Parameter	DRV_SOFTWARE_REV_MINOR	
Supported PDH Devices	All	
Possible Values	Positive integer	
Default Value	Read from driver	
Description	Minor revision level of the software driver.	
General Parameter	DRV_TIMER_TICK_INTERVAL	
Possible Values	DRV_50_MSEC DRV_100_MSEC DRV_250_MSEC DRV_500_MSEC DRV_1_SEC	
Default Value	DRV_250_MSEC	
Description	Contains the time interval between adjacent **DRV_tick()** function calls.	

6.2.2 Address and Function Pointer Parameters

Table 6-2 displays the address and function pointer parameters for the RS8228. Unless otherwise indicated, all Address/Function parameters are Read/Write.

Table 6-2. Address and Function Pointer Parameters

Pointer Parameter	DRV_ASYNCH_EVENT_FUNC_PTR	
Possible Values	non-NULL address	
Default Value	NULL	
Description	Pointer to the application defined function of Section 4.3.7 that will handle the DRV generated asynchronous events.	
Pointer Parameter	DRV_DEVICE_ADDR	
Possible Values	non-NULL address	
Default Value	NULL	
Description	Base address of the PDH device in the host processor memory.	
Pointer Parameter	DRV_OPTIONAL_PTR	
Possible Values	non-NULL address	
Default Value	NULL	
Description	Pointer to an application defined optional pointer of type “void”.	

6.2.3 Common FM Configuration Parameters

Table 6-3 displays the common FM configuration parameters for the RS8228. These parameters control the overall FM system behavior.

Table 6-3. Common FM Configuration Parameters *(1 of 2)*

FM Parameter	DRV_FM_INTEGRATION_TIME	
Possible Values	DRV_250_MSEC DRV_500_MSEC DRV_1_SEC DRV_2500_MSEC DRV_5_SEC DRV_10_SEC DRV_15_SEC DRV_20_SEC	
Default Value	DRV_2500_MSEC	
Description	Contains the failure integration time, which is the amount of time that a defect must be continually present before a failure is declared.	

***Table 6-3. Common FM Configuration Parameters** (2 of 2)*

FM Parameter	DRV_FM_DECAY_TIME	
Possible Values	DRV_250_MSEC DRV_500_MSEC DRV_1_SEC DRV_2500_MSEC DRV_5_SEC DRV_10_SEC DRV_15_SEC DRV_20_SEC	
Default Value	DRV_10_SEC	
Description	Contains the failure decay time, which is the amount of time that a defect must be continually absent before a failure is cleared.	
FM Parameter	DRV_RESET_FM_STATES	
Possible Values	DRV_INACTIVE DRV_ACTIVE	
Default Value	DRV_INACTIVE	
Description	Reconfigures all FM states to default values.	

6.2.4 RS8228 Configuration (CF) Parameters

Table 6-4 displays the ATM CF parameters for the RS8228.

Table 6-4. ATM CF Parameters

CF Parameter	DRV_ATM_PHY_LAYER_INTERFACE	TRIB
Possible Values	DRV_T1_FORMAT DRV_E1_FORMAT DRV_DS3_FORMAT DRV_E3_FORMAT DRV_J2_FORMAT DRV_GENERAL_PURPOSE DRV_POWER_DOWN	
Default Value	DRV_T1_FORMAT	
Description	Determines the physical layer interface mode. When the device is in the general purpose mode it performs a serial bit search to find byte and cell alignment.	
CF Parameter	DRV_ATM_UTOPIA_PHY_ADDRESS	TRIB
Possible Values	0x00 - 0x1f	
Default Value	Each port is initialized to its zero-based tributary number. (i.e. Trib1 = 0)	
Description	Programs the Multi-PHY device address for each port. Each port must be assigned a unique address before activating the UTOPIA interface.	

Table 6-4. ATM CF Parameters

CF Parameter	DRV_ATM_UTOPIA_2_MODE	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_ACTIVE	
Description	Selects the UTOPIA level 1 or UTOPIA level 2 mode.	
CF Parameter	DRV_ATM_UTOPIA_DISABLE	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_INACTIVE	
Description	When DRV_ACTIVE it disables the UTOPIA interface for corresponding port. The UTOPIA PHY address must be configured before activating the UTOPIA interface.	
CF Parameter	DRV_ATM_UTOPIA_BUS_WIDTH	TRIB
Possible Values	DRV_BYTE DRV_WORD	8-bit mode 16-bit mode
Default Value	DRV_WORD	
Description	Selects the UTOPIA bus width of byte or word.	
CF Parameter	DRV_ATM_UTOPIA2_MODE	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	UTOPIA 1 mode UTOPIA 2 mode
Default Value	DRV_ACTIVE	
Description	Selects the UTOPIA bus width of byte or word.	
CF Parameter	DRV_ATM_UTOPIA_CELL_HANDSHAKE	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	UTOPIA cell handshake mode UTOPIA octet handshake mode
Default Value	DRV_ACTIVE	
Description	Selects the UTOPIA handshaking mode.	
CF Parameter	DRV_ATM_RCV_CLK_INVERT	TRIB
Possible Values	DRV_FALLING_EDGE DRV_RISING_EDGE	
Default Value	DRV_RISING_EDGE	
Description	Controls the polarity of the receive clock.	
CF Parameter	DRV_ATM_XMT_CLK_INVERT	TRIB
Possible Values	DRV_FALLING_EDGE DRV_RISING_EDGE	
Default Value	DRV_RISING_EDGE	
Description	Controls the polarity of the transmit clock.	

Table 6-4. ATM CF Parameters

CF Parameter	DRV_ATM_RCV_ADD_COSET	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_ACTIVE	
Description	Controls whether the ATM coset is processed on the receive ATM stream.	
CF Parameter	DRV_ATM_RCV_DESCRAMBLE_CELLS	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_ACTIVE	
Description	Controls the self-synchronizing descrambling of the received ATM cell payload. The headers are not descrambled.	
CF Parameter	DRV_ATM_CORRECT_ERRORED_CELLS	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_ACTIVE	
Description	Controls the correction of single bit errors in the ATM header.	
CF Parameter	DRV_ATM_UP_VCI_FILTER_PATTERN	TRIB
Possible Values	0x0 - 0xFFFF	
Default Value	0x0	
Description	User-programmable filter applied to VCI field of incoming cells.	
CF Parameter	DRV_ATM_UP_VCI_FILTER_MASK	TRIB
Possible Values	0x0 - 0xFFFF	
Default Value	0x0	
Description	User-programmable mask to alter the behavior of the filter pattern. Any bits set in this parameter become a "don't care" in the comparison with the programmed pattern.	
CF Parameter	DRV_ATM_UP_PT_FILTER_PATTERN	TRIB
Possible Values	0x0 - 0x7	
Default Value	0x0	
Description	User-programmable filter applied to PT field of incoming cells.	
CF Parameter	DRV_ATM_UP_PT_FILTER_MASK	TRIB
Possible Values	0x0 - 0x7	
Default Value	0x0	
Description	User-programmable mask to alter the behavior of the filter pattern. Any bits set in this parameter become a "don't care" in the comparison with the programmed pattern.	

