MT90503 API User Guide

Part Number: MT90503

Revision Number: **2.4**

Issue Date: **August 2004**

ZARLINK™ SEMICONDUCTOR

PURCHASE OF THIS PRODUCT DOES NOT GRANT THE PURCHASER ANY RIGHTS UNDER PATENT NO. 5,260,978. USE OF THIS PRODUCT OR ITS RE-SALE AS A COMPONENT OF ANOTHER PRODUCT MAY REQUIRE A LICENSE UNDER THE PATENT WHICH IS AVAILABLE FROM TELCORDIA TECHNOLOGIES, INC., 445 SOUTH STREET, MORRISTOWN, NEW JERSEY 07960.

ZARLINK ASSUMES NO RESPONSIBILITY OR LIABILITY THAT MAY RESULT FROM ITS CUSTOMERS' USE OF ZARLINK PRODUCTS WITH RESPECT TO THIS PATENT. IN PARTICULAR, ZARLINK'S PATENT INDEMNITY IN ITS TERMS AND CONDITIONS OF SALES WHICH ARE SET OUT IN ITS SALES ACKNOWLEDGEMENTS AND INVOICES DOES NOT APPLY TO THIS PATENT.

Table of Contents

Table of Contents

Table of Contents

1.0 Overview

This document defines the C-language application programming interface functions.[1]

The library was compiled using Microsoft Visual C++ 6.0.

1.1 Definitions

Channel A 64 kb TDM stream.

LUT Look Up Table. The table used on each UTOPIA port to perform routing and/or translation of the VPI, VCI of a cell header.

RX The receive direction with respect to the UTOPIA bus. Thus, RX means out of the chip when referring to the TDM bus and into the chip when referring to the UTOPIA bus.

TX The transmit direction with respect to the UTOPIA bus. Thus, TX means into the chip when referring to the TDM bus and out of the chip when referring to the UTOPIA bus.

TSST A TDM time-slot stream.

VC A virtual circuit.

1.2 Documentation and Coding Conventions

- In this document:
- all addresses are byte addresses.
- numbers are decimal unless otherwise specified.
- a word is 16 bits, and a byte 8 bits.
- all memory locations are laid-out in the little endian format.
- when a parameter value is greater than 32 bits it is stored in an array where the lowest indexed element contains the LSB.

All function parameters are passed in C structures to allow for compatibility of code upgrades. Each parameter is documented here with 3 fields:

Direction – indicates if the parameter is an input (IN), output (OUT), or input and output (IO) of the function. When a parameter is a pointer the direction is indicated as direction/direction, where the first direction refers to the pointer itself (typically IN) and the second direction (after the slash) refers to the memory pointed to by the pointer. Thus, a pointer direction of IN/OUT indicates that the pointer is an input to the function (i.e. the value of the pointer will not be modified), and the memory pointed to by the pointer is used for output.

Type – indicates the C type of the parameter. A ULONG is 32 bit value. Parameters may also be declared as arrays and are documented here as ULONG[*x*] where *x* indicates the number of elements. Also used in the API are unsigned characters (8-bit values) indicated as BYTEs. As with ULONGs, parameters may also be declared as arrays and are documented here as BYTE[*x*] where *x* indicates the number of elements.

Default – indicates the default value the parameter is initialized to by an associated function for initializing the structure. A value of UNDEFINED means that the _def function will initialize the parameter to a value for that parameter which indicates undefined. The API will return an error if the parameter remains undefined when the structure is passed to the associated function.

1. The MT90503 API software was developed with the assistance of OCTASIC Inc.

Every function has an associated "_def" or default version that initializes the parameter structure. Even if the function requires no inputs there is a _def version. If the _def function is always used to initialize parameter structures, future versions of the API can be backward compatible with older user code as any new feature parameters can be initialized properly.

1.2.1 Code Header Files

The code of the API is split into three compilable entities: API, APIISR, and APIMI (these blocks are described later in **Section "1.5 System Architecture"**). Because the code is in separate entities, each entity has its respective. H file for the functions exported by that entity. These files are needed by the user application to call the functions. The files are listed below, as well as their relation to the code entities:

- API => mt90503_api.h
- APIISR => mt90503_apiisr.h
- APIMI => mt90503_apimi.h

Also, as explained later in this document, the user must supply C code to the API. The user code provides the link between the API and APIISR entities, and allows the three entities to perform read and write accesses to the chip. These functions are described in **Section "3.0 User Supplied Function Descriptions"**. The definitions of the structures needed by all user-supplied functions are contained in mt90503_apiud.h. The file is needed by the user-supplied functions for the definitions of the structures used.

1.3 API Function Summary

Initialization Functions

mt90503_open	Performs all required operations to initialize the chip.
mt90503_open_instance_size	Returns the required size of the instance structure.
mt90503_close	Performs all necessary clean-up to cease using the chip.
mt90503_get_hw_revision	Returns the device revision number.

ATM Functions

mt90503_open_cbr_vc	Opens a bi-directional CBR VC (from TDM bus to UTOPIA bus and vice-versa).
mt90503_open_data_vc	Opens a unidirectional data VC (from UTOPIA bus to data cell FIFO or UTOPIA bus).
mt90503_close_data_vc	Closes a unidirectional data VC.

TDM Functions

mt90503_open_channel_in_vc	Adds a bidirectional channel to an open CBR VC (64 kbs in TX and in RX).
mt90503_open_channel_in_loopback	Opens a unidirectional loopback channel from TDM bus to TDM bus.
mt90503_close_channel	Closes a channel.

Statistics Functions

mt90503_get_chip_statistics	Gets general chip statistics structure.
mt90503_convert_chip_statistics_to_text	Converts chip statistics structure to a string.
mt90503_get_cbr_vc_statistics	Gets statistics structure for an open CBR VC.

mt90503_convert_cbr_vc_statistics_to_text	Converts CBR VC statistics structure to a string.
Utility Functions	
mt90503_get_handle_list	Retrieves a list of channel handles in a user specified state.
Diagnostics Functions	
mt90503_get_h100_diagnostics	Gets diagnostics structure for H100 bus.
mt90503_convert_h100_diagnostics_to_text	Converts H100 diagnostics structure to a string.
mt90503_get_console_messages	Gets diagnostic API console messages.
H100 Functions	
mt90503_set_h100_master_mode	Sets the chip's role as master on the H100 bus.
mt90503_set_h100_slave_mode	Sets which master the chip will obey.
Data Cell Functions	
mt90503_send_data_cell	Inserts a raw ATM cell in the TX data cell FIFO.
mt90503_send_test_cell	Inserts a raw ATM cell in the TX data cell FIFO that will be treated as received on a specified UTOPIA RX port.
mt90503_receive_data_cell	Retrieves a raw ATM cell from the RX data cell FIFO if one is available.
CAS Functions	
mt90503_get_cas_change	Retrieves a CAS change message if one is available.
mt90503_change_tx_cpu_cas	Changes the CAS value inserted, in the TX direction of a VC, by the CPU.
mt90503_change_rx_cpu_cas	Changes the CAS value inserted, in the RX direction of a VC, by the CPU.
mt90503_select_cas_source	Allows the user to change the source of CAS information.
Clock Recovery Functions	
mt90503_get_clk_recovery_point	Returns clock recovery points to recover a network clock from a channel.
GPIO Functions	
mt90503_set_gpio_value	Sets the specified GPIO pin to the given value.
mt90503_get_gpio_value	Gets the current input level of the specified GPIO pin.
Interrupt Functions	
mt90503_interrupt_service_routine	Function to be called when the chip asserts its interrupt.
mt90503_mask_interrupt	Function is called to temporarily disable the chip's interrupt pins.
mt90503_configure_interrupts	Function is called to change the configuration of interrupts
Polling Functions	

mt90503_poll_chip_stats	Maintains chip statistics
mt90503_poll_vc_stats	Maintains VC statistics

1.4 User Supplied Function Summary

In order to allow implementation independence the API functions make all accesses to the device through user supplied read and write functions. The requirements and considerations for these routines can be found in **Section "3.0 User Supplied Function Descriptions"**.

Write Functions

mt90503_driver_write_api	Performs a single word write to the chip.
mt90503_driver_write_apiisr	Performs a single word write to the chip.
mt90503_driver_write_osisr	Performs a single word write to the chip.
mt90503_driver_write_smear_api	Performs a smear of a word to a block of addresses.
mt90503_driver_write_smear_apiisr	Performs a smear of a word to a block of addresses.
mt90503_driver_write_smear_osisr	Performs a smear of a word to a block of addresses.

Read Functions

mt90503_driver_read_api	Performs a single word read from the chip.
mt90503_driver_read_apiisr	Performs a single word read from the chip.
mt90503_driver_read_osisr	Performs a single word read from the chip.
mt90503_driver_read_burst_api	Performs a burst of reads from the chip.
mt90503_driver_read_burst_apiisr	Performs a burst of reads from the chip.
mt90503_driver_read_burst_osisr	Performs a burst of reads from the chip.
mt90503_driver_read_debug_api	Performs a burst of reads from the chip with parity.
mt90503_driver_read_debug_apiisr	Performs a burst of reads from the chip with parity.
mt90503_driver_read_debug_osisr	Performs a burst of reads from the chip with parity.

API ISR Interface

mt90503_access_apiisr	API ISR entry point for API code entity.

1.5 System Architecture

The API is structured such that the code is stateless. All state of the API is contained in user allocated memory. This memory is referred to as the instance structures of the chip. For every API function called by the user, one of the chip's instance structure pointers is provided as a parameter. This allows the API code to service multiple chips. The instance structure pointers may be stored by the user in an array, and indexed by chip number. When an API function is to be called, the appropriate pointer can then be retrieved from the list, via the chip's index, and passed to the function.

The system architecture of the API is described below for an embedded system in two different interrupt-handling methods: with and without deferred procedure calls. In the first case, a deferred procedure call is not used, and the API's ISR is called by the OS's ISR directly at the interrupt priority level. This architecture is depicted below. All blocks shaded in dark grey in the two figures are API code. All other blocks represent code provided by the user.

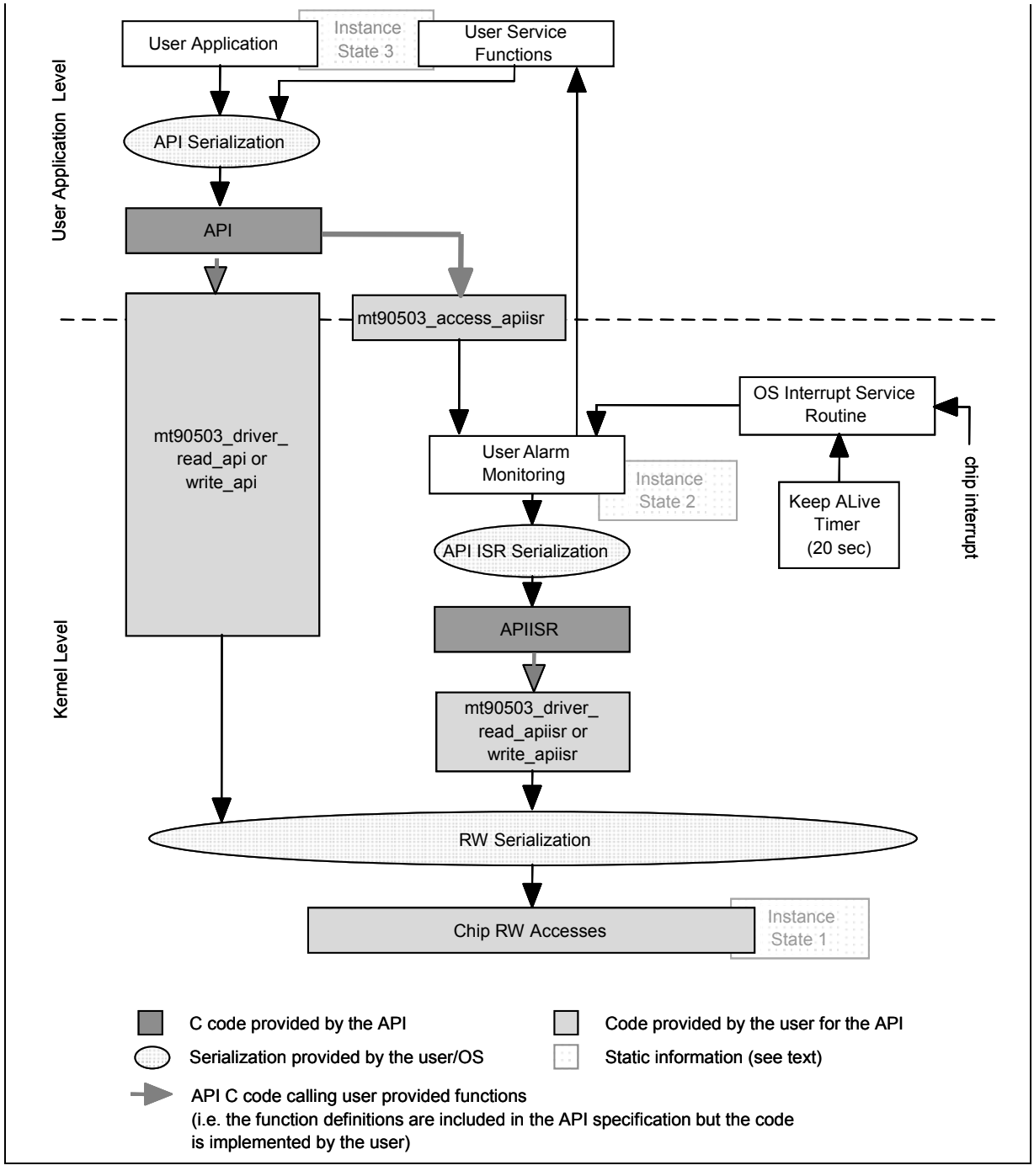


Figure 1 - System Architecture without Deferred Interrupt Procedure Call

The next figure depicts an architecture that uses deferred procedure calls. The OS's ISR simply defers the calling of the API's ISR to a later time, and at a lower interrupt priority level.