Table 6-4. ATM CF Parameters

CF Parameter	DRV_ATM_UP_CLP_FILTER_PATTERN	TRIB
Possible Values	0x0 - 0x1	
Default Value	0x0	
Description	User-programmable filter applied to CLP field of incoming cells.	
CF Parameter	DRV_ATM_UP_CLP_FILTER_MASK	TRIB
Possible Values	0x0 - 0x1	
Default Value	0x0	
Description	User-programmable mask to alter the behavior of the filter pattern. Any bits set in this parameter become a "don't care" in the comparison with the programmed pattern.	
CF Parameter	DRV_ATM_RCV_CELL_FILTERING	TRIB
Possible Values	DRV_BLOCK DRV_PASS	Cells that match are rejected. Cells that match are accepted.
Default Value	DRV_PASS	
Description	Controls whether the user-programmable pattern and mask are used to select cells or reject cells.	
CF Parameter	DRV_ATM_DETECT_ERRORED_CELLS	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_ACTIVE	
Description	Controls whether HEC checking is performed as part of the cell validation criteria.	
CF Parameter	DRV_ATM_XMT_ADD_COSET	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_ACTIVE	
Description	Controls the addition of the ATM coset to the transmit HEC value.	
CF Parameter	DRV_ATM_XMT_SCRAMBLE_CELLS	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_ACTIVE	
Description	Controls the self-synchronizing scrambling of the transmit ATM cell payload. The header is not scrambled.	
CF Parameter	DRV_ATM_XMT_PARITY_TYPE	TRIB
Possible Values	DRV_EVEN_PARITY DRV_ODD_PARITY	
Default Value	DRV_ODD_PARITY	
Description	Selects the transmit and receive parity mode for the UTOPIA interface.	

Table 6-4. ATM CF Parameters

CF Parameter	DRV_ATM_XMT_FILL	TRIB
Possible Values	0x00 - TxCIAv line asserted if 1 more complete cell can be accepted 0x01 - TxCIAv line asserted only if room for 2 more cells 0x10 - TxCIAv line asserted only if room for at least 3 more cells 0x11 - TxCIAv line asserted only if can accept at least 4 more cells	
Default Value	0x00	
Description	Determines the threshold value for anticipating when the transmit ATM FIFO will be full. For maximum performance when using a standard ATM layer device, do not change the default value.	
CF Parameter	DRV_ATM_XMT_GENERATE_HEC	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_ACTIVE	
Description	Controls whether the transmit ATM HEC value is internally generated or accepted from the UTOPIA interface.	
CF Parameter	DRV_ATM_XMT_IDLE_CLP	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_ACTIVE	
Description	The value of the CLP (cell loss priority) inserted into an idle/unassigned cell header.	
CF Parameter	DRV_ATM_XMT_CLP	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_INACTIVE	
Description	The value of the CLP (cell loss priority) inserted into transmit cell header when CLP header insertion is enabled - DRV_ATM_INSERT_XMT_CLP.	
CF Parameter	DRV_ATM_XMT_IDLE_PAYLOAD	TRIB
Possible Values	0x0 - 0xff	
Default Value	0x6a	
Description	The value of the payload bytes inserted into an idle cell.	
CF Parameter	DRV_ATM_XMT_IDLE_PT	TRIB
Possible Values	0x0 - 0x7	
Default Value	0x0	
Description	The value of the PT (payload type) inserted into an idle cell header.	

Table 6-4. ATM CF Parameters

CF Parameter	DRV_ATM_XMT_PT	TRIB
Possible Values	0x0 - 0x7	
Default Value	0x0	
Description	The value of the PT (payload type) inserted into a cell header when PT header insertion is enabled - DRV_ATM_INSERT_XMT_PT.	
CF Parameter	DRV_ATM_XMT_IDLE_VCI	TRIB
Possible Values	0x0 - 0xffff	
Default Value	0x0	
Description	The value of the VCI inserted into an idle/unassigned cell header.	
CF Parameter	DRV_ATM_XMT_VCI	TRIB
Possible Values	0x0 - 0xffff	
Default Value	0x0	
Description	The value of the VCI inserted into cells when VCI in the header is enabled - DRV_ATM_INSERT_XMT_VCI.	
CF Parameter	DRV_ATM_IU_VCI_FILTER_PATTERN	TRIB
Possible Values	0x0 - 0xFFFF	
Default Value	0x0	
Description	Idle filter applied to VCI field of incoming cells.	
CF Parameter	DRV_ATM_IU_VCI_FILTER_MASK	TRIB
Possible Values	0x0 - 0xFFFF	
Default Value	0x0	
Description	Idle mask to alter the behavior of the filter pattern. Any bits set in this parameter become a "don't care" in the comparison with the programmed pattern.	
CF Parameter	DRV_ATM_IU_PT_FILTER_PATTERN	TRIB
Possible Values	0x0 - 0x7	
Default Value	0x0	
Description	Idle filter applied to PT field of incoming cells.	
CF Parameter	DRV_ATM_IU_PT_FILTER_MASK	TRIB
Possible Values	0x0 - 0x7	
Default Value	0x0	
Description	Idle mask to alter the behavior of the filter pattern. Any bits set in this parameter become a "don't care" in the comparison with the programmed pattern.	

Table 6-4. ATM CF Parameters

CF Parameter	DRV_ATM_IU_CLP_FILTER_PATTERN	TRIB
Possible Values	0x0 - 0x1	
Default Value	0x0	
Description	Idle filter applied to CLP field of incoming cells.	
CF Parameter	DRV_ATM_IU_CLP_FILTER_MASK	TRIB
Possible Values	0x0 - 0x1	
Default Value	0x0	
Description	Idle mask to alter the behavior of the filter pattern. Any bits set in this parameter become a "don't care" in the comparison with the programmed pattern.	
CF Parameter	DRV_ATM_XMT_SCRAMBLE_HEADER	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_INACTIVE	
Description	Enables distributed sample scrambling (DSS) using the $x^{31} + x^{28} + 1$ polynomial to scramble the entire cell contents except the HEC byte. If both SSS and DSS scrambling is enabled, SSS scrambling overrides.	
CF Parameter	DRV_ATM_RCV_DESCRAMBLE_HEADER	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_INACTIVE	
Description	Enables distributed sample descrambling (DSS) using the $x^{31} + x^{28} + 1$ polynomial to descramble the entire cell contents except the HEC byte. If both SSS and DSS descrambling is enabled, SSS descrambling overrides.	
CF Parameter	DRV_ATM_OUTPUT_PORT1	TRIB
Possible Values	0x0 - 0xF	
Default Value	0x0	
Description	Controls the value to be output on the LStatOut[0:3] pins when set for programmed output.	
CF Parameter	DRV_ATM_STATUS_SELECT	TRIB
Possible Values	0x00 - RcvrHld, HECCorr, HECDet, and LOCD 0x01 - NonMatch, IdleRcvd, CellRcvd, and CellSent 0x02 - RxOvfl, TcOfvl, SOCErr, and ParErr 0x03 - Value of OutStat control register	
Default Value	0x3	
Description	Controls the signals appearing on the LStatOut[0:3] output pins.	