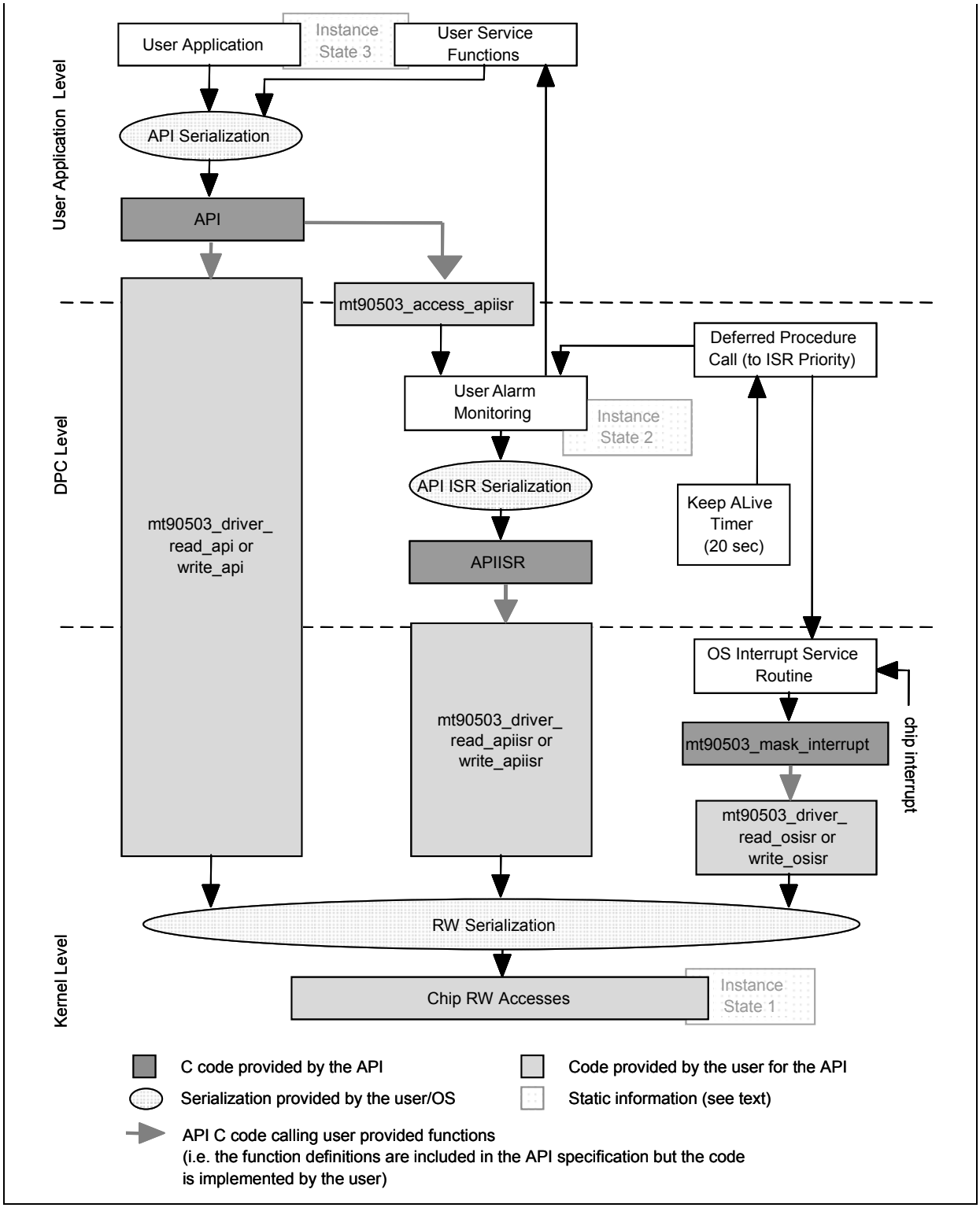


Figure 2 - System Architecture with Deferred Interrupt Procedure Call

In both architectures, an API serialization layer is needed to avoid a race condition between two threads, utilizing the same instance structure pointers, and attempting to call an API function. The serialization may be implemented

in the form of a semaphore or mutex, for example. The serialization layer lies between the API and the user application (and is managed by the user).

Another serialization layer is needed for the APIISR code entity. The code entities are described below. Because the user is responsible for calling the APIISR within this entity when an interrupt is received, and because the API can itself call the APIISR, a race condition exists. The serialization performed at this level could be implemented via interrupt priority levels, for example.

Finally, the chip-level read and write accesses must be serialized as well. The serialization is necessary because a read/write access is split into several accesses to the CPU indirection registers of the chip. To insure that an access is completed correctly, these accesses to the indirection registers must be an atomic operation.

The API is contained in three different code entities, each of which may run at a different software/OS layer. The three sections correspond to the boxes in the figures labeled:

- API
- APIISR
- mt90503_mask_interrupt

The API code entity contains the majority of the functions that are called by the user, at a user priority level. These functions are not as fast as the APIISR function, and thus should not be serialized with interrupt execution. Because of this, and because the APIISR often runs at a higher priority level, the APIISR must be separated from the main API code.

The APIISR code entity contains the API's interrupt handling function. The function in this code entity is called by the OS's ISR upon receiving an interrupt from the hardware. It can also be called from the API code entity to access resources that the API and the ISR share. Thus, serialization between the OS's ISR's calls and the API's calls to this entity must be implemented by the user.

The API's ISR does not have to run at the same priority level as the OS's ISR. To do this, the interrupt signal line must be masked out in the MT90503 to be able to execute at a lower priority level (temporarily disabling the interrupt). The smallest code entity, mt90503_mask_interrupt, performs this task. It does so with only two accesses to the chip: one read and one write. The read is performed to query the state of the chip's interrupt register. This is necessary for systems that have multiple devices sharing an interrupt line. If the chip has flagged an interrupt, a write is performed to the chip to disable the interrupt pin. It executes very quickly, thus allowing other high-priority interrupts to be serviced immediately. Because this function has no access to the instance structures of the chip, it need not be serialized with any other part of code. This function masks out all interrupts for a period of 16ms. If the API's ISR has not completed within 16 ms of masking out the interrupts, another interrupt will be generated. If no interrupt is present, the function will return a status code that allows the user to avoid an unnecessary call to the APIISR.

Because the API and APIISR entities may lie in different OS priority levels, and because some OSs protect and separate kernel space memory from user space memory, the two code entities do not access the same memory. Each code entity needs a pointer to its own distinct block of memory. The API entity needs a pointer to a block of user space memory, and the APIISR entity needs a block of kernel space memory. Each portion can only access its own memory block.

As stated earlier, the API and APIISR entities share some information. Some API functions need to return information that is gathered by the APIISR and stored in its instance structure. Because the API does not have direct access to the APIISR's instance structure, the API is given access to the APIISR's information through a user supplied function, mt90503_access_apiisr. The function serves as a "messaging pipe" between the two entities. See the function description **mt90503_access_apiisr**.

As stated earlier, the API is structured to support multiple chips. Each chip instance requires its own pair of pointers to user allocated memory: the API instance structure and the APIISR instance structure. These pointers can be stored in an array, and indexed by chip number. When the OS enforces independent memory spaces two arrays must be kept: one in the user application's memory space for the API instance structures, and the other in the ISR's

memory space for the APIISR structures. The two arrays are depicted in the figures above as “Instance State 3” and “Instance State 2”, respectively.

The size of these memory blocks is determined by the API function mt90503_open_instance_size, described later in this document. The function is called for each chip, before initially configuring it. The function takes a chip configuration structure as a parameter and uses it to return the memory size required for the API and APIISR instance structures. See the function description **mt90503_open_instance_size**.

The read and write routines supplied by the user are used by the API functions to access all chips which the API code is servicing. The chip and its associated instance structure are configured via a call to the function mt90503_open. The function receives a chip configuration structure as a parameter. In this structure is the user_chip_number parameter that is intended to be the index of the chip being opened. Because every API and APIISR function receives a pointer to an instance structure as the first parameter, the chip number is available to all API functions. The only use of the user_chip_number by the API is to provide it as a parameter to the read/write functions. By associating a chip number to a particular chip, the correct device can be accessed in the user provided read/write routines. For example, chip number could be associated to a base address in the system. The user can then offset the provided address of a read/write routine and perform an access to the correct device. As illustrated in the two figures above, this information is easily stored as an array of chip specific information (e.g. base addresses) and can be indexed by the chip number. Note that the same chip number can be used to access system arrays kept by the user in different memory regions (e.g. user vs. kernal):

- API instance structure pointer array (Instance State 3),
- APIISR instance structure pointer array (Instance State 2),
- Read/Write function chip info (Instance State 1).

The two figures above indicate that two or three versions of the same read and write functions must be supplied. These functions differ only in the layer of their entry point. The functions in the group mt90503_driver_read_api or write_api are accessible only from the user application space, the group mt90503_driver_read_apiisr or write_apiisr from the DPC priority level in kernel space, and the group mt90503_driver_read_osisr or write_osisr from the interrupt priority level in the kernel space. In the case where deferred procedure calls are not used, the third group is not needed.

The mt90503_interrupt_service_routine, located in the APIISR code entity, returns a vector of the interrupts that were serviced; it is the responsibility of the User Alarm Monitoring function to call any required user functions to continue the servicing in the user application. For example, if the user wanted to service data cells (AAL0) as soon as they are received, the rx_data_fifo_stale_time parameter would be set to the minimum desired delay value (e.g. 1ms) and the alarm_data_cell_fifo_int_conf parameter would be set to MT90503_INT_NO_TIMEOUT. (The two parameters are part of the chip configuration structure MT90503_CONF.) When a data cell arrives, the chip would assert an interrupt at most 1ms later. In response to the interrupt the OS ISR calls the API ISR, which services the interrupt and returns the vector indicating a data cell interrupt. The User Alarm Monitoring function then calls the user routine (in the user space) for data cells, which calls the API routine mt90503_receive_data_cell to obtain the cell.

The next figure depicts the system architecture used to perform the debugging of the API. The architecture is implemented on a Windows NT platform. Note that the API’s ISR is located in the user space to facilitate debugging. Also important is the presence of a separate thread. This thread is dedicated to handling interrupts only. It waits for a flag from the OS’s ISR indicating that an interrupt has been generated. Upon receiving the flag, the interrupt thread calls the API’s ISR. The thread then performs appropriate actions based on the value of the event vector returned by the API’s ISR.

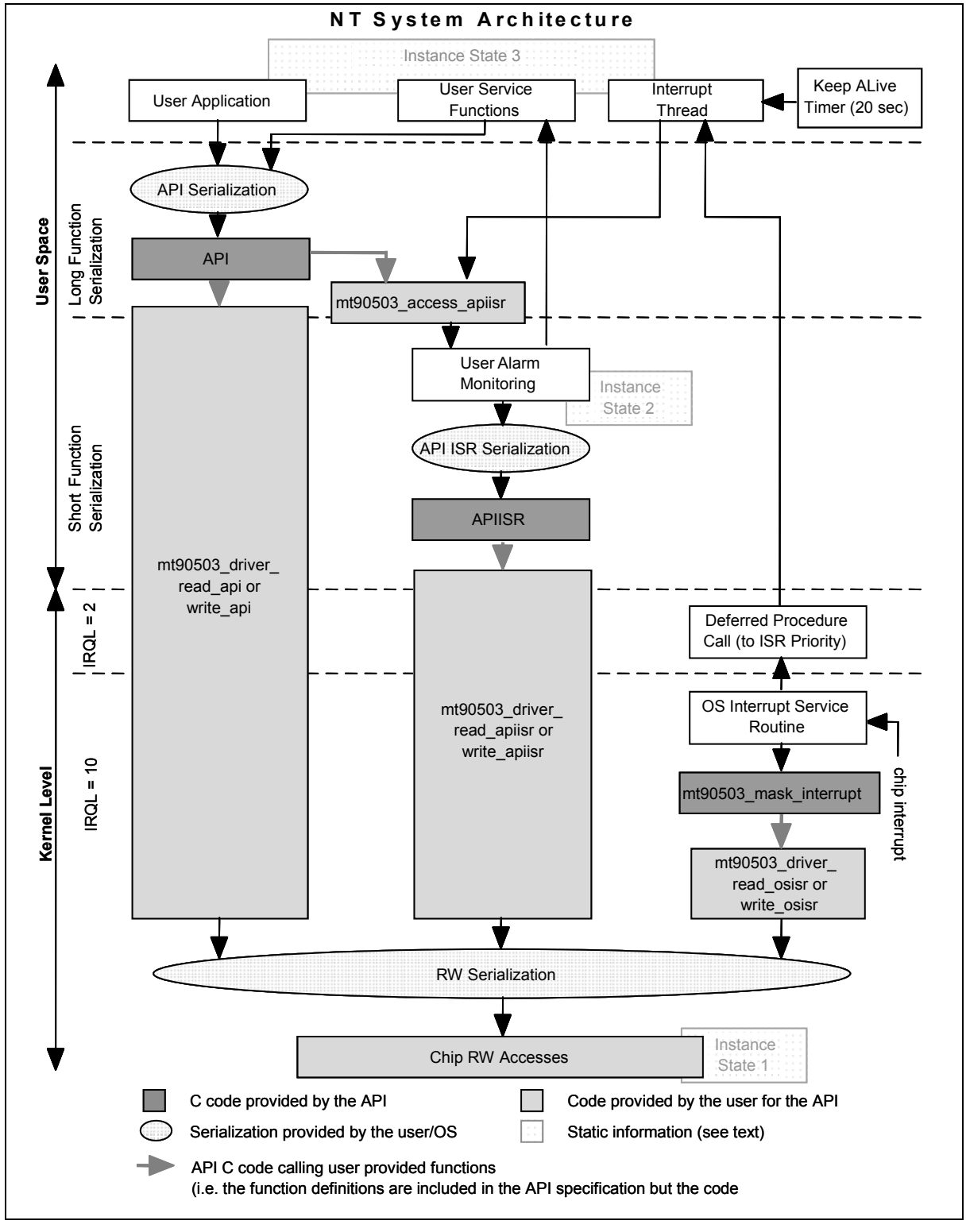


Figure 3 - NT System Architecture

The APIISR must be called at least every 20 seconds. If it is not, counters within the chip will not be updated in the API correctly, causing the API to fall out of synchronization with the chip, which can lead to system to failure. In a

system where the interrupt line of the chip is routed to a CPU, APIISR code insures that the APIISR will be called at least at the required frequency. In the case where the interrupt line is not physically routed, a keep-alive timer is needed by the system, as illustrated in the system architecture figures above. The timer insures the APIISR is called at least every 20 seconds. Although calling the APIISR every 20 seconds is enough to keep the chip running correctly, VC and chip statistics counters also have to be kept up to date via calls to the **mt90503_poll_vc_stats** and the **mt90503_poll_chip_stats** functions. If such calls are not done frequently enough, incorrect statistics may be present in the API structures.

2.0 API Function Descriptions

Each function's use as well as its parameters is described here in detail. The typical usage of the above functions is as follows:

- A parameter structure is allocated.
- The appropriate open default configuration function is called. These functions are identified by the "_def" suffix at the end of the function name.
- The user changes the default configuration structure to suit his needs.
- The actual function is called.

An example of this sequence is the initialization of the chip. Note that in the following example the system architecture is assumed to have all code (API and APIISR) in the same memory space:

```
#include "mt90503_api.h"

void main( )

{
            MT90503_INSTANCE_API*              pmt90503_api;
            MT90503_INSTANCE_APIISR*           pmt90503_apiisr;
            MT90503_CONF                       mt90503_conf;
            MT90503_INSTANCE_SIZE              mt90503_inst_size;
            ULONG                              result;


            // Inserting default values into structure configuration parameters.
            mt90503_open_def(&mt90503_conf);


            // Change default parameters as needed (e.g. changing the clock
            frequencies).
            mt90503_conf.upclk_freq = 30000000;
            mt90503_conf.mclk_freq = 70000000;


            // Inserting default values into MT90503_INSTANCE_SIZE structure
            parameters.
            mt90503_open_instance_size_def(&mt90503_conf, &mt90503_inst_size);



            // Get the size of the MT90503_INSTANCE structures.
            result = mt90503_open_instance_size(&mt90503_conf,(&mt90503_inst_size
            );
            if (result!= MT90503ER_GENERIC_OK)
            {
                        // Error handling.
```

```
            }

            // Allocate memory for the mt90503_instance structure

            pmt90503_api = MT90503_INSTANCE_API*)malloc(
                                    mt90503_inst_size.instance_api_size);
            if (pmt90503_api == NULL)
            {
                        // Error handling.
            }
            pmt90503_apiisr = (MT90503_INSTANCE_APIISR*)malloc(
                                    mt90503_inst_size.instance_apiisr_size);
            if (pmt90503_apiisr == NULL)
            {
                        // Error handling.
            }

            // Perform the actual configuration of the chip.
            result = mt90503_open(pmt90503_api, &mt90503_conf);
            if (result!= MT90503ER_GENERIC_OK)
            {
            // Error handling.
            }

}
```

Every function has a pointer to the chip's API instance structure as the first parameter. This instance structure is created by the user before the call to **mt90503_open** and is unique to each chip being managed by the software. The structure keeps the state of an instance of a chip and is required to perform any operations on the chip. The APIISR instance structure is kept by the system, and passed as a parameter to the APIISR code entity when the interrupt service routine is to be called (by the user or the API entity).

2.1 Initialization Functions

2.1.1 mt90503_open; mt90503_open_sw & mt90503_open_hw

Using the provided configuration structure MT90503_CONF, mt90503_open performs all the necessary operations to configure the chip and initialize the instance structure. Note that the functions **mt90503_open_def** and **mt90503_open_instance_size** are typically called, in their respective order, before this function.

The mt90503_open_def function inserts default values into the MT90503_CONF structure. The default value of a structure field is indicated following the field's description.

The **mt90503_open** function initializes both the API and APIISR instance structures used to monitor the status of the chip's resources, and performs all accesses to the chip necessary to initialize the device according to the provided configuration. This function can be split into two separate steps, the initialization of the instance structures (software) and the initialization of the device based on the instance structure (hardware), by calling the **mt90503_open_sw** function followed by the **mt90503_open_hw** function. The **mt90503_open_sw** function initializes the API and APIISR instance structures. The **mt90503_open_hw** function initializes the chip according to the contents of the instance structures, and thus must be called after **mt90503_open_sw**.

Independent of which method is used to open the chip (mt90503_open, or mt90503_open_sw followed by mt90503_open_hw), the **mt90503_open_def** is always used to initialize the **MT90503_CONF** structure.

Usage

```
#include “mt90503_api.h”
            ULONG mt90503_open_def( MT90503_CONF* pmt90503_conf );
            ULONG mt90503_open( MT90503_INSTANCE_API* pmt90503_api,
                        MT90503_CONF* pmt90503_conf );
            ULONG mt90503_open_sw( MT90503_INSTANCE_API* pmt90503_api,
                        MT90503_CONF* pmt90503_conf );

            ULONG mt90503_open_hw( MT90503_INSTANCE_API* pmt90503_api );
```

Return Values

MT90503ER_GENERIC_OK	Indicates success

Also see **Section "4.0 Return Codes"** for non-successful codes.

Parameters

pmt90503_api a pointer to the chip’s API instance structure. This structure will be filled in by this function call. It contains information of the current state and configuration of the chip. After initialization by mt90503_open this structure is supplied to all subsequent function calls. The structure must be created and kept by the application software until mt90503_close is called.

pmt90503_conf a pointer to an initial configuration structure MT90503_CONF. The definition of the structure is provided in **Section "5.0 Configuration Structures"**, as are the default values inserted by mt90503_open_def.

2.1.2 mt90503_open_instance_size

Using the provided configuration structure MT90503_CONF, **mt90503_open_instance_size** calculates the amount of memory required for the MT90503_INSTANCE_API and MT90503_INSTANCE_APIISR structures of the chip. An MT90503_INSTANCE_API structure and an MT90503_INSTANCE_APIISR structure must be allocated and pointers created by the user before calling the **mt90503_open** function; both pointers must point to blocks of contiguous memory whose sizes are determined by this function.

The mt90503_open_instance_size_def function inserts default values into the MT90503_ INSTANCE_SIZE structure. The default value of a structure field is indicated following the field’s description.

Usage

```
#include “mt90503_api.h”

ULONG mt90503_open_instance_size_def(MT90503_INSTANCE_SIZE* pinstance_size );

ULONG mt90503_open_instance_size( MT90503_CONF* pconf,
            MT90503_INSTANCE_SIZE* pinstance_size );
```

Return Values

MT90503ER_GENERIC_OK	Indicates success

Also see **Section "4.0 Return Codes"** for non-successful codes.

Parameters

pconf a pointer to an initial configuration structure MT90503_CONF. The definition of the structure is provided in **Section "5.0 Configuration Structures"**.
See **2.1.2 mt90503_open_instance_size** for a default configuration of the chip. The user allocates this structure.

pinstance_size pointer to an MT90503_INSTANCE_SIZE structure. The definitions of the structure’s elements are listed below. The user allocates this structure.

2.1.2.1 Structure MT90503_INSTANCE_SIZE

instance_api_size ?? – ??

This value is returned by the function and indicates the size, in bytes, of the MT90503_INSTANCE_API memory block that must be allocated to support the supplied configuration.

Direction: Out Type: ULONG

Default: *NOT MODIFIED*

instance_apiisr_size ?? – ??

This value is returned by the function and indicates the size, in bytes, of the MT90503_INSTANCE_APIISR memory block that must be allocated to support the supplied configuration.

Direction: Out Type: ULONG

Default: *NOT MODIFIED*

2.1.3 mt90503_close

This function closes any VCs or channels that may still be open and then puts the chip in reset.

The mt90503_close_def function inserts default values into the MT90503_CLOSE_CHIP structure. The default value of a structure field is indicated following the field's description.

Usage

```
#include "mt90503_api.h"

ULONG mt90503_close_def( MT90503_INSTANCE_API* pmt90503_api,
            MT90503_CLOSE_CHIP* pclose_chip );

ULONG mt90503_close( MT90503_INSTANCE_API* pmt90503_api,
            MT90503_CLOSE_CHIP* pclose_chip );
```

Return Values

`MT90503ER_GENERIC_OK` Indicate success.

Also see **Section "4.0 Return Codes"** for non-successful codes.

Parameters

pmt90503_api pointer to the instance structure of the chip.

pclose_chip pointer to an MT90503_CLOSE_CHIP structure. The definitions of the structure's elements are listed below. The user allocates this structure.

2.1.3.1 Structure MT90503_CLOSE_CHIP

Currently there are no parameters for this structure.

2.1.4 mt90503_get_hw_revision

This routine returns the hardware revision number of the MT90503. The revision number is contained in a register of the device. This function may be called before the device is open and only requires upclk to be present on the device.

The mt90503_get_hw_revision_def function inserts default values into the MT90503_REVISION structure. The default value of a structure field is indicated following the field's description.

Usage

```
#include "mt90503_api.h"

ULONG mt90503_get_hw_revision_def(MT90503_REVISION* prevision );

ULONG mt90503_get_hw_revision(MT90503_REVISION* prevision );
```

Return Values

MT90503ER_GENERIC_OK	Indicates success

Also see **Section "4.0 Return Codes"** for non-successful codes.

Parameters

prevision pointer to an MT90503_REVISION structure. The definitions of the structure's elements are listed below. The user allocates this structure.

2.1.4.1 Structure MT90503_REVISION

user_chip_number identifier

This number is carried down to the user-supplied read/write routines to distinguish which chip the API is servicing. This can be used as an array index of the chip to be serviced to retrieve the correct instance pointer. If only one chip is being serviced by the API, then this parameter can be ignored. See **Section "1.5 System Architecture"**.

Direction: IN Type: ULONG

Default: UNDEFINED

revision_number 0 – ??

This value is returned by the function and indicates the revision of the device.

Direction: Out Type: ULONG

Default: UNDEFINED

2.2 ATM Functions:

2.2.1 mt90503_open_cbr_vc

This function opens a constant bit rate VC from the TDM bus to UTOPIA bus, and vice-versa. On the TX side, data is taken from the VC's channels on the TDM bus, assembled by the TXSAR into ATM cells, and sent on the UTOPIA bus. On the RX side, ATM cells are received on UTOPIA, disassembled by the RXSAR, and sent out onto the VC's channels on the TDM bus.

If the call to this function is successful, cells are sent by the TXSAR and received by the RXSAR immediately following the call. However, no active channels are associated with the VC. Thus, cells assembled by the TXSAR will contain null bytes until active channels are allocated to the VC. Channels are allocated to a VC via the function **mt90503_open_channel_in_vc**.

When the UTOPIA module receives cells for this VC, they will be routed according to the **rx_normal_cell_routing** and **rx_oam_cell_routing** fields. This routing must not conflict with other VCs that are received on a common

UTOPIA port. Two VCs conflict when, after the application of the **u_tx*p*_network_mask** (where *p* is the port), they have the same header on the same UTOPIA port. Each UTOPIA port is given a **u_tx*p*_network_mask** during the call to **mt90503_open**.

The VC can also be used to source the chip's clock recovery FIFOs A and B. Only one CBR VC can be used to source each FIFO. See **Section "2.10 Clock Recovery Functions"**.

This function returns a handle by which the API identifies this VC.

The mt90503_open_cbr_vc_def function inserts default values into the CBR VC configuration structure, MT90503_CBR_VC. The default value of a structure field is indicated following the field's description.

Usage

```
#include "mt90503_api.h"

ULONG mt90503_open_cbr_vc_def( MT90503_INSTANCE_API* pmt90503_api,
            MT90503_CBR_VC* pmt90503_cbr_vc );

ULONG mt90503_open_cbr_vc( MT90503_INSTANCE_API* pmt90503_api,
            MT90503_CBR_VC* pmt90503_cbr_vc );
```

Return Values

MT90503ER_GENERIC_OK	Indicates success

Also see **Section "4.0 Return Codes"** for non-successful codes.

Parameters

pmt90503_api pointer to an API instance structure of the chip

pmt90503_cbr_vc pointer to an MT90503_CBR_VC structure. The definitions of the structure's elements are listed below. The user allocates this structure.

2.2.1.1 Structure MT90503_CBR_VC

pvc_hndl

pointer to a single ULONG which returns the handle for the created VC. This handle is a unique value that identifies the VC in all future function calls affecting this VC. The handle's ULONG must be allocated by the user prior to calling this function.

Direction: IN/OUT Type: POINTER

Default: NULL

header 32 bit field

header of the VC. Header fields are in the following order (starting from bit 31): GFC, VPI, VCI, PT, CLP.

Direction: IN Type: ULONG

Default: MT90503_NULL_HEADER

rx_tx_utopia_port MT90503_PORTA
MT90503_PORTB
MT90503_PORTC

Sets the UTOPIA TX port to which the VC's cells are destined once they exit the TXSAR, and from which UTOPIA RX port cells enter the chip. The VC can only be associated to one port. This field works in conjunction with **loopback**.

Direction: IN Type: ULONG

Default: MT90503_INVALID_UTOPIA_PORT

wheel_number {0 – 14,0xFFFFFFFF}

The wheel to use to map the events of this VC. A value of 0xFFFFFFFF will allow the API to use any wheel that meets the event requirements. See **Section "5.1.6 TXSAR Scheduler Parameters"**.

Direction: IN Type: ULONG

Default: 0xFFFFFFFF

vc_payload_type see below for values.

The type and size of the VC's payload. The choices are:

MT90503_VC_TYPE_FULLY_FILLED_AAL0
Fully filled AAL0 cell. There are 48 payload bytes per cell.

MT90503_VC_TYPE_PARTIALLY_FILLED_AAL0
Partially filled AAL0 cell. The range of partial fills is 4 to 47 and must be specified in **vc_payload_size**. A value of 48 will be considered a fully filled cell.

MT90503_VC_TYPE_FULLY_FILLED_AAL5
Fully filled AAL5 cell. There are 40 payload bytes per cell.

MT90503_VC_TYPE_ PARTIALLY_FILLED _AAL5
Partially filled AAL5 cell. The range of partial fills is {8, 16, 24, 32} and must be specified in **vc_payload_size**. A value of 40 will be considered a fully filled cell.

MT90503_VC_TYPE_FULLY_FILLED_UNSTRUCTURED_AAL1
Fully filled AAL1 unstructured cell. There are 47 payload bytes per cell. An unstructured AAL1 VC is one with no P-Byte in any of its cells; payload bytes are aligned on byte boundaries.

MT90503_VC_TYPE_PARTIALLY_FILLED_UNSTRUCTURED_AAL1
Partially filled AAL1 unstructured cell. The range of partial fills is 4 to 46 and must be specified in **vc_payload_size**. A value of 47 is interpreted as a fully filled cell.

MT90503_VC_TYPE_FULLY_FILLED_STRUCTURED_AAL1
Fully filled AAL1 structured cell. There are 375 payload bytes per 8 cells. A structured AAL1 VC is one that contains a P-Byte in one cell for every cycle of eight cells.

MT90503_VC_TYPE_PARTIALLY_FILLED_STRUCTURED_AAL1
Partially filled AAL1 structured cell. The range of partial fills is 4 to 46 and must be specified in **vc_payload_size**. A value of 47 is interpreted as a fully filled cell. A structured AAL1 VC is one that contains a P-Byte in one cell for every cycle of eight cells.

Direction: IN Type: ULONG

Default: MT90503_VC_TYPE_FULLY_FILLED_ STRUCTURED_AAL1

vc_payload_size (see vc_payload_type)

A size must be specified for **vc_payload_type**'s of:

MT90503_VC_TYPE_PARTIALLY_FILLED_AAL0
MT90503_VC_TYPE_PARTIALLY_FILLED_UNSTRUCTURED_AAL1
MT90503_VC_TYPE_PARTIALLY_FILLED_STRUCTURED_AAL1

For all other **vc_payload_type**'s this parameter is unused.

Direction: IN Type: ULONG

Default: 0

vc_support_of_cas see below for values.

Determines whether multi-framing and CAS is used, and if so on which links. There are six possible modes for CAS support. Note that to support multi-framing and CAS values on a VC, the **vc_payload_type** field must be set to MT90503_VC_TYPE_FULLY_FILLED_STRUCTURED_AAL1. The **vc_cas_type** field specifies the type of multi-framing and CAS supported.

MT90503_NO_MF_CAS

No CAS values are carried either in the TDM channels of the VC or in the ATM cells.

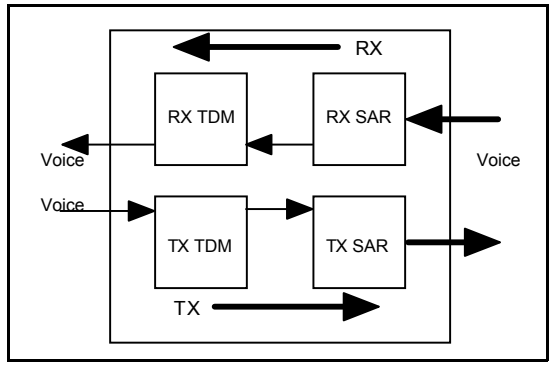


Figure 4 - MT90503_NO_MF_CAS

MT90503_STRICT_MF_CAS_TDM_ATM
CAS values are carried both on the TDM bus and UTOPIA bus. On the TX side of the VC, CAS values are received from the TDM bus and inserted into ATM cells assembled by the TXSAR. On the RX side of the VC, CAS values are taken from ATM cells disassembled by the RXSAR and sent onto the TDM bus. The multi-frame integrity is respected.

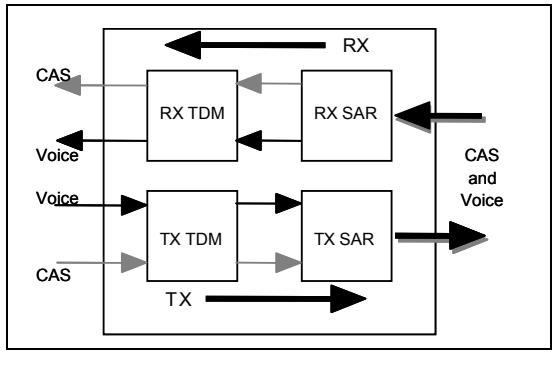


Figure 5 - MT90503_STRICT_MF_CAS_TDM_ATM, MT90503_NOT_STRICT_MF_CAS_TDM_ATM

MT90503_NOT_STRICT_MF_CAS_TDM_ATM
CAS values are carried both on the TDM bus and UTOPIA bus. On the TX side of the VC, CAS values are received from the TDM bus and inserted into ATM cells assembled by the TXSAR. On the RX side of the VC, CAS values are taken from ATM cells disassembled by the RXSAR and sent onto the TDM bus. The multi-frame integrity is not respected. The transmission and reception SAR delay is smaller in this mode than in the strict mode.