Table 6-4. ATM CF Parameters

CF Parameter	DRV_ATM_HEC_ERROR_PATTERN	TRIB
Possible Values	0x0 - 0xff	
Default Value	0x0	
Description	Controls the insertion of errors in the HEC byte. Bits set in this parameter are XOR'ed with the calculated HEC value to invert the associated bit.	
CF Parameter	DRV_ATM_DISCARD_IDLE_CELLS	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_ACTIVE	
Description	Enables the deletion of cells matching the idle cell header criteria.	
CF Parameter	DRV_ATM_CELL_RECIEVER	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_INACTIVE	
Description	Enables the ATM cell delineator to receive cells.	
CF Parameter	DRV_ATM_PASS_THRU	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_INACTIVE	
Description	When set to DRV_ACTIVE this bit disables Loss of Cell Delination. When set to DRV_INACTIVE this bit enables received ATM cells to be dropped when the cell delineator is not in cell alignment.	
CF Parameter	DRV_ATM_CHIP_SELECT_POLARITY	TRIB, STATIC
Possible Values	DRV_HIGH DRV_LOW	
Default Value	DRV_HIGH	
Description	Controls the polarity of the chip select output polarity.	
CF Parameter	DRV_ATM_XMT_SYNC_POLARITY	TRIB
Possible Values	DRV_HIGH DRV_LOW	
Default Value	DRV_HIGH	
Description	Controls the polarity of the transmit sync input.	
CF Parameter	DRV_ATM_RCV_HOLD_POLARITY	TRIB
Possible Values	DRV_HIGH DRV_LOW	
Default Value	DRV_HIGH	
Description	Controls the polarity of the receive hold input.	

Table 6-4. ATM CF Parameters

CF Parameter	DRV_ATM_RCV_SYNC_POLARITY	TRIB
Possible Values	DRV_HIGH DRV_LOW	
Default Value	DRV_HIGH	
Description	Controls the polarity of the receive sync input.	
CF Parameter	DRV_ATM_INSERT_XMT_CLP	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_INACTIVE	
Description	Enables the CLP field in the outgoing ATM header to be filled in with the value in the header field register.	
CF Parameter	DRV_ATM_INSERT_XMT_PT	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_INACTIVE	
Description	Enables the PT field in the outgoing ATM header to be filled in with the value in the header field register.	
CF Parameter	DRV_ATM_INSERT_XMT_VCI	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_INACTIVE	
Description	Enables the VCI field in the outgoing ATM header to be filled in with the value in the header field register.	
CF Parameter	DRV_ATM_INSERT_XMT_VPI	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_INACTIVE	
Description	Enables the VPI field in the outgoing ATM header to be filled in with the value in the header field register.	
CF Parameter	DRV_ATM_INSERT_XMT_GFC	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_INACTIVE	
Description	Enables the GFC field in the outgoing ATM header to be filled in with the value in the header field register.	

Table 6-4. ATM CF Parameters

CF Parameter	DRV_ONESEC_MODE	
Possible Values	DRV_NO_LATCH DRV_INPUT_LATCH	
Default Value	DRV_NO_LATCH	
Description	Enables one-second counter latching.	

6.2.5 ATM UNI Mode CF Parameters

Table 6-5. ATM UNI Mode CF Parameters

CF Parameter	DRV_ATM_UNI_MODE	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_ACTIVE	
Description	When parameter is set to DRV_ACTIVE the UNI interface is utilized when set to DRV_INACTIVE the NNI interface is utilized.	
CF Parameter	DRV_ATM_UP_GFC_FILTER_PATTERN	TRIB
Possible Values	0x0 - 0xf	
Default Value	0x0	
Description	User-programmable filter applied to GFC field of incoming cells.	
CF Parameter	DRV_ATM_UP_GFC_FILTER_MASK	TRIB
Possible Values	0x0 - 0xf	
Default Value	0x0	
Description	User-programmable mask to alter the behavior of the filter pattern. Any bits set in this parameter become a "don't care" in the comparison with the programmed pattern.	
CF Parameter	DRV_ATM_XMT_GFC	TRIB
Possible Values	0x0 - 0x0f	
Default Value	0x0	
Description	The value of the GFC transmit header values inserted into the transmit cell header when GFC in the header is enabled - DRV_ATM_INSERT_XMT_GFC.	
CF Parameter	DRV_ATM_XMT_IDLE_GFC	TRIB
Possible Values	0x0 - 0x0f	
Default Value	0x0	
Description	The value of the GFC (generic flow control) tansmit idle inserted into an idle cell header.	
CF Parameter	DRV_ATM_IU_GFC_FILTER_PATTERN	TRIB
Possible Values	0x0 - 0xf	
Default Value	0x0	
Description	Controls the value of the GFC that identifies and idle cell.	

Table 6-5. ATM UNI Mode CF Parameters

CF Parameter	DRV_ATM_IU_GFC_FILTER_MASK	TRIB
Possible Values	0x0 - 0xF	
Default Value	0xFF	
Description	Idle cell mask to alter the behavior of the filter pattern. Any bits set in this parameter become a "don't care" in the comparison with the programmed pattern.	
CF Parameter	DRV_ATM_UP_VPI_FILTER_PATTERN	TRIB
Possible Values	0x00 - 0xff	
Default Value	0x00	
Description	User-programmable filter applied to VPI field of incoming cells.	
CF Parameter	DRV_ATM_UP_VPI_FILTER_MASK	TRIB
Possible Values	0x00 - 0xff	
Default Value	0x00	
Description	User-programmable mask to alter the behavior of the filter pattern. Any bits set in this parameter become a "don't care" in the comparison with the programmed pattern.	
CF Parameter	DRV_ATM_IU_VPI_FILTER_PATTERN	TRIB
Possible Values	0x00 - 0xff	
Default Value	0x00	
Description	Idle cell filter applied to VPI field of incoming cells.	
CF Parameter	DRV_ATM_IU_VPI_FILTER_MASK	TRIB
Possible Values	0x00 - 0xff	
Default Value	0x00	
Description	User-programmable mask to alter the behavior of the filter pattern. Any bits set in this parameter become a "don't care" in the comparison with the programmed pattern.	
CF Parameter	DRV_ATM_XMT_IDLE_VPI	TRIB
Possible Values	0x00 - 0xff	UNI Mode
Default Value	0x00	
Description	The value of the VPI inserted in an idle/unassigned cell header.	
CF Parameter	DRV_ATM_XMT_VPI	TRIB
Possible Values	0x00 - 0xff	UNI Mode
Default Value	0x00	
Description	The value of the VPI inserted into a cell header when VPI header field is enabled - DRV_ATM_INSERT_XMT_VPI.	

Table 6-5. ATM UNI Mode CF Parameters

CF Parameter	DRV_ATM_INSERT_XMT_GFC	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_INACTIVE	
Description	Enables the GFC field in the outgoing ATM header to be filled in with the value in the header field register.	