MT90503_MF_CAS_TDM_CPU

The exchange of CAS values is between the TDM bus and the CPU. Therefore there are no CAS values transmitted or received in the ATM cells of the VC. On the TX side of the VC, CAS values are received from the TDM bus. Changes in the CAS values are reported to the CPU. On the RX side the CPU inserts CAS values onto the TDM bus. The CAS value inserted on a channel is specified when the channel is opened (see mt90503_open_channel_in_vc). The CAS value inserted by the CPU can be changed once the channel is open, via a call to mt90503_change_rx_cpu_cas.

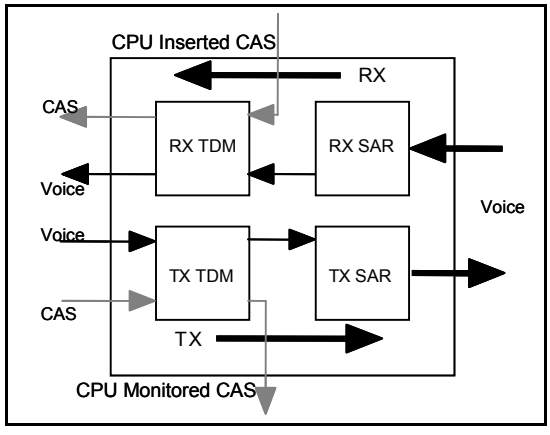


Figure 6 - MT90503_MF_CAS_TDM_CPU

MT90503_MF_CAS_ATM_CPU

The exchange of CAS values is between the ATM cells of the VC and the CPU. Therefore there are no CAS values transmitted or received on the TDM bus. On the RX side of the VC, CAS values are received from the ATM cells and disassembled by the RXSAR. Changes in the CAS values are reported to the CPU. On the TX side the CPU inserts CAS values into the ATM cells assembled by the TXSAR. The CAS values inserted by the CPU are specified when each channel in the VC is opened (see mt90503_open_channel_in_vc). Each channel contributes one CAS value. This CAS value can be changed once the channel is open via a call to mt90503_change_tx_cpu_cas.

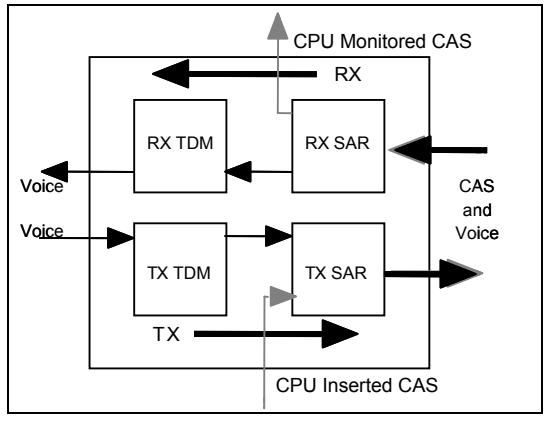


Figure 7 - MT90503_MF_CAS_ATM_CPU

MT90503_MF_CAS_TDM_ATM_CPU

The exchange of CAS values is between the TDM bus, the CPU, and the ATM cells of the VC. On the RX side of the VC, CAS values are received from the ATM cells disassembled by the

RXSAR. Changes in the CAS values are reported to the CPU. The CPU inserts CAS values on the TDM bus. These CAS values are specified per channel when the channels are opened (see **mt90503_open_channel_in_vc**). On the TX side, CAS values are received from the TDM bus. CAS changes are reported to the CPU. The CPU inserts CAS values into the ATM cells assembled by the TXSAR. The CAS values inserted are specified per channel when the channels are opened (see **mt90503_open_channel_in_vc**). Each channel contributes one CAS value. The CAS values inserted by the CPU in the TX direction can be changed per channel once the channel is open, via a call to **mt90503_change_tx_cpu_cas**. The CAS values inserted by the CPU in the RX direction can be changed per channel once the channel is open, via a call to **mt90503_change_rx_cpu_cas**.

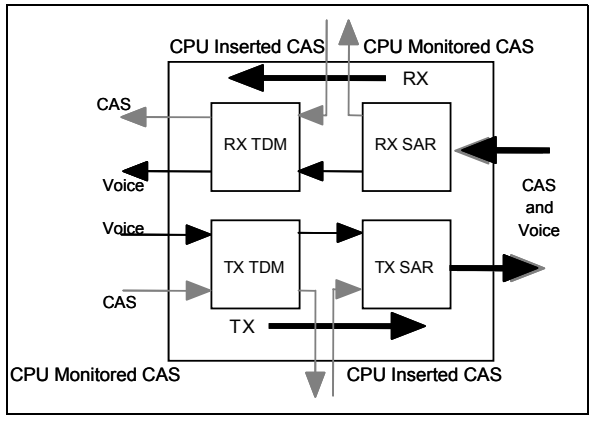


Figure 8 - MT90503_MF_CAS_TDM_ATM_CPU

Direction: IN Type: ULONG

Default: MT90503_NO_MF_CAS

vc_cas_type MT90503_E1_MF_CAS
MT90503_T1_MF_CAS

Sets the type of multi-framing used if multi-framing is supported on this VC. E1 multi-framing is 16 frames long. T1 multi-framing is 24 frames long.

Direction: IN Type: ULONG

Default: MT90503_E1_MF_CAS

number_of_channels 1 – 2048 (see below for guidelines)

The trunk size of the VC in 64 kps channels. Once this function returns successfully, the VC is configured with inactive channels. Cells are sent containing null bytes. Received cells are disassembled into the VC's individual channels, but the resulting data is discarded. As channels are added to this VC, via calls to **mt90503_open_channel_in_vc**, valid data will be sent/received in the cells. The number of channels added to this VC cannot exceed the trunk size. The following table gives guidelines for the relationship between **number_of_channels** and **vc_payload_type**:

vc_payload_type	
AAL0, AAL5, and AAL1 Unstructured VCs.	Because there is no framing structure within the ATM cells (i.e. no p-byte), the payload size of the VC must be a multiple of the trunk size if voice is to be carried on this VC. If this relationship holds, then a constant mapping exists between the payload bytes of the ATM cells sent and the channels of the VC. Thus the destination of the VC can properly disassemble its cells into the proper channels. If the relationship does not hold, voice cannot be carried on the VC, but the possibility for video/data streams still remains. The range of number_of_channels is 1 to 2048. For example, if the payload size is set to 24 then number_of_channels can be set to 1, 2, 3, 4, 6, 8, 12, or 24. If payload size is not a multiple of the trunk size, the individual channels of the VC cannot be distinguished when the cells are received. Thus voice could not be carried under such conditions. However, video/data streams could be carried over such a VC. Rules of thumb: • One channel per VC always works for voice. • For AAL0 fully-filled the following number of channels work for voice: {1, 2, 3, 4, 6, 8, 12, 16, 24, 48}. • For AAL1 unstructured fully-filled the following number of channels work for voice: {1, 47}.
AAL1 Structured VCs.	Because there is a framing structure in the ATM cells (i.e. the p-byte), the destination of the VC can always synchronize itself so as to properly disassemble the payload of the cells into the proper channels. Both voice and video can be carried on such a VC. Thus the payload size of the cells has no impact on the number of channels that can be added to such a VC. The range is 1 to 2048.
Multi-framing CAS VCs.	For the same reasons as for AAL1 structured VCs, the payload size has no impact on the number of channels that can be allocated to a multi-framing CAS VC. However, the range for multi-framing CAS VCs is 1 to 128.

Direction: IN Type: ULONG

Default: 1

rx_normal_cell_routing 0, or the OR of any or all of:
MT90503_PORTA
MT90503_PORTB

MT90503_PORTC
MT90503_DATA_CELL_FIFO

This field routes the VC's non-OAM cells entering the chip via rx_tx_utopia_port. The values can be ORed together to broadcast the cell. Since this function opens a bi-directional CBR VC, normal cells are automatically routed to the RXSAR. If the field is set to 0, the normal cells on this VC will only go to the RXSAR.

Direction: IN Type: ULONG

Default: 0

rx_oam_cell_routing 0, or the OR of any or all of:
MT90503_PORTA
MT90503_PORTB
MT90503_PORTC
MT90503_DATA_CELL_FIFO

This field routes the VC's OAM cells entering the chip via **rx_tx_utopia_port**. The values can be ORed together to broadcast the cell. If the field is set to 0, the OAM cells on this VC will be discarded.

Direction: IN Type: ULONG

Default: 0

clk_recov_source_a TRUE / FALSE

Whether this VC is used as the source for generating clock recovery points for FIFO A.

Direction: IN Type: ULONG

Default: FALSE

clk_recov_source_b TRUE / FALSE

Whether this VC is used as the source for generating clock recovery points for FIFO B.

Direction: IN Type: ULONG

Default: FALSE

loopback TRUE / FALSE

In the TX direction, the cells produced by the TXSAR can either exit the chip from the determined UTOPIA port (no loopback, FALSE) or can be treated as if they had entered the chip from that same port (loopback, TRUE). See **rx_tx_utopia_port** above. A typical application is to perform self tests by feeding data on the TDM bus and routing the cells produced by the TXSAR back to the RXSAR, which disassembles the cells, and puts the data on the TDM bus.

Direction: IN Type: ULONG

Default: FALSE

maximum_cdv 0 – ?? (see below for guidelines)

The amount of CDV the VC must absorb in microseconds. A larger value for this field means the buffers used for disassembling the cells will be larger. A value of 1000 us means that +/- 1 ms of delay variation will be absorbed; thus a worst case variation of 2 ms between two cells could potentially be absorbed.

T1 multi-framing CAS VCs with strict multi-framing	rx_circular_buffer_size = ((maximum_cdv / 64) + (47 / number_of_channels) + 72) * 4/3
T1 Mmulti-framing CAS VCs with FASTCAS	rx_circular_buffer_size = ((maximum_cdv / 64) + (47 / number_of_channels) + 48) * 4/3
E1 multi-framing CAS VCs with strict multi-framing	rx_circular_buffer_size = ((maximum_cdv / 64) + (47 / number_of_channels) + 48) * 2
E1 Mmulti-framing CAS VCs with FASTCAS	rx_circular_buffer_size = ((maximum_cdv / 64) + (47 / number_of_channels) + 32) * 2
Non Multi-framing VCs	rx_circular_buffer_size = (maximum_cdv / 64) + (47 / number_of_channels)

Direction: IN Type: ULONG

Default: 8000 (8 ms)

rx_circular_buffer_size **{128, 256, 512, 1024, 0xFFFFFFFF}**

Specifies the size of RX circular buffer that the API must use to absorb CDV (in bytes). When circular buffer playback for underrun padding is **not** being used the recommended value for this field is 0xFFFFFFFF, which allows the API to automatically assign a circular buffer size based on the **maximum_cdv** required. When circular buffer playback for underrun padding method is used this parameter will allow the user to force the circular buffer to a specific size which may be equal to or greater than the size required by **maximum_cdv**. If the specified size is less than that required by **maximum_cdv** the return code will be an unsuccessful code. See **rx_underrun_pad_type** and MT90503_RX_PAD_UR_WITH_OLD_DATA.

Direction: IN Type: ULONG

Default: 0xFFFFFFFF

cut_vc_detect_time 100-10000

The number of milliseconds during which no cells are received before a VC is declared cut.

Direction: IN Type: ULONG

Default: 1000 (1 sec)

2.2.2 mt90503_open_data_vc

This function opens a VC from UTOPIA bus to the data cell FIFO or back to the UTOPIA bus. The VC can be opened for many uses, such as terminating the VC in the data cell FIFO, routing the VC to a secondary SAR, or performing a header change on the VC.

When the UTOPIA module receives cells for this VC, they will be routed according to the **rx_normal_cell_routing** and **rx_oam_cell_routing** fields. This routing must not conflict with other VCs that are received on a common UTOPIA port. Two VCs conflict when, after the application of the **u_tx*p*_network_mask** (where *p* is the port), the received cells have the same header on the same UTOPIA port. Each UTOPIA port is given a **u_tx*p*_network_mask** during the call to **mt90503_open**.

This function returns a handle by which the API identifies this VC.

The mt90503_open_data_vc_def function inserts default values into the configuration structure of the data VC, MT90503_DATA_VC. The default value of a structure field is indicated following the field's description.

Usage

```
#include “mt90503_api.h”

ULONG mt90503_open_data_vc_def( MT90503_INSTANCE_API* pmt90503_api,
            MT90503_DATA_VC* pdata_vc );

ULONG mt90503_open_data_vc( MT90503_INSTANCE_API* pmt90503_api,
            MT90503_DATA_VC* pdata_vc);
```

Return Values

MT90503ER_GENERIC_OK	Indicates success

Also see **Section "4.0 Return Codes"** for non-successful codes.

Parameters

pmt90503_api pointer to an API instance structure of the chip

pdata_vc pointer to an MT90503_DATA_VC structure. The definitions of the structure’s elements are listed below.

2.2.2.1 Structure MT90503_DATA_VC

pvc_hndl pointer to a single ULONG which returns the handle for the created VC. This handle is a unique value that identifies the VC in all future function calls affecting this VC. The handle's ULONG must be allocated by the user prior to calling this function.

Direction: IN/OUT Type: POINTER

Default: NULL

rx_utopia_port MT90503_PORTA
MT90503_PORTB
MT90503_PORTC

The UTOPIA port on which cells for this VC will enter the chip.

Direction: IN Type: ULONG

Default: MT90503_INVALID_UTOPIA_PORT

header 32 bit field

header of the VC. Header fields are in the following order (starting from bit 31): GFC, VPI, VCI, PT, CLP.

Direction: IN Type: ULONG

Default: MT90503_NULL_HEADER

rx_normal_cell_routing 0 or the OR of any or all of:
MT90503_PORTA
MT90503_PORTB
MT90503_PORTC
MT90503_DATA_CELL_FIFO

The routing of non-OAM cells once they have entered the chip via the UTOPIA RX port specified in **rx_utopia_port**. Since this is a data VC, the cells cannot be routed to the RXSAR. A value of 0 will cause non-OAM cells to be discarded.

Direction: IN Type: ULONG

Default: 0

rx_oam_cell_routing 0, or the OR of any or all of:
MT90503_PORTA
MT90503_PORTB
MT90503_PORTC
MT90503_DATA_CELL_FIFO

The routing of OAM cells once they have entered the chip via the UTOPIA RX port specified in **rx_utopia_port**. A value of 0 will cause OAM cells to be discarded.

Direction: IN Type: ULONG

Default: 0

replace_gfc TRUE / FALSE

Whether the VPI[11:8] bits of the cell's header are to be changed, or not, by the VC's LUT entry. The new value of the GFC bits is determined by the value of **new_gfc**. Header translation is not possible if cells on this VC are routed to the RX data cell FIFO. This parameter must be FALSE is this case.

Direction: IN Type: ULONG

Default: FALSE

replace_vpi TRUE / FALSE

Whether the VPI[7:0] bits of the cell's header are to be changed, or not, by the VC's LUT entry. The new value of the VPI bits is determined by the value of **new_vpi**. Header translation is not possible if cells on this VC are routed to the RX data cell FIFO. This parameter must be FALSE is this case.

Direction: IN Type: ULONG

Default: FALSE

replace_vci TRUE / FALSE

Whether the VCI bits of the cell's header are to be changed, or not, by the VC's LUT entry. The new value of the VCI bits is determined by the value of **new_vci**. Header translation is not possible if cells on this VC are routed to the RX data cell FIFO. This parameter must be FALSE is this case.

Direction: IN Type: ULONG

Default: FALSE

new_gfc 4-bit field

The new GFC bits of the cell's header if GFC replacement is requested (**replace_gfc** = TRUE). The GFC bits will be replaced by the LUT entry corresponding to the VC, as the VC is routed.

Direction: IN Type: ULONG

Default: 0x0

new_vpi 8-bit field

The new VPI bits of the cell's header if VPI replacement is requested (**replace_vpi** = TRUE). The VPI bits will be replaced by the LUT entry corresponding to the VC, as the VC is routed.

Direction: IN Type: ULONG

Default: 0x00

new_vci 16 bit field

The new VCI bits of the cell's header if VCI replacement is requested (**replace_vci** = TRUE). The VCI bits will be replaced by the LUT entry corresponding to the VC, as the VC is routed.

Direction: IN Type: ULONG

Default: 0x0000

2.2.3 mt90503_close_vc

This function closes the VC indicated by the handle **pvc_hndl**, regardless of the payload type of the VC (AAL2 or data). All resources that were reserved by the call to **mt90503_open_aal*x*_vc** are released. If the VC is an AAL1 VC, all channels allocated to the VC will also be closed by this function.

The mt90503_close_vc_def function inserts default values into the MT90503_CLOSE_VC structure. The default value of a structure field is indicated following the field's description.

Usage

```
#include "mt90503_api.h"

ULONG mt90503_close_vc_def( MT90503_INSTANCE_API* pmt90503_api,
            MT90503_CLOSE_VC* pclose_vc );

ULONG mt90503_close_vc( MT90503_INSTANCE_API* pmt90503_api,
            MT90503_CLOSE_VC* pclose_vc );
```

Return Values

MT90503ER_GENERIC_OK	Indicates success.

Also see **Section "4.0 Return Codes"** for non-successful codes.

Parameters

pmt90503_api a pointer to an API instance structure of the chip.

pclose_vc pointer to an MT90503_CLOSE_VC structure. The definitions of the structure's elements are listed below.

2.2.3.1 Structure MT90503_CLOSE_VC

pvc_hndl a pointer to a single ULONG, containing the handle of the VC created by a call to mt90503_open_*xxxx*_vc. The value of the handle will be modified to a unique value for closed handles as a code check.

Direction: IN/IO Type: POINTER

Default: NULL

2.3 TDM Functions

2.3.1 mt90503_open_channel_in_vc

Adds a full-duplex 64 kps channel to an open CBR VC.

This function activates the inactive channel **channel_number** in the specified CBR VC. The specified TX TSST of the channel on the TDM bus will be routed by the TXSAR in the CBR VC's channel **channel_number**. The RXSAR will route this same channel to the specified RX TSST on the TDM bus.

The chip drives the RX TSST when this function is called.

The function reserves the memory needed for the channel's circular buffer in the data memory.

The directions TX and RX of the TSSTs are with respect to the UTOPIA ports. Thus, a TX TSST enters the chip on the TDM bus.

The TX and RX TSSTs are reserved for the channel. In the case where the specified VC is a multi-frame and CAS VC the two TSSTs must be on even numbered streams. Furthermore, the TSST that is on the following stream from the TX TSST is also reserved, and the same rule applies to the RX TSST. For example, if the specified VC is a multi-framing CAS VC and the requested TSSTs are:

tx_timeslot = 2;

tx_stream = 0;

rx_timeslot = 7;

rx_stream = 4;

then the following TSSTs will also be reserved:

tx_timeslot = 2;

tx_stream = 1;

rx_timeslot = 7;

rx_stream = 5;

A TSST can only be used by one channel, whether it is a CBR VC channel or a low-latency-loopback channel.

When multi-framing is not specified the following structure parameters are not interpreted:

rx_underrun_cas_pad_type
rx_underrun_cas_pad_value
tx_initial_cpu_cas_value
rx_initial_cpu_cas_value
ignore_cas_enable_bit

This function returns a handle by which the API identifies this channel.

The mt90503_open_channel_in_vc_def function inserts default values into the channel configuration structure MT90503_CBR_CH. The default value of a structure field is provided following that field's description.

Usage

```
#include “mt90503_api.h”

ULONG mt90503_open_channel_in_vc_def( MT90503_INSTANCE_API* pmt90503_api,
            MT90503_CBR_CH* pcbr_ch );

ULONG mt90503_open_channel_in_vc( MT90503_INSTANCE_API* pmt90503_api,
            MT90503_CBR_CH* pcbr_ch);
```

Return Values

MT90503ER_GENERIC_OK	Indicates success

Also see **Section "4.0 Return Codes"** for non-successful codes.

Parameters

pmt90503_api pointer to an API instance structure of the chip

pcbr_ch pointer to an MT90503_CBR_CH structure. The definitions of the structure’s elements are listed below.

2.3.1.1 Structure MT90503_CBR_CH

pch_hndl pointer to a single ULONG which returns the handle for the created channel. This handle is a unique value that identifies the channel in all future function calls affecting this channel. The user allocates the ULONG for the handle prior to a call to this function.

Direction: IN/OUT Type: POINTER

Default: NULL

vc_hndl the handle which was created by the call to mt90503_open_cbr_vc for the VC to which the requested TSSTs will be added.

Direction: IN Type: ULONG

Default: MT90503_INVALID_HANDLE

tx_timeslot 0 – 127 for stream frequency of 8 MHz
0 – 63 for stream frequency of 4 MHz
0 – 31 for stream frequency of 2 MHz

The timeslot of the TX TSST. Note that the directions TX and RX are with respect to the UTOPIA ports. Thus, a TX TSST enters the chip, and RX TSST exits the chip.

Direction: IN Type: ULONG

Default: MT90503_INVALID_TIMESLOT

tx_stream 0 – 31

The stream of the TX TSST.

Direction: IN Type: ULONG

Default: MT90503_INVALID_STREAM

rx_timeslot see **tx_timeslot**

Direction: IN Type: ULONG

Default: MT90503_INVALID_TIMESLOT

rx_stream see **tx_stream**

Direction: IN Type: ULONG

Default: MT90503_INVALID_STREAM

channel_number 0 – (**number_of_channels –** 1)

The zero-based index of the channel to open in the VC. The range is from 0 to the maximum number of channels the VC can support minus 1.

Direction: IN Type: ULONG

Default: MT90503_INVALID_CHANNEL_NUM

rx_underrun_pad_type see below for values.

If an underrun occurs this is the data byte that will be sent on the TDM bus.

MT90503_RX_PAD_UR_WITH_OLD_DATA
The data that was written to the buffer X frames prior to this frame remains untouched. If there is no multi-framing and CAS supported by the VC then the value of X is 128, 256, 512, or 1024, depending on the size of the circular buffers (which depends on the CDV supported by the VC). If multi-framing and CAS is supported then X has the value 4, 8, 16, or 32 multi-frames, depending on the size of the circular buffers.

MT90503_RX_PAD_UR_NULL_BYTE
The **null_byte** field programmed via the call to **mt90503_open**.

MT90503_RX_PAD_UR_SILENCE_PATTERN_A
A byte of the silence pattern A is written. This cannot be selected if the **silent_tone_length** field of the MT90503_CONF structure was set to 0 when **mt90503_open** was called.

MT90503_RX_PAD_UR_SILENCE_PATTERN_B
A byte of the silence pattern B is written. This cannot be selected if the **silent_tone_length** field of the MT90503_CONF structure was set to 0 when **mt90503_open** was called.

Direction: IN Type: ULONG

Default: MT90503_RX_PAD_UR_NULL_BYTE

rx_underrun_cas_pad_type see below for values.

The type of CAS padding that will be sent on the TDM bus if a CAS underrun occurs.

MT90503_PAD_UR_CAS_VALUE
Use the pad value in **rx_underrun_cas_pad_value**.

MT90503_PAD_UR_CAS_WITH_OLD_CAS
The CAS value that is present in the circular buffer is the one sent. This CAS value was written 4, 8, 16, or 32 multi-frames prior to the underrun. The number of multi-frames depends on the size of the circular buffers (which depends on the amount of CDV absorbed by the VC).

Direction: IN Type: ULONG

Default: MT90503_PAD_UR_CAS_VALUE

rx_underrun_cas_pad_value 4 bit field

The CAS value that will be sent on the TDM bus if a CAS underrun occurs and MT90503_PAD_UR_CAS_VALUE is selected for **rx_underrun_cas_pad_type**.

Direction: IN Type: ULONG

Default: 0xF

rx_initial_cpu_cas_value 4 bit field

The CAS value that is placed on the H100 bus by the CPU if the VC's **vc_support_of_cas** parameter requires it. This value can be changed via a call to **mt90503_change_rx_cpu_cas**.

Direction: IN Type: ULONG

Default: 0xF

tx_initial_cpu_cas_value 4 bit field

The CAS value that is inserted, by the CPU, into ATM cells assembled by the TXSAR if the VC's **vc_support_of_cas** parameter requires it. This value can be changed via a call to **mt90503_change_tx_cpu_cas**.

Direction: IN Type: ULONG

Default: 0xF

ignore_cas_enable_bit TRUE / FALSE

If TRUE the CAS enable bit on the TDM bus is ignored. The CAS value is latched once every 16/24 frames, depending on the **vc_cas_type** field of the VC.

Direction: IN Type: ULONG

Default: FALSE

2.3.2 mt90503_open_channel_in_loopback

This function opens a channel in low-latency-loopback from TDM bus to TDM bus. Low-latency-loopback implies that the data coming in on the TX TSST of the channel will be placed on the RX TSST of the channel with 2 frames of delay.

A handle to this channel is returned. This handle is necessary to reference the newly created channel in the future.

The directions TX and RX of the TSSTs are with respect to the UTOPIA ports. Thus, a TX TSST enters the chip, via the TDM bus.

A TSST can only be used once, whether it is in an XXPCM channel, an HDLC stream, or a low-latency-loopback channel.

This function returns a handle by which the API identifies this channel.

The mt90503_open_channel_in_loopback_def function inserts default values into the channel configuration structure MT90503_LLL_CH. The default value of a structure field is provided following that field's description.

Usage

```
#include “mt90503_api.h”

ULONG mt90503_open_ channel_in_loopback_def(
            MT90503_INSTANCE_API* pmt90503_api,
            MT90503_LLL_CH* plll_ch);

ULONG mt90503_open_ channel_in_loopback(
            MT90503_INSTANCE_API* pmt90503_api,
            MT90503_LLL_CH* plll_ch );
```

Return Values

`MT90503ER_GENERIC_OK`	Indicates success

Also see **Section "4.0 Return Codes"** for non-successful codes.

Parameters

pmt90503_api pointer to an API instance structure of the chip

plll_ch pointer to an MT90503_LLL_CH structure. The definitions of the structure's elements are listed below.

2.3.2.1 Structure MT90503_LLL_CH

pch_hndl pointer to a single ULONG which returns the handle for the created low-latency-loopback channel. This handle is a unique value that identifies the channel in all future function calls affecting this channel. The user allocates the ULONG for the handle.

Direction: IN/OUT Type: POINTER

Default: NULL

tx_timeslot 0 – 127 for stream frequency of 8 MHz
0 – 63 for stream frequency of 4 MHz
0 – 31 for stream frequency of 2 MHz

The timeslot of the TX TSST. Note that the directions TX and RX are with respect to the UTOPIA ports. Thus, a TX TSST enters the chip, and RX TSST exits the chip.

Direction: IN Type: ULONG

Default: MT90503_INVALID_TIMESLOT

tx_stream 0 – 31

The stream of the TX TSST.

Direction: IN Type: ULONG

Default: MT90503_INVALID_STREAM

rx_timeslot see **tx_timeslot**

Default: MT90503_INVALID_TIMESLOT

rx_stream see **tx_stream**

Default :MT90503_INVALID_STREAM

2.3.3 mt90503_close_channel

This function closes the channel indicated by **pch_hndl**, regardless of the type of the channel.

This function releases all resources that were reserved by the call to the function that opened the channel.

The mt90503_close_channel_def function inserts default values into the MT90503_CLOSE_CH structure. The default value of a structure field is indicated following the field's description.

Usage

```
#include "mt90503_api.h"

ULONG mt90503_close_channel_def( MT90503_INSTANCE_API* pmt90503_api,
            MT90503_CLOSE_CH* pclose_ch );

ULONG mt90503_close_channel( MT90503_INSTANCE_API* pmt90503_api,
            MT90503_CLOSE_CH* pclose_ch );
```

Return Values

MT90503ER_GENERIC_OK	Indicates success

Also see **Section "4.0 Return Codes"** for non-successful codes.

Parameters

pmt90503_api pointer to an API instance structure of the chip

pclose_ch pointer to an MT90503_CLOSE_CH structure. The definitions of the structure's elements are listed below.

2.3.3.1 Structure MT90503_CLOSE_CH

pch_hndl pointer to a single ULONG, containing the handle which was created by the call to the function which opened the channel. This handle is modified to a unique value for closed handles as a code check.

Direction: IN/IO Type: POINTER

Default: NULL

2.4 Statistics Functions

Unless otherwise noted detected conditions indicate events that have occurred since the previous read of the same set of statistics. (i.e. values are reset when read by a statistics function.) All counts (identified by the name ending in '**_cnt**' are only reset when the underlying entity is opened. (e.g. all counters returned by **mt90503_get_cbr_vc_statistics** are reset when the VC for which statistics are returned was opened.) These counters are free running for the existence of the underlying entity and may wrap.

2.4.1 mt90503_get_chip_statistics

This function fills an MT90503_CHIP_STATS structure with the current statistics for the chip. All statistics returned by this function are initialized (e.g. counters set to 0) by the function **mt90503_open**.

The mt90503_get_chip_statistics_def function inserts default values into the MT90503_CHIP_STATS structure. The default value of a structure field is indicated following the field's description.

Usage

```
#include "mt90503_api.h"

ULONG mt90503_get_chip_statistics_def( MT90503_INSTANCE_API* pmt90503_api,
            MT90503_CHIP_STATS* pchip_stats );

ULONG mt90503_get_chip_statistics( MT90503_INSTANCE_API* pmt90503_api,
            MT90503_CHIP_STATS* pchip_stats );
```

Return Values

MT90503ER_GENERIC_OK	Indicates success

Also see **Section "4.0 Return Codes"** for non-successful codes.

Parameters

pmt90503_api pointer to an API instance structure of the chip

pchip_stats pointer to an MT90503_CHIP_STATS statistics structure to be filled in by this routine. The definitions of the structure's elements are listed below. The user allocates this structure.

2.4.1.1 Statistics Structure MT90503_CHIP_STATS

num_data_vcs_open 0 – ??

The number of data VCs currently open.

Direction: OUT Type: ULONG

Default: 0

num_cbr_vcs_open 0 – ??

The number of open CBR VCs currently open.

Direction: OUT Type: ULONG

Default: 0

num_aal0_vcs 0 – ??

The number of open CBR VCs with payload type AAL0.

Direction: OUT Type: ULONG

Default: 0

num_aal5_vcs 0 – ??

The number of open CBR VCs with payload type AAL5.

Direction: OUT Type: ULONG

Default: 0

num_aal1_vcs 0 – ??

The number of open CBR VCs with payload type unstructured AAL1.

Direction: OUT Type: ULONG

Default: 0

num_paal1_vcs 0 – ??