6.2.6 ATM NNI Mode CF Parameters

Table 6-6. ATM NNI Mode CF Parameters

CF Parameter	DRV_ATM_UP_VPI_FILTER_PATTERN	TRIB
Possible Values	0x00 - 0xff	
Default Value	0x00	
Description	User-programmable filter applied to VPI field of incoming cells.	
CF Parameter	DRV_ATM_UP_VPI_FILTER_MASK	TRIB
Possible Values	0x00 - 0xff	
Default Value	0x00	
Description	User-programmable mask to alter the behavior of the filter pattern. Any bits set in this parameter become a "don't care" in the comparison with the programmed pattern.	
CF Parameter	DRV_ATM_IU_VPI_FILTER_PATTERN	TRIB
Possible Values	0x00 - 0xff	
Default Value	0x00	
Description	Idle cell filter applied to VPI field of incoming cells.	
CF Parameter	DRV_ATM_IU_VPI_FILTER_MASK	TRIB
Possible Values	0x00 - 0xff	
Default Value	0x00	
Description	User-programmable mask to alter the behavior of the filter pattern. Any bits set in this parameter become a "don't care" in the comparison with the programmed pattern.	
CF Parameter	DRV_ATM_XMT_IDLE_VPI	TRIB
Possible Values	0x00 - 0xfff	NNI Mode
Default Value	0x00	
Description	The value of the VPI inserted in an idle/unassigned cell header.	
CF Parameter	DRV_ATM_XMT_VPI	TRIB
Possible Values	0x00 - 0xfff	NNI Mode
Default Value	0x00	
Description	The value of the VPI inserted into a cell header when header insertion is enabled.	

6.2.7 ATM DG Parameters

Table 6-7 displays the ATM DG parameters for the RS8228.

Table 6-7. ATM DG Parameters

DG Parameter	DRV_SOFT_RESET	
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_INACTIVE	
Description	When written to DRV_ACTIVE initiates a reset where internal state machines are reset and all registers except bit 7 in 0x202 assume their default values.	
DG Parameter	DRV_DEVICE_STATE_RESET	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_INACTIVE	
Description	When written to DRV_ACTIVE all internal state machines are reset.	
DG Parameter	DRV_ATM_TRIB_RESET	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_INACTIVE	
Description	When written to DRV_ACTIVE initiates a Port Master Reset.	
DG Parameter	DRV_ATM_TRIB_STATE_RESET	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_INACTIVE	
Description	When written to DRV_ACTIVE initiates a Port Logic Reset.	
DG Parameter	DRV_ATM_TERMINAL_LOOPBACK	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_INACTIVE	
Description	Writing DRV_INACTIVE to this bit enables a terminal loopback (i.e. data to be transmitted out the facility is looped back to the receive facility).	
DG Parameter	DRV_ATM_RCV_FIFO_RESET	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_INACTIVE	
Description	Writing DRV_ACTIVE to this bit empties the Rcv ATM FIFO and then holds it indefinitely in reset, causing all incoming cells to be ignored; 0 returns the FIFO to normal operation.	

Table 6-7. ATM DG Parameters

DG Parameter	DRV_ATM_XMT_FIFO_RESET	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_INACTIVE	
Description	Writing DRV_ACTIVE to this bit empties the Xmt ATM FIFO and then holds it indefinitely in reset, forcing idle cells into all outgoing cell slots; 0 returns the FIFO to normal operation.	
DG Parameter	DRV_ATM_XMT_HEC_ERROR_INSERT	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_INACTIVE	
Description	Writing DRV_ACTIVE causes a single or continuous stream of HEC errors to be inserted into the outgoing cell stream; writing a 0 to this bit readies the framer for a subsequent bit error injection, or, discontinues the error injection, respectively.	
General Parameter	DRV_ATM_DEVICE_ID	TRIB
Possible Values	0 - 15	
Default Value	Value read from device	
Description	The part number that uniquely identifies the RS8228 device	
General Parameter	DRV_ATM_DEVICE_REV	TRIB
Possible Values	ORIGINAL_RELEASE NOT_USED REV_B (8228-12)	
Default Value	Value read from device	
Description	The version number that uniquely identifies the RS8228 device	
General Parameter	DRV__ATM_CELL_FACILITY_LOOPBACK	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_INACTIVE	
Description	Writing a DRV_ACTIVE to this bit enables a facility loopback (i.e. data is transmitted back to the facility).	

6.2.8 ATM FM State Parameters

Table 6-8 displays the general FM state parameters for the RS8228.

Table 6-8. ATM FM State Parameters

FM Parameter	DRV_ATM_CHE	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE DRV_NOT_MONITORED	
Default Value	DRV_NOT_MONITORED	
Controlling Parameter	DRV_MONITOR_ATM_CHE	
Description	Indication for a received correctable ATM HEC error. DRV_ACTIVE indicates that there has been a CHE status change.	
FM Parameter	DRV_ATM_RCV_FIFO_OVERRUN	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE DRV_NOT_MONITORED	
Default Value	DRV_NOT_MONITORED	
Controlling Parameter	DRV_MONITOR_RCV_ATM_FIFO_OVERRUN	
Description	Indication for a receive ATM FIFO overrun error. DRV_ACTIVE indicates that there has been a FIFO overrun status change.	
FM Parameter	DRV_ATM_LOCD	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE DRV_NOT_MONITORED	
Default Value	DRV_NOT_MONITORED	
Controlling Parameter	DRV_MONITOR_ATM_LOCD	
Description	This is a delta indication of the loss of cell delineation defect. DRV_ACTIVE indicates that there has been an LCD status change.	
FM Parameter	DRV_ATM_UHE	
Possible Values	DRV_ACTIVE DRV_INACTIVE DRV_NOT_MONITORED	
Default Value	DRV_NOT_MONITORED	
Controlling Parameter	DRV_MONITOR_ATM_UHE	
Description	This is a delta indication for a received uncorrectable ATM HEC error. DRV_ACTIVE indicates that there has been a UHE status change.	

Table 6-8. ATM FM State Parameters

FM Parameter	DRV_ATM_XMT_FIFO_OVERRUN	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE DRV_NOT_MONITORED	
Default Value	DRV_NOT_MONITORED	
Controlling Parameter	DRV_MONITOR_XMT_ATM_FIFO_OVERRUN	
Description	This is a delta indication for a transmit ATM FIFO overrun error. DRV_ACTIVE indicates that there has been a FIFO overrun status change.	
FM Parameter	DRV_ATM_XMT_PARITY_ERROR	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE DRV_NOT_MONITORED	
Default Value	DRV_NOT_MONITORED	
Controlling Parameter	DRV_MONITOR_ATM_PARITY_ERROR	
Description	This is a delta indication for a transmit parity error. DRV_ACTIVE indicates that there has been a parity error status change.	
FM Parameter	DRV_ATM_TSOC_ERROR	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE DRV_NOT_MONITORED	
Default Value	DRV_NOT_MONITORED	
Description	When DRV_ACTIVE is read it indicates Start of Cell Error received.	
FM Parameter	DRV_ATM_CELL_SENT	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE DRV_NOT_MONITORED	
Default Value	DRV_NOT_MONITORED	
Controlling Parameter	DRV_MONITOR_ATM_CELL_SENT	
Description	When DRV_ACTIVE is read it indicates that a cell has been sent.	
FM Parameter	DRV_ATM_BUS_CNFLCT	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE DRV_NOT_MONITORED	
Default Value	DRV_NOT_MONITORED	
Controlling Parameter	DRV_MONITOR_ATM_BUS_CNFLCT	
Description	When DRV_ACTIVE is read it indicates that a bus conflict occured.	