The number of open CBR VCs with payload type structured AAL1. This only includes CBR VCs that support no multi-framing CAS.

Direction: OUT Type: ULONG

Default: 0

num_t1_cas_vcs 0 – ??

The number of open T1, strict multi-framing VCs. This includes all open CBR VCs that support multi-framing CAS and the type of multi-framing CAS used is T1.

Direction: OUT Type: ULONG

Default: 0

num_e1_cas_vcs 0 – ??

The number of open E1, strict multi-framing VCs. This includes all open CBR VCs that support multi-framing CAS and the type of multi-framing CAS used is E1.

Direction: OUT Type: ULONG

Default: 0

num_strict_mf_cas_tdm_atm_vcs 0 – ??

The number of currently open CBR VCs, that were opened by the **mt90503_open_cbr_vc** function with the **vc_support_of_cas** parameter set to MT90503_STRICT_MF_CAS_TDM_ATM in the MT90503_CBR_VC structure.

Direction: OUT Type: ULONG

Default: 0

num_not_strict_mf_cas_tdm_atm_vcs 0 – ??

The number of currently open CBR VCs, that were opened by the **mt90503_open_cbr_vc** function with the **vc_support_of_cas** parameter set to MT90503_NOT_STRICT_MF_CAS_TDM_ATM in the MT90503_CBR_VC structure.

Direction: OUT Type: ULONG

Default: 0

num_mf_cas_tdm_cpu_vcs 0 – ??

The number of currently open CBR VCs, that were opened by the **mt90503_open_cbr_vc** function with the **vc_support_of_cas** parameter set to MT90503_MF_CAS_TDM_CPU in the MT90503_CBR_VC structure.

Direction: OUT Type: ULONG

Default: 0

num_mf_cas_atm_cpu_vcs 0 – ??

The number of currently open CBR VCs, that were opened by the **mt90503_open_cbr_vc** function with the **vc_support_of_cas** parameter set to MT90503_MF_CAS_ATM_CPU in the MT90503_CBR_VC structure.

Direction: OUT Type: ULONG

Default: 0

num_mf_cas_tdm_atm_cpu_vcs 0 – ??

The number of currently open CBR VCs, that were opened by the **mt90503_open_cbr_vc** function with the **vc_support_of_cas** parameter set to MT90503_MF_CAS_TDM_ATM_CPU in the MT90503_CBR_VC structure.

Direction: OUT Type: ULONG

Default: 0

num_channels_in_vcs 0 – ??

The number of channels currently open in all CBR VCs.

Direction: OUT Type: ULONG

Default: 0

num_channels_in_loopback 0 – ??

The number of channels currently open in loopback.

Direction: OUT Type: ULONG

Default: 0

cmem_parity_error0_cnt 0 - ??

The number of parity errors detected on the bits [7:0] of the control memory data pins. This counter is an approximation. The count is constructed from a count of active interrupts indicating this error, and serviced by the API's ISR.

Direction: OUT Type: ULONG

Default: 0

cmem_parity_error1_cnt 0 - ??

The number of parity errors detected on the bits [15:8] of the control memory data pins. This counter is an approximation. The count is constructed from a count of active interrupts indicating this error, and serviced by the API's ISR.

Direction: OUT Type: ULONG

Default: 0

dmem_parity_error0_cnt 0 - ??

The number of parity errors detected on the bits [7:0] of the data memory data pins. This counter is an approximation. The count is constructed from a count of active interrupts indicating this error, and serviced by the API's ISR.

Direction: OUT Type: ULONG

Default: 0

dmem_parity_error1_cnt 0 - ??

The number of parity errors detected on the bits [15:8] of the control memory data pins. This counter is an approximation. The count is constructed from a count of active interrupts indicating this error, and serviced by the API's ISR.

Direction: OUT Type: ULONG

Default: 0

rxsar_fifo_cell_loss_cnt 0 - ??

The number of cell losses detected in the RX SAR input cell FIFO. This counter is an approximation. The count is constructed from a count of active interrupts indicating this error, and serviced by the API's ISR.

Direction: OUT Type: ULONG

Default: 0

txa_fifo_cell_loss_cnt 0 - ??

The number of cell losses detected in the TX A output cell FIFO. This counter is an approximation. The count is constructed from a count of active interrupts indicating this error, and serviced by the API's ISR.

Direction: OUT Type: ULONG

Default: 0

txb_fifo_cell_loss_cnt 0 - ??

The number of cell losses detected in the TX B input cell FIFO. This counter is an approximation. The count is constructed from a count of active interrupts indicating this error, and serviced by the API's ISR.

Direction: OUT Type: ULONG

Default: 0

txc_fifo_cell_loss_cnt 0 - ??

The number of cell losses detected in the TX C input cell FIFO. This counter is an approximation. The count is constructed from a count of active interrupts indicating this error, and serviced by the API's ISR.

Direction: OUT Type: ULONG

Default: 0

rxa_parity_error_cnt 0 - ??

The number of payload-byte parity errors detected in cells received on port RX A. This counter is an approximation. The count is constructed from a count of active interrupts indicating this error, and serviced by the API's ISR.

Direction: OUT Type: ULONG

Default: 0

rxb_parity_error_cnt 0 - ??

The number of payload-byte parity errors detected in cells received on port RX B. This counter is an approximation. The count is constructed from a count of active interrupts indicating this error, and serviced by the API's ISR.

Direction: OUT Type: ULONG

Default: 0

rxc_parity_error_cnt 0 - ??

The number of payload-byte parity errors detected in cells received on port RX C. This counter is an approximation. The count is constructed from a count of active interrupts indicating this error, and serviced by the API's ISR.

Direction: OUT Type: ULONG

Default: 0

u_phya_alarm_cnt 0 - ??

The number of PHY alarms detected on PHY A. This count is an approximation. The count is constructed from a count of active interrupts indicating this error, and serviced by the API's ISR.

Direction: OUT Type: ULONG

Default: 0

u_phyb_alarm_cnt 0 - ??

The number of PHY alarms detected on PHY B. This count is an approximation. The count is constructed from a count of active interrupts indicating this error, and serviced by the API's ISR.

Direction: OUT Type: ULONG

Default: 0

all_fifos_cell_loss_cnt 32 bit unsigned counter

The number of cells lost due to FIFO overflows. This includes the FIFOs to the RXSAR, UTOPIA TX port A, UTOPIA TX port B and, UTOPIA TX port C.

Direction: OUT Type: ULONG

Default: 0

rxa_cell_arrival_cnt[2] 64 bit unsigned counter

The number of cells received on UTOPIA RX port A. This does not include loopback cells or test data cells that were looped back on the port. Element 0 of the array contains bits [31;0] of the counter, and element 1 bits [63:32].

Direction: OUT Type: ULONG[2]

Default: 0

txa_cell_departure_cnt[2] 64 bit unsigned counter

The number of cells transmitted on UTOPIA TX port A. Element 0 of the array contains bits [31;0] of the counter, and element 1 bits [63:32].

Direction: OUT Type: ULONG[2]

Default: 0

rxb_cell_arrival_cnt[2] see rxa_cell_arrival_cnt

txb_cell_departure_cnt[2] see txa_cell_departure_cnt

rxc_cell_arrival_cnt[2] see rxa_cell_arrival_cnt

txc_cell_departure_cnt[2] see txa_cell_departure_cnt

txsar_cell_departure_cnt[2] 64 bit unsigned counter

The number of cells that have exited the TXSAR. This count also includes all cells sent from the TX data cell FIFO as well as loopback cells. Element 0 of the array contains bits [31;0] of the counter, and element 1 bits [63:32].

Direction: OUT Type: ULONG[2]

Default: 0

rxsar_cell_arrival_cnt[2] 64 bit unsigned counter

The number of cells that have entered the RXSAR. This count also includes all cells sent from the RX data cell FIFO as well as loopback cells. Element 0 of the array contains bits [31;0] of the counter, and element 1 bits [63:32].

Direction: OUT Type: ULONG[2]

Default: 0

rx_data_buffer_overflow TRUE / FALSE

If TRUE the RX data buffer has overflowed. See the **rx_data_buffer_size** configuration parameter.

Direction: OUT Type: ULONG

Default: FALSE

soft_rx_data_buffer_overflow TRUE / FALSE

If TRUE the soft RX data buffer has overflowed. See the **soft_rx_data_buffer_size** configuration parameter.

Direction: OUT Type: ULONG

Default: FALSE

cas_data_buffer_overflow TRUE / FALSE

If TRUE the RX data buffer has overflowed. See the **rx_data_buffer_size** configuration parameter.

Direction: OUT Type: ULONG

Default: FALSE

soft_cas_data_buffer_overflow TRUE / FALSE

If TRUE the soft RX data buffer has overflowed. See the **soft_rx_data_buffer_size** configuration parameter.

Direction: OUT Type: ULONG

Default :FALSE

rx_vc_event_buffer_overflow TRUE / FALSE

If TRUE the RX VC event buffer has overflowed. See the **rx_vc_event_buffer_size** configuration parameter.

Direction: OUT Type: ULONG

Default: FALSE

clk_recov_a_buffer_overflow TRUE / FALSE

If TRUE the clock recovery A buffer has overflowed. See the **clk_recov_a_buffer_size** configuration parameter.

Direction: OUT Type: ULONG

Default: FALSE

soft_clk_recov_a_buffer_overflow TRUE / FALSE

If TRUE the soft clock recovery A buffer has overflowed. See the **soft_clk-recov_a_buffer_size** configuration parameter.

Direction: OUT Type: ULONG

Default: FALSE

clk_recov_b_buffer_overflow see **clk_recov_a_buffer_overflow**

Default: FALSE

soft_clk_recov_b_buffer_overflow see **soft_clk_recov_a_buffer_overflow**

Default: FALSE

soft_console_buffer_overflow TRUE / FALSE

If TRUE the soft console buffer has overflowed. See the **soft_console_buffer_size** configuration parameter.

Direction: OUT Type: ULONG

Default: FALSE

chip_fatal_internal_error TRUE / FALSE

If true the chip has encountered a fatal error and will need to be reset to operate correctly. This will occur when the processing load required exceeds the capability of the mclk frequency. If the mclk frequency is set to it's maximum this error should not occur.

Direction: OUT Type: ULONG

Default: FALSE

chip_internal_error TRUE / FALSE

If true the chip has encountered a non fatal error but will need have some channels closed to resume proper operation. This will occur when the processing load required exceeds the capability of the mclk frequency. If the mclk frequency is set to it's maximum this error should not occur.

Direction: OUT Type: ULONG

Default: FALSE

chip_api_fatal TRUE / FALSE

The API has caused a fatal chip error and the chip will need to be reset to operate correctly. The value in **chip_api_diagnostic** should be reported to the vendor.

Direction: OUT Type: ULONG

Default: FALSE

chip_api_diagnostic 0 – 0xFFFFFFFF

Report this value to the vendor if **chip_api_fatal** is TRUE.

Direction: OUT Type: ULONG

Default: 0

excessive_errors TRUE / FALSE

The number of errors occurring exceeds the capacity of the chip to report them all. This can be caused by the mclk frequency being too low but is more likely caused by excessive errors being generated. Examples of conditions that will cause excessive errors are; the H100 clock is not maintaining stratum 4 compliance, or the ATM cell loss or bit error rate of a VC(s) exceeds 10^{-12}.

Direction: OUT Type: ULONG

Default: FALSE

2.4.2 mt90503_convert_chip_statistics_to_text

This function converts an MT90503_CHIP_STATS statistics structure to a text string. The MT90503_CHIP_STATS statistics structure is returned by the **mt90503_get_chip_statistics** function.

The mt90503_convert_chip_statistics_to_text_def function inserts default values into the MT90503_CONVERT_CHIP_STATS structure. The default value of a structure field is indicated following the field's description.

Usage

```
#include “mt90503_api.h”

ULONG mt90503_convert_chip_statistics_to_text_def(
            MT90503_INSTANCE_API* pmt90503_api,
            MT90503_CONVERT_CHIP_STATS* pconvert_chip_stats );

ULONG mt90503_convert_chip_statistics_to_text(
            MT90503_INSTANCE_API* pmt90503_api,
            MT90503_CONVERT_CHIP_STATS* pconvert_chip_stats );
```

Return Values

`MT90503ER_GENERIC_OK` Indicates success always

Parameters

pmt90503_api

pointer to an API instance structure of the chip

pconvert_chip_stats

pointer to an MT90503_CONVERT_CHIP_STATS structure to be filled in by this routine. The definitions of the structure's elements are listed below. The user allocates this structure.

2.4.2.1 Structure MT90503_CONVERT_CHIP_STATS

pchip_stats pointer to an MT90503_CHIP_STATS statistics structure to be converted to text. The definitions of the structure's elements are listed in the **mt90503_get_chip_statistics** function description.

Direction: IN/IN Type: POINTER

Default: NULL

pstring pointer to the returned text string. The required length of the string is defined by MT90503_CHIP_STATS_STRING_LENGTH (in bytes). The user allocates the string.

Direction: IN/OUT Type: POINTER

Default: NULL

2.4.3 mt90503_get_cbr_vc_statistics

This function fills an MT90503_VC_STATS structure with the current statistics for a CBR VC. All statistics returned by this function are initialized (e.g. counters set to 0) by the function **mt90503_open_cbr_vc**.

The returned configuration structure covers all MIB statistics as specified in af-vtoa-0078.000, and supplies additional configuration information programmed during the VC open. The statistics names as given in af-vtoa-0078.000 relate to the names in the structure as follows:

atmfCESReassCells	mib_rxstr_cell_cnt
atmfCESHdrErrors Counter32	mib_aal1_crc_err_cnt,
	mib_aal1_parity_err_cnt
atmfCESPointerReframes Counter32	mib_pbyte_absent_err_cnt,
	mib_pbyte_range_err_cnt,
	mib_pbyte_framing_err_cnt
atmfCESPointerParityErrors Counter32	mib_pbyte_parity_err_cnt
atmfCESAal1SeqErrors Counter32	mib_single_cell_loss_cnt,
	mib_multiple_cell_loss_cnt,
	mib_cell_misinserted_cnt
atmfCESLostCells Counter32	mib_single_cell_loss_cnt,
	mib_multiple_cell_loss_cnt
atmfCESMisinsertedCells Counter32	mib_cell_misinserted_cnt
atmfCESBufUnderflows Counter32	mib_slip_underrun_cnt
atmfCESBufOverflows Counter32	mib_slip_overrun_cnt

The mt90503_get_cbr_vc_statistics_def function inserts default values into the MT90503_VC_STATS structure. The default value of a structure field is indicated following the field's description.

Usage

```
#include “mt90503_api.h”

ULONG mt90503_get_vc_statistics_def( MT90503_INSTANCE_API* pmt90503_api,
            MT90503_VC_STATS* pvc_stats );

ULONG mt90503_get_vc_statistics( MT90503_INSTANCE_API* pmt90503_api,
            MT90503_VC_STATS* pvc_stats );
```

Return Values

MT90503ER_GENERIC_OK	Indicates success

Also see **Section "4.0 Return Codes"** for non-successful codes.

Parameters

pmt90503_api pointer to an API instance structure of the chip

pvc_stats pointer to an MT90503_VC_STATS statistics structure to be filled in by this routine. The definitions of the structure’s elements are listed below. The user allocates this structure.

2.4.3.1 Structure MT90503_VC_STATS

vc_hndlidentifier handle returned from the call to **mt90503_open_cbr_vc**.

Direction: IN Type: ULONG

Default: MT90503_INVALID_HANDLE

reset_statistics TRUE / FALSE

Resets the statistics counters for the indicated CBR VC after returning their current values.