Table 6-8. ATM FM State Parameters

FM Parameter	DRV_ATM_RCV_HLD	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE DRV_NOT_MONITORED	
Default Value	DRV_NOT_MONITORED	
Controlling Parameter	DRV_MONITOR_ATM_RCV_HLD	
Description	When DRV_ACTIVE is read it indicates that a receiver hold has occured.	
FM Parameter	DRV_ATM_RCV_CELL	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE DRV_NOT_MONITORED	
Default Value	DRV_NOT_MONITORED	
Controlling Parameter	DRV_MONITOR_ATM_RCV_CELL	
Description	When DRV_ACTIVE is read it indicates that a cell has been received.	
FM Parameter	DRV_ATM_XMT_FIFO_OVERRUN	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE DRV_NOT_MONITORED	
Default Value	DRV_NOT_MONITORED	
Controlling Parameter	DRV_MONITOR_XMT_ATM_FIFO_OVERRUN	
Description	When DRV_ACTIVE is read it indicates Transmit overflow condition.	
FM Parameter	DRV_ATM_RCV_FIFO_OVERRUN	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE DRV_NOT_MONITORED	
Default Value	DRV_NOT_MONITORED	
Controlling Parameter	DRV_MONITOR_RCV_ATM_FIFO_OVERRUN	
Description	When DRV_ACTIVE is read it indicates Receive overflow condition.	
FM Parameter	DRV_DEFECT_ATM_LOCD	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE DRV_NOT_MONITORED	
Default Value	DRV_NOT_MONITORED	
Controlling Parameter	DRV_MONITOR_DEFECT_ATM_LOCD	
Description	When DRV_ACTIVE is read it indicates ATM LOCD defects.	

Table 6-8. ATM FM State Parameters

FM Parameter	DRV_ATM_RCV_IDLE_CELL	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE DRV_NOT_MONITORED	
Default Value	DRV_NOT_MONITORED	
Controlling Parameter	DRV_MONITOR_ATM_RCV_IDLE_CELL	
Description	When DRV_ACTIVE is read it indicates that an idle cell has been received.	
FM Parameter	DRV_MONITOR_ATM_RCV_IDLE_CELL	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_INACTIVE	
Description	Controls monitoring of the ATM idle cell received event.	
FM Parameter	DRV_ATM_NON_MATCH	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE DRV_NOT_MONITORED	
Default Value	DRV_NOT_MONITORED	
Controlling Parameter	DRV_MONITOR_ATM_NON_MATCH	
Description	When DRV_ACTIVE is read it indicates that a non-matching cell has been received.	
FM Parameter	DRV_ATM_NON_ZERO_GFC	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE DRV_NOT_MONITORED	
Default Value	DRV_NOT_MONITORED	
Controlling Parameter	DRV_MONITOR_ATM_NON_ZERO_GFC	
Description	When DRV_ACTIVE is read it indicates that a non-zero gfc has been received.	
FM Parameter	DRV_ATM_TSOC_ERROR	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE DRV_NOT_MONITORED	
Default Value	DRV_NOT_MONITORED	
Controlling Parameter	DRV_MONITOR_ATM_TSOC_ERROR	
Description	When DRV_ACTIVE is read it indicates that a start of cell error occured.	

6.2.9 ATM FM Controlling Parameters

Table 6-9 displays the FM controlling parameters for the RS8228.

Table 6-9. FM Controlling Parameters

FM Parameter	DRV_MONITOR_ATM_CHE	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_INACTIVE	
Description	Controls the monitoring of received correctable ATM HEC errors.	
FM Parameter	DRV_MONITOR_ATM_RCV_FIFO_OVERRUN	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_INACTIVE	
Description	Controls the monitoring of ATM FIFO overrun errors.	
FM Parameter	DRV_MONITOR_ATM_LOCD	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_INACTIVE	
Description	Controls the monitoring of LOCD.	
FM Parameter	DRV_MONITOR_ATM_UHE	
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_INACTIVE	
Description	Controls the monitoring of received uncorrectable ATM HEC errors..	
FM Parameter	DRV_MONITOR_ATM_XMT_FIFO_OVERRUN	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_INACTIVE	
Description	Controls the monitoring of the transmit ATM FIFO overrun errors.	
FM Parameter	DRV_MONITOR_ATM_PARITY_ERROR	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_INACTIVE	
Description	Controls monitoring of transmit parity errors.	
FM Parameter	DRV_MONITOR_ATM_TSOC_INPUT	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_INACTIVE	
Description	Controls monitoring of the Start of Cell Errors.	

Table 6-9. FM Controlling Parameters

FM Parameter	DRV_MONITOR_XMT_ATM_FIFO_OVERRUN	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_INACTIVE	
Description	Controls monitoring of the Transmit overflow condition.	
FM Parameter	DRV_MONITOR_ATM_RCV_FIFO_OVERRUN	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_INACTIVE	
Description	Controls monitoring of the Receive overflow condition.	
FM Parameter	DRV_MONITOR_DEFECT_ATM_LOCD	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_INACTIVE	
Description	Controls monitoring of the ATM LOCD defects	
FM Parameter	DRV_MONITOR_ATM_RCV_HLD	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_INACTIVE	
Description	Controls monitoring of the ATM receiver hold event.	
FM Parameter	DRV_MONITOR_ATM_RCV_IDLE_CELL	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_INACTIVE	
Description	Controls monitoring of the ATM idle cell received event.	
FM Parameter	DRV_MONITOR_ATM_RCV_CELL	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_INACTIVE	
Description	Controls monitoring of the ATM cell received event.	
FM Parameter	DRV_MONITOR_ATM_NON_ZERO_GFC	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_INACTIVE	
Description	Controls monitoring of the ATM non-zero gfc event.	

Table 6-9. FM Controlling Parameters

FM Parameter	DRV_MONITOR_ATM_CELL_SENT	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_INACTIVE	
Description	Controls monitoring of the cell sent event.	
FM Parameter	DRV_MONITOR_ATM_BUS_CNFLCT	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_INACTIVE	
Description	Controls monitoring of the bus conflict event.	
FM Parameter	DRV_MONITOR_ATM_TSOC_ERROR	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_INACTIVE	
Description	Controls monitoring of the start of cell error.	

6.2.10 ATM PM Count Parameters

Table 6-10 displays the ATM PM count parameters for the RS8228.

Table 6-10. ATM PM Count Parameters

PM Parameter	DRV_CURR_ATM_UHE DRV_PREV_ATM_UHE DRV_CURR_D_ATM_UHE DRV_PREV_D_ATM_UHE	TRIB
Possible Values	0x0 - 0x1F	
Default Value	NONE	
Description	Count of uncorrectable header errors.	
PM Parameter	DRV_CURR_ATM_CHE DRV_PREV_ATM_CHE DRV_CURR_D_ATM_CHE DRV_PREV_D_ATM_CHE	TRIB
Possible Values	0x0 - 0x1F	
Default Value	NONE	
Description	Count of corrected header errors.	
PM Parameter	DRV_CURR_ATM_RCV_CELLS DRV_PREV_ATM_RCV_CELLS DRV_CURR_D_ATM_RCV_CELLS DRV_PREV_D_ATM_RCV_CELLS	TRIB
Possible Values	0x0 - 0x1F	
Default Value	NONE	
Description	Count of cells written to RCV ATM FIFO.	
PM Parameter	DRV_CURR_ATM_XMT_CELLS DRV_PREV_ATM_XMT_CELLS DRV_CURR_D_ATM_XMT_CELLS DRV_PREV_D_ATM_XMT_CELLS	TRIB
Possible Values	0x0 - 0x1F	
Default Value	NONE	
Description	Count of cells pulled from Xmt ATM FIFO.	
PM Parameter	DRV_CURR_ATM_LOCD DRV_PREV_ATM_LOCD DRV_CURR_D_ATM_LOCD DRV_PREV_D_ATM_LOCD	TRIB
Possible Values	0x0 - 0x1F	
Default Value	NONE	
Description	Count of loss of cell delineation event.	

Table 6-10. ATM PM Count Parameters

PM Parameter	DRV_CURR_ATM_NON_MATCHING DRV_PREV_ATM_NON_MATCHING DRV_CURR_D_ATM_NON_MATCHING DRV_PREV_D_ATM_NON_MATCHING	TRIB
Possible Values	0x0 - 0x1F	
Default Value	NONE	
Description	Count of loss of cell delineation event.	
PM Parameter	DRV_CURR_CELL_SENT DRV_PREV_CELL_SENT DRV_CURR_D_CELL_SENT DRV_PREV_D_CELL_SENT	TRIB
Possible Values	0x0 - 0x1F	
Default Value	NONE	
Description	Count of non-idle cells.	

6.2.11 ATM PM Parameters

Table 6-11 displays the ATM PM parameters for the RS8228.

Table 6-11. ATM PM Parameters

PM Parameter	DRV_CURR_TCV_ATM_UHE DRV_CURR_D_TCV_ATM_UHE	TRIB
Possible Values	0x0 - 0x1F	
Default Value	0	
Description	Value used to determine if count of uncorrected header error threshold has been violated.	
PM Parameter	DRV_CURR_TCV_ATM_CHE DRV_CURR_D_TCV_ATM_CHE	TRIB
Possible Values	0x0 - 0x1F	
Default Value	0	
Description	Value used to determine if count of corrected header error threshold has been violated.	
PM Parameter	DRV_CURR_TCV_ATM_RCV_CELLS DRV_CURR_D_TCV_ATM_RCV_CELLS	TRIB
Possible Values	0x0 - 0x1F	
Default Value	0	
Description	Value used to determine if RCV ATM Cell threshold has been violated.	

Table 6-11. ATM PM Parameters

PM Parameter	DRV_CURR_TCV_ATM_XMT_CELLS DRV_CURR_D_TCV_ATM_XMT_CELLS	TRIB
Possible Values	0x0 - 0x1F	
Default Value	0	
Description	Value used to determine if XMT ATM Cell threshold has been violated.	
PM Parameter	DRV_CURR_TCV_ATM_LOCD DRV_CURR_D_TCV_ATM_LOCD	TRIB
Possible Values	0x0 - 0x1F	
Default Value	0	
Description	Value used to determine if the ATM LOCD threshold has been violated.	
PM Parameter	DRV_CURR_TCV_ATM_NON_MATCHING DRV_CURR_D_TCV_ATM_NON_MATCHING	TRIB
Possible Values	0x0 - 0x1F	
Default Value	0	
Description	Value used to determine if the ATM NON-MATCHING threshold has been violated.	
PM Parameter	DRV_CURR_TCV_CELL_SENT DRV_CURR_D_TCV_CELL_SENT	TRIB
Possible Values	0x0 - 0x1F	
Default Value	0	
Description	Value used to determine if Cell Sent threshold has been violated.	

6.2.12 ATM PM Threshold Crossing Alert (TCA) Parameters

Table 6-12 displays the ATM PM TCA parameters for the RS8228.

Table 6-12. ATM PM TCA Parameters

PM Parameter	DRV_CURR_TCA_ATM_UHE DRV_PREV_TCA_ATM_UHE DRV_CURR_D_TCA_ATM_UHE DRV_PREV_D_TCA_ATM_UHE	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_INACTIVE	
Description	Indicates uncorrectable header error threshold has been exceeded.	

Table 6-12. ATM PM TCA Parameters

PM Parameter	DRV_CURR_TCA_ATM_CHE DRV_PREV_TCA_ATM_CHE DRV_CURR_D_TCA_ATM_CHE DRV_PREV_D_TCA_ATM_CHE	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_INACTIVE	
Description	Indicates correctable header error threshold has been exceeded.	
PM Parameter	DRV_CURR_TCA_ATM_RCV_CELLS DRV_PREV_TCA_ATM_RCV_CELLS DRV_CURR_D_TCA_ATM_RCV_CELLS DRV_PREV_D_TCA_ATM_RCV_CELLS	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_INACTIVE	
Description	Indicates RCV ATM FIFO threshold has been exceeded.	
PM Parameter	DRV_CURR_TCA_ATM_XMT_CELLS DRV_PREV_TCA_ATM_XMT_CELLS DRV_CURR_D_TCA_ATM_XMT_CELLS DRV_PREV_D_TCA_ATM_XMT_CELLS	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_INACTIVE	
Description	Indicates XMT ATM FIFO threshold has been exceeded.	
PM Parameter	DRV_CURR_TCA_ATM_LOCD DRV_PREV_TCA_ATM_LOCD DRV_CURR_D_TCA_ATM_LOCD DRV_PREV_D_TCA_ATM_LOCD	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_INACTIVE	
Description	Indicates ATM LOCD threshold has been exceeded.	
PM Parameter	DRV_CURR_TCA_ATM_NON_MATCHING DRV_PREV_TCA_ATM_NON_MATCHING DRV_CURR_D_TCA_ATM_NON_MATCHING DRV_PREV_D_TCA_ATM_NON_MATCHING	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_INACTIVE	
Description	Indicates ATM NON-MATCHING threshold has been exceeded.	

Table 6-12. ATM PM TCA Parameters

PM Parameter	DRV_CURR_TCA_CELL_SENT DRV_PREV_TCA_CELL_SENT DRV_CURR_D_TCA_CELL_SENT DRV_PREV_D_TCA_CELL_SENT	TRIB
Possible Values	DRV_ACTIVE DRV_INACTIVE	
Default Value	DRV_INACTIVE	
Description	Indicates Cell Sent threshold has been exceeded.	