Direction: OUT Type: ULONG

Default: FALSE

header see MT90503_CBR_VC struct of **mt90503_open_cbr_vc**

Direction: OUT Type: same

Default: MT90503_NULL_HEADER

rx_tx_utopia_port see MT90503_CBR_VC struct of **mt90503_open_cbr_vc**

Direction: OUT Type: same

Default: MT90503_INVALID_UTOPIA_PORT

loopback see MT90503_CBR_VC struct of **mt90503_open_cbr_vc**

Direction: OUT Type: same

Default: FALSE

number_of_channels see MT90503_CBR_VC struct of **mt90503_open_cbr_vc**

Direction: OUT Type: same

Default: 0

rx_normal_cell_routing see MT90503_CBR_VC struct of **mt90503_open_cbr_vc**

Direction: OUT Type: same

Default :0

rx_oam_cell_routing see MT90503_CBR_VC struct of **mt90503_open_cbr_vc**

Direction: OUT Type: same

Default: 0

vc_payload_type see MT90503_CBR_VC struct of **mt90503_open_cbr_vc**

Direction: OUT Type: same

Default: MT90503_INVALID_PAYLOAD_TYPE

vc_payload_size see MT90503_CBR_VC struct of **mt90503_open_cbr_vc**

Direction: OUT Type: same

Default: 0

vc_support_of_cas see MT90503_CBR_VC struct of **mt90503_open_cbr_vc**

Direction: OUT Type: same

Default: MT90503_NO_MF_CAS

vc_cas_type see MT90503_CBR_VC struct of **mt90503_open_cbr_vc**

Direction: OUT Type: same

Default: MT90503_INVALID_CAS_TYPE

maximum_cdv see MT90503_CBR_VC struct of **mt90503_open_cbr_vc**

Direction: OUT Type: same

Default: 0

wheel_number 0-14

The wheel that is being used to map the events of the VC.

Direction: OUT Type: ULONG

Default: MT90503_INVALID_WHEEL

rx_circular_buffer_size **{128, 256, 512, 1024}**

RX circular buffer size that is being used for this VC in order to absorb CDV (in bytes).

Direction: OUT Type: ULONG

Default: 256

cdv_monitored_cnt 32 bit unsigned counter

The number of times the cell delay variation (CDV) of the VC has been monitored for excess delay.

Direction: OUT Type: ULONG

Default: 0

cdv_absorbtion_buffer_min_fill -2047 – 2047

The minimum fill of the CDV absorption buffer observed on the VC (in frames). Values greater than the circular buffer size and smaller than zero indicate that slips occurred.

Direction: OUT Type: LONG

Default: 0

cdv_absorbtion_buffer_max_fill -2047 – 2047

The maximum fill of the CDV absorption buffer observed on the VC (in frames). Values greater than the circular buffer size and smaller than zero indicate that slips occurred.

Direction: OUT Type: LONG

Default: 0

monitored_cdv 0 – 4095

The CDV present on the VC during the last monitoring period (in frames).

Direction: OUT Type: ULONG

Default: 0

txstr_cell_cnt[2] 64 bit unsigned counter

The number of cells sent on the VC by the TXSAR. The array is to be interpreted as a 64-bit value. Element 0 of the array is the lower-order 32 bits of the value.

Direction: OUT Type: ULONG[2]

Default: 0

mib_rxstr_cell_cnt[2] 64 bit unsigned counter

The number of cells received on the VC by the RXSAR. The array is to be interpreted as a 64-bit value. Element 0 of the array is the lower-order 32 bits of the value.

Direction: OUT Type: ULONG[2]

Default: 0

mib_pbyte_absent_err_cnt 32 bit unsigned counter

The number of times a P-Byte was expected in a cell but not received.

Direction: OUT Type: ULONG

Default: 0

mib_pbyte_framing_err_cnt 32 bit unsigned counter

The number of times the P-Byte value in a cell did not match the expected P-Byte value.

Direction: OUT Type: ULONG

Default: 0

mib_pbyte_range_err_cnt 32 bit unsigned counter

The number of received P-Bytes that were out of range. A P-Byte is out of range if it is greater than the trunk size of the VC or if it points to payload bytes that are unused by the cell (i.e. the cells are partially filled).

Direction: OUT Type: ULONG

Default: 0

mib_pbyte_parity_err_cnt 32 bit unsigned counter

The number of parity errors detected on received P-Byte values.

Direction: OUT Type: ULONG

Default: 0

mib_aal1_crc_err_cnt 32 bit unsigned counter

The number of CRC errors detected on received AAL1 bytes.

Direction: OUT Type: ULONG

Default: 0

mib_aal1_parity_err_cnt 32 bit unsigned counter

The number of parity errors detected on received AAL1 bytes.

Direction: OUT Type: ULONG

Default: 0

vc_cut_time 32 bit unsigned counter

The time in milliseconds that the VC has been in the current cut state. If 0 the VC is not currently cut. See **cut_vc_detect_time** parameter in MT90503_CBR_VC structure. This field is not reset each time the statistics function is called. It is a counter that resets to 0 when the cut condition ceases and may wrap if the cut condition lasts long enough.

Direction: OUT Type: ULONG

Default: 0

vc_cut_total_time 32 bit unsigned counter

The total time in milliseconds the VC has been in the cut state since it was opened. See **cut_vc_detect_time** parameter in MT90503_CBR_VC structure. This field is only reset when a VC is opened.

Direction: OUT Type: ULONG

Default: 0

mib_slip_overrun_cnt 32 bit unsigned counter

The number of overruns detected on the VC since it was opened.

Direction: OUT Type: ULONG

Default: 0

mib_slip_underrun_cnt 32 bit unsigned counter

The number of underruns detected on the VC since it was opened.

Direction: OUT Type: ULONG

Default: 0

mib_cell_misinserted_cnt 32 bit unsigned counter

The number of mis-inserted cells. A cell is mis-inserted if its sequence number (in the AAL1 byte) is one less than the sequence number of the last received cell, and one greater than the sequence number of the second last received cell. For example, receiving cells with sequence numbers 3, 5, 4 in that order implies that the cell with sequence number 5 was mis-inserted.

Direction: OUT Type: ULONG

Default: 0

mib_multiple_cell_loss_cnt 32 bit unsigned counter

The number of multiple cell losses. A multiple cell loss occurs when the received cell's sequence number (in the AAL1 byte) is not equal to the last cell's sequence number plus 1, and not equal to the last cell's sequence number plus 2.

Direction: OUT Type: ULONG

Default: 0

mib_single_cell_loss_cnt 32 bit unsigned counter

The number of single cell losses. A single cell loss occurs when the received cell's sequence number (in the AAL1 byte) is equal to the last cell's sequence number plus 2.

Direction: OUT Type: ULONG

Default: 0

2.4.4 mt90503_convert_cbr_vc_statistics_to_text

This function converts an MT90503_VC_STATS statistics structure to a text string. The MT90503_VC_STATS statistics structure is returned by the **mt90503_get_vc_statistics** function.

The mt90503_convert_cbr_vc_statistics_to_text_def function inserts default values into the MT90503_CONVERT_VC_STATS structure. The default value of a structure field is indicated following the field's description.

Usage

```
#include "mt90503_api.h"

ULONG mt90503_convert_vc_statistics_to_text_def(
            MT90503_INSTANCE_API* pmt90503_api,
            MT90503_CONVERT_VC_STATS* pconvert_vc_stats );

ULONG mt90503_convert_vc_statistics_to_text(
            MT90503_INSTANCE_API* pmt90503_api,
            MT90503_CONVERT_VC_STATS* pconvert_vc_stats );
```

Return Values

`MT90503ER_GENERIC_OK` Indicates success always

Parameters

pmt90503_api pointer to an API instance structure of the chip

pconvert_vc_stats pointer to an MT90503_CONVERT_VC_STATS structure to be filled in by this routine. The definitions of the structure's elements are listed below. The user allocates this structure.

2.4.4.1 Structure MT90503_CONVERT_VC_STATS

pvc_stats pointer to an MT90503_CHIP_STATS statistics structure to be converted to text. The definition of the structure elements is provided in the **mt90503_get_vc_statistics** function description.

Direction: IN/IN Type: POINTER

Default: NULL

pstring pointer to the returned string. The required length of the string is defined by MT90503_VC_STATS_STRING_LENGTH (in bytes). The user allocates the string.

Direction: IN/OUT Type: POINTER

Default: NULL

2.5 Utility Functions

2.5.1 mt90503_get_handle_list

This function returns a list of handles of a certain type.

The mt90503_get_handle_list_def function inserts default values into the MT90503_HANDLE_REQUEST structure. The default value of a structure field is indicated following the field's description.

Usage

```
#include "mt90503_api.h"

ULONG mt90503_get_handle_list_def( MT90503_INSTANCE_API* pmt90503_api,
            MT90503_HANDLE_REQUEST* phandle_request );

ULONG mt90503_get_handle_list( MT90503_INSTANCE_API* pmt90503_api,
            MT90503_HANDLE_REQUEST* phandle_request );
```

Return Values

MT90503ER_GENERIC_OK	Indicates success

Also see **Section "4.0 Return Codes"** for non-successful codes.

Parameters

pmt90503_api pointer to an API instance structure of the chip.

phandle_request pointer to an MT90503_HANDLE_REQUEST structure that defines the list being requested. The user allocates this structure. The definitions of the structure's elements are listed below.

2.5.1.1 Structure MT90503_HANDLE_REQUEST

max_hndl1 – 2097152

Maximum number of handles to be returned in the handles list parameter, phndl_list.

Direction: IN Type: ULONG

Default: 0

hndl_type MT90503_HNDL_CBR_VC
MT90503_HNDL_CUT_CBR_VC
MT90503_HNDL_DATA_VC

MT90503_HNDL_CHANNEL_IN_VC
MT90503_HNDL_CHANNEL_IN_LOOPBACK

Defines the type of handle that is being requested.

Direction: IN Type: ULONG

Default: MT90503_INVALID_HANDLE_TYPE

num_valid_hndl 0 – max_hndl

This value is the number of valid handles returned. Note if the returned list is **max_hndl** handles long there may be more handles of the requested type.

Direction: OUT Type: ULONG

Default: 0

phndl_list Pointer to a list of ULONGs. The length of the list is max_hndl. This list will be filled by the function with all handles of the requested handle type. The user allocates this list.

Direction: IN/OUT Type: POINTER

Default: NULL

2.6 Diagnostics Functions

2.6.1 mt90503_get_h100_diagnostics

This function fills an MT90503_H100_DIAG structure with the current diagnostic information of the H100 bus of the chip.

The mt90503_get_h100_diagnostics_def function inserts default values into the MT90503_H100_DIAG structure. The default value of a structure field is indicated following the field's description.

Usage

```
#include "mt90503_api.h"

ULONG mt90503_get_h100_diagnostics_def(
            MT90503_INSTANCE_API* pmt90503_api,
            MT90503_H100_DIAG* ph100_diag );

ULONG mt90503_get_h100_diagnostics(
            MT90503_INSTANCE_API* pmt90503_api,
            MT90503_H100_DIAG* ph100_diag );
```

Return Values

`MT90503ER_GENERIC_OK` Indicates success

Also see **Section "4.0 Return Codes"** for non-successful codes.

Parameters

pmt90503_api pointer to an API instance structure of the chip

ph100_diag pointer to an MT90503_H100_DIAG structure to be filled in by this routine. The user allocates this structure.

2.6.1.1 Structure MT90503_H100_DIAG

h100_clk_a_bad TRUE / FALSE

If TRUE, the H.100 signal ct_c8_a has failed to comply with the H100 specification. This monitors the clock edges and will detect period violations of ± 35 ns from the 122 ns nominal specification to within the resolution of the mclk frequency.

Direction: OUT Type: ULONG

Default: FALSE

h100_clk_b_bad TRUE / FALSE

If TRUE, the H.100 signal ct_c8_b has failed to comply with the H100 specification.

Direction: OUT Type: ULONG

Default: FALSE

h100_frame_a_bad TRUE / FALSE

If TRUE, the H.100 signal ct_frame_a has failed to comply with the H100 specification. This monitors that the H100 frame signal and will detect a violation if it is not asserted once and only once every 1024 H100 bus clock cycles.

Direction: OUT Type: ULONG

Default: FALSE

h100_frame_b_bad TRUE / FALSE

If TRUE, the H.100 signal ct_frame_b has failed to comply with the H100 specification.

Direction: OUT Type: ULONG

Default: FALSE

h100_clk_a_bad_cnt 0 – ??

The number of times **h100_clk_a_bad** has transitioned from FALSE to TRUE.

Direction: OUT Type: ULONG

Default: 0

h100_clk_b_bad_cnt 0 – ??

The number of times **h100_clk_b_bad** has transitioned from FALSE to TRUE.

Direction: OUT Type: ULONG

Default: 0

h100_frame_a_bad_cnt 0 – ??

The number of times **h100_frame_a_bad** has transitioned from FALSE to TRUE.

Direction: OUT Type: ULONG

Default: 0

h100_frame_b_bad_cnt 0 – ??

The number of times **h100_frame_b_bad** has transitioned from FALSE to TRUE.

Direction: OUT Type: ULONG

Default: 0

bus_master MT90503_H100_MASTER_A
MT90503_H100_MASTER_B

Which clock is currently the master clock of the bus.

Direction: OUT Type: ULONG

Default: MT90503_H100_MASTER_A

bus_master_bad TRUE / FALSE

If TRUE, the bus master clock has failed to comply with the H100 specification.

Direction: OUT Type: ULONG

Default: FALSE

bus_backup MT90503_H100_BACKUP_A
MT90503_H100_BACKUP_B

Which clock is the current backup clock.

Direction: OUT Type: ULONG

Default: MT90503_H100_BACKUP_A

bus_backup_bad TRUE / FALSE

If TRUE, the backup clock has failed to comply with the H100 specification.

Direction: OUT Type: ULONG

Default: FALSE

master_mode see MT90503_H100_MASTER_PARMS structure of **mt90503_set_h100_master_mode**.

The master mode of the chip.

Direction: OUT Type: same

Default: MT90503_H100_MASTERA

slave_mode see MT90503_H100_SLAVE_PARMS structure of **mt90503_set_h100_slave_mode**.

The slave mode of the chip.

Direction: OUT Type: same

Default: MT90503_H100_TRACKA

2.6.2 mt90503_convert_h100_diagnostics_to_text

This function converts an MT90503_H100_DIAG structure to a text string. The MT90503_H100_DIAG structure is returned by the **mt90503_get_h100_diagnostics** function.

The mt90503_convert_h100_diagnostics_to_text_def function inserts default values into the MT90503_CONVERT_H100_DIAG structure. The default value of a structure field is indicated following the field's description.

Usage

```
#include "mt90503_api.h"

ULONG mt90503_convert_h100_diagnostics_to_text_def(
            MT90503_INSTANCE_API* pmt90503_api,
            MT90503_CONVERT_H100_DIAG* pconvert_h100_diag);

ULONG mt90503_convert_h100_diagnostics_to_text(
            MT90503_INSTANCE_API* pmt90503_api,
            MT90503_CONVERT_H100_DIAG* pconvert_h100_diag);
```

Return Values

MT90503ER_GENERIC_OK	Indicates success always

Parameters

pmt90503_api
pointer to an API instance structure of the chip

pconvert_h100_diag
pointer to an MT90503_CONVERT_H100_DIAG structure to be filled in by this routine. The definitions of the structure's elements are listed below. The user allocates this structure.