7.0 Installation and Initialization

7.1 Distribution File Structure

All C language source code files have a ".c" extension, all header files have a ".h" extension. The files are partitioned into a directories that contain the generic driver files, the telecom device specific files, a sample application file that initializes and starts the software simulator, and the manual pages for the API functions. The directory structure and files for the common distribution are as follows:

drv/

makefile - makefile for the generic driver function library

header files

drv.h - constant definitions and structure definitions needed by application files
drv_defs.h - internal structures and definitions
drv_mon.h - monitor constant definitions and structure definitions
drv_parm.h - constants and macros used to define and build parameter definitions
drv_stor.h - internal structure definitions
drv_sync.h - constants and macros specific to synchronous interfaces
drv_sys.h - system/compiler dependent definitions and configurable conditional compile options
drv_atm.h - DRV ATM constant definitions and structure definitions
drv_ret.h - return values for DRV code

source files

drv_atm.c - ATM performance monitoring functions
drv_conf.c - parameter configurations
drv_fm.c - failure monitoring processing functions
drv_infc.c - application programming interface functions
drv_parm.c - parameter processing functions
drv_pm.c - performance monitoring processing functions
drv_test.c - testing functions
drv_util.c - utility functions
mon_infc.c - monitor interface functions
mon_io.c - monitor telecom device I/O functions
mon_parm.c - monitor parameter access functions
mon_prt.c - monitor printing functions
mon_scr.c - monitor screen objects
mon_tabl.c - monitor look-up tables
solar_io.c - I/O library for using the simulator with Solaris (tested with Solaris 2.4).

soft_sim/

makefile - makefile for the software simulator application

source files

sim_main.c - main function of a software simulator for the driver
example.mqx - example start-up code for the monitor using the MQX real-time OS

The directory structure and files for the RS8228 device specific distribution are as follows:

rs8228/

makefile - makefile for the RS8228 specific library

header files

rs8228.h - RS8228 device structure and constant definitions needed by application files
rs8228_h.h - RS8228 device register offset and hardware mapping definitions
rs8228_s.h - RS8228 device internal software structure definitions

source files

rs8228_conf.c - RS8228 device parameter configurations
rs8228_devc.c - RS8228 device configuration structures
rs8228_infc.c - RS8228 device specific interface functions
rs8228_parm.c - RS8228 device specific parameter access functions
rs8228s_sim.c - RS8228 device software simulator functions

7.2 Porting Guide

The driver software is written in standard ANSI C. The source files and header files must be compiled and linked with the user application files. The standard input and standard output file pointers and the “stdio.h” header to be included for use by the driver monitor are also defined in **drv.h**. The application code must include the header file **rs8228.h** for definitions of the driver functions, structures, and parameters.

7.3 Data Types

The top of the **drv.h** header file contains all the typedefs that must be configured so that size of the driver fields and variables are matched to the appropriate compiler data types. Table 7-1 summarizes the data types.

Table 7-1. Driver Date Type Sizes

Data Type Definition	# of 8 bit bytes	sign
DRV_REG	2	unsigned
DRV_REG_DATA	1	unsigned
DRV_RETURN	4	signed
DRV_PARM_ID	4	unsigned
DRV_PARM	larger of 4 or sizeof(ptrdiff_t)	unsigned
DRV_PARM_TYPE	1	unsigned
DRV_FLAG	2	unsigned
DRV_BIT_MAP	4	unsigned
DRV_TRIB	2	unsigned
DRV_SCR_INDEX	1	unsigned
DRV_TRACE_TYPE	1	unsigned
DRV_TRACE_CHKSUM	2	unsigned
DRV_TRACE_CRC7	1	unsigned

7.4 Software Simulator

Provided with the driver software distribution are example makefiles for building the driver software and code that provides a method to simulate the telecom device. This allows the driver software to be built and tested on a workstation host or a target host that does not have the device. The preprocessor token SOFT_SIM should be defined during compilation when building the driver software simulator.

The software simulator uses an array in memory to simulate the device. The base address of the array used as the base address of the device. A special driver monitor function, “fw” (force write), can be used to manipulate bits that are defined as read-only in the physical device. Interrupt status bits can be set in the simulated device to generate asynchronous events. The software simulator will automatically clear any bits that normally clear after a read access.

Although loopback tests cannot be simulated, the simulator allows event processing software to be tested independently of the hardware.

7.5 Compilation Options

Various preprocessor directives are outlined in drv_sys.h; see comments there.

7.5.1 Rcv FIFO Overflow Workaround

The Rcv FIFO Overflow workaround is enabled by default via the `RS8228_RCV_FIFO_OVRFLO_PATCH` definition in drv_sys.h; see comments there.

7.6 Driver Initialization

The driver software is initialized by configuring the DRV_INIT structure with the application specific data and calling the **DRV_init**() function. The DRV_INIT structure is used to pass any special parameters needed by the driver software during initialization. Before setting your application specific values you must initialize the DRV_INIT structure by calling the **DRV_init_default()** function, to set all fields in the structure to default values.

7.6.1 DRV_INIT fields

An application can customize the driver initialization process to some extent via the DRV_INIT{} data structure. See drv_defs.h for its definition.

7.6.2 Example Start-Up Code

Below is an example start-up sequence for the driver software. In a multi-threaded environment, it is desirable to have the DRV_tick() function called from a high priority task that is waiting for an event that is scheduled at the timer tick rate. If a monitor is running, it can be scheduled by a lower priority task for its periodic tick.

```
#include “rs8228.h”

DRV_DEVICE drv_dev;
RS8228_DEVICE RS8228_device;

extern void USER_event(DRV_DEVICE *, DRV_EVENT *);
extern void USER_int_enable(DRV_DEVICE *);
extern void USER_int_disable(DRV_DEVICE *);

void main()
{
DRV_INIT    drv_init;

    /*
     * The driver software will set all fields to default values.
     */
    DRV_init_default(&drv_dev, &drv_init);

   /*
    * Configure the RS8228 address and other init. parms for
```

```
    * this application.
    */
    drv_init.addr = (DRV_REG_DATA *)(0x0b0000);
    drv_init.dt   = &RS8228_dev_tables;
    drv_init.dc   = &RS8228_dev_config;
    drv_init.ds   = &RS8228_device;

    drv_init.mon_prompt            = “RS8228”;

    drv_init.asynch_event_func_ptr  = (DRV_PARM)USER_event;
    drv_init.intr_disable_func_ptr  = (DRV_PARM)USER_int_disable;
    drv_init.intr_enable_func_ptr   = (DRV_PARM)USER_int_enable;

    drv_init.destructive_reg_test   = DRV_ACTIVE;
    drv_init.format                 = DRV_SONET;
    drv_init.default_reg_test       = DRV_ACTIVE;

    if ( DRV_init(&drv_dev, &drv_init) != DRV_SUCCESS ) {
        exit(-1);
    }

    /*
     * Initialize the monitor and start the other tasks
     */
    DRV_mon(&drv_dev, DRV_MON_INIT, “MON> ");

      _Create( DRV Monitor Task );
      _Create( DRV Tick Task );     /* highest priority */
      _Create( DRV Time Task );

    while (1) {
        /*
         * Send character strings to the monitor
         */
        DRV_mon(&drv_dev, DRV_MON_USER_DATA, gets() );
    }
}

void mon_tick( )         /* DRV Monitor Task */
{
    while(1) {
        /*
         * Give the monitor an event about every second
         */
        DRV_mon(&drv_dev, DRV_MON_TICK, NULL);
        Timeout(1 SECOND);
    }
}

void drv_tick( unsigned long parm )       /* DRV Tick Task */
{
    while (1) {
         /*
          * This task should be waiting for the tick event before
```

```
             * it arrives. If not, timing for Performance Monitoring
             * and Failure Monitoring will be skewed. The User event
             * handler function will be called from DRV_tick().
             */
            Receive_message( from DRV Time Task );
            DRV_tick(&drv_dev);
        }
    }

    void drv_time( )    /* DRV Time Task */
    {
        while (1) {
            /*
             * Schedule the DRV_tick() call so it runs at a precise
             * rate. Since DRV_tick() takes a varying amount of time
             * to complete, it should not be called in the same loop
             * that blocks for the timer tick length. FM and PM need a
             * very precise tick.
             */
            _Send_message( to DRV Tick Task );
            Timeout(250 MSEC);
        }
    }
```

7.7 Application Discussion

7.7.1 Compatibility with Future Releases

The DRV_INIT structure should be initialized by calling the function **DRV_init_default()**, which will set all fields in the structure to default values. If a future release of the driver software includes a new field in the DRV_INIT structure, it will be defaulted to a value that is compatible with older versions of the driver software. This allows the application code to immediately link in an updated driver version without having to immediately add any code to take advantage of new, additional features.

Application code should only call driver functions defined in Table 4-1 in order to maintain compatibility with future releases of the driver software. In addition, only definitions and structures defined in the files “drv.h” and the device specific header “rs8228.h” should be used. The use of internal driver functions, macros, and definitions is not supported. These functions are subject to changes to facilitate performance and feature improvements. Although a pointer to the DRV_DEVICE structure is passed to the driver functions, its fields should not be directly accessed by application code.

7.7.2 Use of the device Interrupt Signal

Depending on the application of the telecom device or the programming style employed, the use of the device interrupt signal is optional. The features provided by the driver software can be implemented entirely with the periodic timer function **DRV_tick()** called every **T** milliseconds. If the device interrupt line is used, the required processing at each timer interrupt will be minimized. This is at a cost of increased processing for every change in state of an underlying defect.

7.7.3 Use of the DRV Event Generation Functions

Depending on the desired operating behavior of the driver software, use of the driver’s event generation features is optional. If the environment is configured as a synchronous environment, the user will let the OH, FM, and PM subsystems quietly go about their jobs compiling states and statistics. Periodically, the user will retrieve the current information for further processing by the user’s application software. It is the user’s responsibility to maintain any necessary synchronization with the FM integration/decay times and the PM accumulation periods.

If the user prefers to operate in an asynchronous environment, the driver software can be set to generate function calls upon changes in received overhead values, failures in Diagnostics, changes in FM states, and the end of PM periods. Because the actions taken upon these events are application dependent, the user has the responsibility of writing these event handling routines.

7.7.4 Multiple Device Support

Multiple devices in a single target load can be supported by the driver software. For each device, there must be a unique instance of the DRV_DEVICE structure and the RS8228_DEVICE structure along with the unique base address of the device. The RS8228_DEV_CONFIG structure is a constant, look-up table and all instances of a particular device driver type point to the same structure.

Depending on the application programming needs, the user defined functions can also be shared by the multiple driver supported devices. All functions are passed the address of the DRV_DEVICE structure and this is used to uniquely identify the individual devices. An optional, user-defined parameter is also contained in the DRV_DEVICE structure and could be used to contain a pointer to any pre-defined hardware mappings needed by the user defined functions.

7.7.5 Multi-Threaded Access in a Multi-Tasking Environment

The driver software caches reads from all device registers that have bits that clear after being read. For this reason, multi-threaded reads are permissible, although there is still the possibility of a higher priority task preempting a lower priority service call. Access to the internal overhead trace buffers should not be multi-threaded. The task that calls DRV_tick() should be the highest priority task that accesses the driver software.

Index

D

I

R

Further Information:
literature@conexant.com
1-800-854-8099 (North America)
33-14-906-3980 (International)

Web Site
www.conexant.com

World Headquarters
Conexant Systems, Inc.
4311 Jamboree Road,
P.O. Box C
Newport Beach, CA 92658-8902
Phone: (949) 483-4600
Fax: (949) 483-6375

U.S. Florida/South America
Phone: (727) 799-8406
Fax: (727) 799-8306

U.S. Los Angeles
Phone: (805) 376-0559
Fax: (805) 376-8180

U.S. Mid-Atlantic
Phone: (215) 244-6784
Fax: (215) 244-9292

U.S. North Central
Phone: (630) 773-3454
Fax: (630) 773-3907

U.S. Northeast
Phone: (978) 692-7660
Fax: (978) 692-8185

U.S. Northwest/Pacific West
Phone: (408) 249-9696
Fax: (408) 249-7113

U.S. South Central
Phone: (972) 733-0723
Fax: (972) 407-0639

U.S. Southeast
Phone: (919) 858-9110
Fax: (919) 858-8669

U.S. Southwest
Phone: (949) 483-9119
Fax: (949) 483-9090

APAC Headquarters
Conexant Systems Singapore,
Pte. Ltd.
1 Kim Seng Promenade
Great World City
#09-01 East Tower
Singapore 237994
Phone: (65) 737 7355
Fax: (65) 737 9077

Australia
Phone: (61 2) 9869 4088
Fax: (61 2) 9869 4077

China
Phone: (86 2) 6361 2515
Fax: (86 2) 6361 2516

Hong Kong
Phone: (852) 2 827 0181
Fax: (852) 2 827 6488

India
Phone: (91 11) 692 4780
Fax: (91 11) 692 4712

Korea
Phone: (82 2) 565 2880
Fax: (82 2) 565 1440

Europe Headquarters
Conexant Systems France
Les Taissounieres B1
1681 Route des Dolines
BP 283
06905 Sophia Antipolis Cedex
France
Phone: (33 4) 93 00 33 35
Fax: (33 4) 93 00 33 03

Europe Central
Phone: (49 89) 829 1320
Fax: (49 89) 834 2734

Europe Mediterranean
Phone: (39 02) 9317 9911
Fax: (39 02) 9317 9913

Europe North
Phone: (44 1344) 486 444
Fax: (44 1344) 486 555

Europe South
Phone: (33 1) 41 44 36 50
Fax: (33 1) 41 44 36 90

Middle East Headquarters
Conexant Systems
Commercial (Israel) Ltd.
P.O. Box 12660
Herzlia 46733, Israel
Phone: (972 9) 952 4064
Fax: (972 9) 951 3924

Japan Headquarters
Conexant Systems Japan Co., Ltd.
Shimomoto Building
1-46-3 Hatsudai,
Shibuya-ku, Tokyo
151-0061 Japan
Phone: (81 3) 5371 1567
Fax: (81 3) 5371 1501

Taiwan Headquarters
Conexant Systems, Taiwan Co., Ltd.
Room 2808
International Trade Building
333 Keelung Road, Section 1
Taipei 110, Taiwan, ROC
Phone: (886 2) 2720 0282
Fax: (886 2) 2757 6760