2.6.2.1 Structure MT90503_CONVERT_H100_DIAG

ph100_diag pointer to an MT90503_H100_DIAG structure to be converted to text. The definition of the structure elements is provided in the mt90503_get_h100_diagnostics function description.

Direction: IN/IN Type: POINTER

Default: NULL

pstring pointer to the returned string. The required length of the string is defined by MT90503_H100_DIAG_STRING_LENGTH (in bytes). The user allocates the string.

Direction: IN/OUT Type: POINTER

Default: NULL

2.6.3 mt90503_get_console_msgs

This function returns debug messages from the API in a text string. These messages include detail of errors and warnings.

The mt90503_get_console_msgs_def function inserts default values into the MT90503_CONSOLE_MSG structure. The default value of a structure field is indicated following the field's description.

Usage

```
#include "mt90503_api.h"

ULONG mt90503_get_console_msgs_def( MT90503_INSTANCE_API* pmt90503_api,
            MT90503_CONSOLE_MSG* pconsole_msg );

ULONG mt90503_get_console_msgs( MT90503_INSTANCE_API* pmt90503_api,
            MT90503_CONSOLE_MSG* pconsole_msg );
```

Return Values

MT90503ER_GENERIC_OK	Indicates success

Also see **Section "4.0 Return Codes"** for non-successful codes.

Parameters

pmt90503_api pointer to an API instance structure of the chip.

pconsole_msg pointer to an MT90503_CONSOLE_MSG structure to be filled in by this routine. The definitions of the structure's elements are listed below. The user allocates this structure.

2.6.3.1 Structure MT90503_CONSOLE_MSG

pstring pointer to the returned string. The required length of the string, in bytes, is **mt90503_console_buffer_size,** which was configured by **mt90503_open**. The user allocates the string.

Direction: IN/OUT Type: POINTER

Default: NULL

2.7 H100 Functions

2.7.1 mt90503_set_h100_master_mode

This function sets the role of the chip as bus master on the H100 bus.

The mt90503_set_h100_master_mode_def function inserts default values into the MT90503_H100_MASTER_PARMS structure. The default value of a structure field is indicated following the field's description.

Usage

```
#include "mt90503_api.h"

ULONG mt90503_set_h100_master_mode_def(
            MT90503_INSTANCE_API*   pmt90503_api,   MT90503_H100_MASTER_PARMS   *
            pmt90503_h100_master_parms );

ULONG     mt90503_set_h100_master_mode(     MT90503_INSTANCE_API*     pmt90503_api,
            MT90503_H100_MASTER_PARMS * pmt90503_h100_master_parms );
```

Return Values

`MT90503ER_GENERIC_OK`	Indicates success

Also see **Section "4.0 Return Codes"** for non-successful codes.

Parameters

pmt90503_api
pointer to an API instance structure of the chip

pmt90503_h100_master_parms
pointer to an MT90503_H100_MASTER_PARMS structure. This structure is allocated by the user. The definitions of the structure's elements are listed below.

2.7.1.1 Structure MT90503_H100_MASTER_PARMS

master_mode MT90503_H100_MASTERA
MT90503_H100_MASTERB
MT90503_H100_MASTERAB
MT90503_H100_BACKUPA
MT90503_H100_BACKUPB
MT90503_H100_DISABLED

Determines which H100 clocks the chip is to drive.

The "_MASTER" modes drive the corresponding ct_c8 and ct_frame signal(s) as well as the compatibility signals. The "_BACKUP" modes drive the corresponding ct_c8 and ct_frame signals; the ct_c8/ct_frame signals will be generated in phase with the master ct_c8/ct_frame signals in backup mode. The "_DISABLED" mode does not drive any clock or frame signals. The initial setting is "_DISABLED" when the **mt90503_open** function returns.

Direction: IN Type: ULONG

Default: MT90503_H100_MASTERA

2.7.2 mt90503_set_h100_slave_mode

This function sets the slave mode of the chip and determines the clock used by the chip to synchronize all data transfers on the H100 bus.

If the chip does a fallback onto another clock then data transfers will continue to be synchronized on the fallback clock until slaveship mode is set once again, regardless of the state of the chosen clock.

The mt90503_set_h100_slave_mode_def function inserts default values into the MT90503_H100_SLAVE_PARMS structure. The default value of a structure field is indicated following the field's description.

Usage

```
#include "mt90503_api.h"

ULONG mt90503_set_h100_slave_mode_def(
            MT90503_INSTANCE_API*   pmt90503_api,   MT90503_H100_SLAVE_PARAMS   *
            pmt90503_h100_slave_parms );

ULONG mt90503_set_h100_slave_mode(
            MT90503_INSTANCE_API*   pmt90503_api,   MT90503_H100_SLAVE_PARAMS   *
            pmt90503_h100_slave_parms );
```

Return Values

`MT90503ER_GENERIC_OK`	Indicates success

Also see **Section "4.0 Return Codes"** for non-successful codes.

Parameters

pmt90503_api
pointer to an API instance structure of the chip

pmt90503_h100_slave_parms
pointer to an MT90503_H100_SLAVE_PARMS structure. The definitions of the structure's elements are listed below.

2.7.2.1 Structure MT90503_H100_SLAVE_PARMS

slave_mode MT90503_H100_TRACKA
MT90503_H100_TRACKB
MT90503_H100_TRACKA_FALLBACKB
MT90503_H100_TRACKB_FALLBACKA
MT90503_H100_DISABLED

Determines how the chip is to synchronize its data transfers on the H100 bus.

The "_TRACK" modes with no "_FALLBACK" perform data transfers synchronized to the "_TRACK*x*" clock no matter the condition of that clock or associated frame signal. The "_FALLBACK" modes synchronize data transfers to the "_FALLBACK*x*" clock and associated frame signal if the "_TRACK*x*_" clock or associated frame signal is not behaving according to the H100 specification. The "_DISABLED" mode disables all transfers on the H100 bus. Thus, no data can be placed on the bus, and the received data is ignored. The initial setting is "_DISABLED" when the **mt90503_open** function returns.

Direction: IN Type: ULONG

Default: MT90503_H100_TRACKA_FALLBACKB

2.8 Data Cell Functions

2.8.1 mt90503_send_data_cell

This function transmits a CPU generated ATM cell. The cell is placed at the tail of the data cell FIFO of the TXSAR. Once the cell reaches the head of the FIFO the cell will be transmitted on the specified UTOPIA port.

The mt90503_send_data_cell_def function inserts default values into the MT90503_TX_DATA_CELL structure. The default value of a structure field is indicated following the field's description.

Usage

```
#include "mt90503_api.h"

ULONG mt90503_send_data_cell_def( MT90503_INSTANCE_API* pmt90503_api,
            MT90503_TX_DATA_CELL* ptx_data_cell );

ULONG mt90503_send_data_cell( MT90503_INSTANCE_API* pmt90503_api,
            MT90503_TX_DATA_CELL* ptx_data_cell );
```

Return Values

`MT90503ER_GENERIC_OK`
Indicates success

`MT90503ER_SEND_DATA_CELL_BUFFER_FULL`
`when there is no room in the send data cell buffer. The cell was not queued to be sent.`

Also see **Section "4.0 Return Codes"** for non-successful codes.

Parameters

pmt90503_api pointer to an instance structure of the chip

ptx_data_cell pointer to an MT90503_TX_DATA_CELL structure. The user allocates this structure. The definitions of the structure's elements are listed below.

2.8.1.1 Structure MT90503_TX_DATA_CELL

header 32 bit field

The header of the cell. Header fields are in the following order (starting from bit 31): GFC, VPI, VCI, PT, CLP.

Direction: IN Type: ULONG

Default: MT9503_NULL_HEADER

payload[12] 12 element array of 32 bit fields

An array of the 48 payload bytes of the cell. The payload bytes of the cell are arranged in the array as follows:

	b31 - b24	b23 - b16	b15 -b8	b7 - b0
payload[0]	payload byte 0	payload byte 1	payload byte 2	payload byte 3
payload[1]	payload byte 4	payload byte 5	payload byte 6	payload byte 7
…	…	…	…	…
payload[11]	payload byte 44	payload byte 45	payload byte 46	payload byte 47

Direction: IN Type: ULONG[12]

Default: 0

tx_utopia_port 0 or the OR of any or all of:
MT90503_PORTA
MT90503_PORTB
MT90503_PORTC

Indicates how the data cell is to be routed by the UTOPIA module. The cell can be broadcast, so the values can be ORed together. If set to 0, the cell will be discarded.

Direction: IN Type: ULONG

Default: MT90503_INVALID_UTOPIA_PORT

2.8.2 mt90503_send_test_cell

This function transmits a CPU generated ATM cell. The cell is placed at the tail of the data cell FIFO of the TXSAR. Once the cell reaches the head of the FIFO the cell will be treated as if it were received on the specified UTOPIA port (i.e. it will use the LUT for the specified port and the entry identified by the header to route the cell). This function can be used to test RX hardware or software functions of the system.

The mt90503_send_test_cell_def function inserts default values into the MT90503_TX_TEST_CELL structure. The default value of a structure field is indicated following the field's description.

Usage

```
#include "mt90503_api.h"

ULONG mt90503_send_test_cell_def( MT90503_INSTANCE_API* pmt90503_api,
            MT90503_TX_TEST_CELL* ptx_test_cell );

ULONG mt90503_send_test_cell( MT90503_INSTANCE_API* pmt90503_api,
            MT90503_TX_TEST_CELL* ptx_test_cell );
```

Return Values

`MT90503ER_GENERIC_OK`
Indicates success

`MT90503ER_SEND_DATA_CELL_BUFFER_FULL`
when there is no room in the send data cell buffer. The cell was not queued to be sent.

Also see **Section "4.0 Return Codes"** for non-successful codes.

Parameters

pmt90503_api pointer to an API instance structure of the chip

ptx_test_cell pointer to an MT90503_TX_TEST_CELL structure. The definitions of the structure's elements are listed below.

2.8.2.1 Structure MT90503_TX_TEST_CELL

header 32 bit field

The header of the cell. Header fields are in the following order (starting from bit 31): GFC, VPI, VCI, PT, CLP.

Direction: IN Type: ULONG

Default: MT90503_NULL_HEADER

payload[12] 12 element array of 32 bit fields

An array of the 48 payload bytes of the cell. The payload bytes of the cell are arranged in the array as follows:

	b31 - b24	b23 - b16	b15 -b8	b7 - b0
payload[0]	payload byte 0	payload byte 1	payload byte 2	payload byte 3
payload[1]	payload byte 4	payload byte 5	payload byte 6	payload byte 7
…	…	…	…	…
payload[11]	payload byte 44	payload byte 45	payload byte 46	payload byte 47

Direction: IN Type: ULONG[12]

Default: 0

rx_utopia_port MT90503_PORTA
MT90503_PORTB
MT90503_PORTC

Indicates which port's LUT will be used to treat the cell.

Direction: IN Type: ULONG

Default: MT90503_INVALID_UTOPIA_PORT

2.8.3 mt90503_receive_data_cell

This function retrieves the oldest received data cell. The cells are buffered in the SSRAM and/or an API maintained soft buffer in received order. See the **soft_rx_data_buffer_size** in the MT90503_CONF structure.

The mt90503_receive_data_cell_def function inserts default values into the MT90503_RX_DATA_CELL structure. The default value of a structure field is indicated following the field's description.

Usage

```
#include "mt90503_api.h"

ULONG mt90503_receive_data_cell_def( MT90503_INSTANCE_API* pmt90503_api,
            MT90503_RX_DATA_CELL* prx_data_cell );

ULONG mt90503_receive_data_cell( MT90503_INSTANCE_API* pmt90503_api,
            MT90503_RX_DATA_CELL* prx_data_cell );
```

Return Values

`MT90503ER_GENERIC_OK`
Indicates success

`MT90503ER_RECEIVE_DATA_CELL_BUFFER_EMPTY`
when there are no data cells in the received data cell buffer. The returned structure is invalid.

Also see **Section "4.0 Return Codes"** for non-successful codes.

Parameters

pmt90503_api pointer to an API instance structure of the chip

prx_data_cell pointer to an MT90503_RX_DATA_CELL structure. The user allocates this structure. The definitions of the structure's elements are listed below.

2.8.3.1 Structure MT90503_RX_DATA_CELL

reset_buffers TRUE / FALSE

If set to TRUE, the hardware and software buffers for the data cells will be emptied. When set to TRUE the function will not return a cell, and **more_cells** will be set to FALSE.

Direction: IN Type: ULONG

Default: FALSE

header 32 bit field

The header of the cell. Header fields are in the following order (starting from bit 31): GFC, VPI, VCI, PT, CLP.

Direction: OUT Type: ULONG

Default: MT90503_NULL_HEADER

payload[12] 12 element array of 32 bit fields

An array of the 48 payload bytes of the cell. The payload bytes of the cell are arranged in the array as follows:

	b31 - b24	b23 - b16	B15 -b8	b7 - b0
payload[0]	payload byte 0	payload byte 1	payload byte 2	payload byte 3
payload[1]	payload byte 4	payload byte 5	payload byte 6	payload byte 7
…	…	…	…	…
payload[11]	payload byte 44	payload byte 45	payload byte 46	payload byte 47

Direction: OUT Type: ULONG[12]

Default: 0

rx_utopia_port MT90503_PORTA
MT90503_PORTB
MT90503_PORTC

The UTOPIA port on which the cell was received.

Direction: OUT Type: ULONG

Default: MT90503_INVALID_UTOPIA_PORT

rx_cell_routing MT90503_CBR_VC_LUT_ENTRY
MT90503_DATA_VC_LUT_ENTRY
MT90503_UNKNOWN_CELL

Indicates how the cell was routed to the data cell FIFO. The cell can have been routed by the LUT entry of a CBR or data VC, or it can have been routed by the port if the cell was declared as unknown (see the **u_rx*p*_ncr** and **u_rx*p*_ocr** cell routing parameters in the MT90503_CONF structure). If a LUT entry routed the cell then the handle to the VC is contained in the **rx_vc_hndl** parameter.

Direction: OUT Type: ULONG

Default: MT90503_INVALID_ROUTING

rx_vc_hndl If the cell was routed to the data cell FIFO via a VC's LUT entry then this field contains the handle to that VC. See rx_cell_routing.

Direction: OUT Type: ULONG

Default: MT90503_INVALID_HANDLE

more_cells TRUE / FALSE

True if there are more cells buffered to be read.

Direction: OUT Type: ULONG

Default: FALSE

2.9 CAS Functions

2.9.1 mt90503_get_cas_change

This function retrieves the oldest CAS change message. The events are buffered in the SSRAM and/or an API maintained soft buffer in occurrence order. See the **soft_cas_data_buffer_size** in the MT90503_CONF structure.

The mt90503_get_cas_change_def function inserts default values into the MT90503_CAS_CHANGE structure. The default value of a structure field is indicated following the field's description.

Usage

```
#include "mt90503_api.h"

ULONG mt90503_get_cas_change_def( MT90503_INSTANCE_API* pmt90503_api,
            MT90503_CAS_CHANGE* pcas_change );

ULONG mt90503_get_cas_change( MT90503_INSTANCE_API* pmt90503_api,
            MT90503_CAS_CHANGE* pcas_change );
```

Return Values

`MT90503ER_GENERIC_OK`

Indicates success

`MT90503ER_CAS_CHANGE_BUFFER_EMPTY`
There are no CAS change messages in the buffer. The returned structure is invalid.

Also see **Section "4.0 Return Codes"** for non-successful codes.

Parameters

pmt90503_api pointer to an API instance structure of the chip

pcas_change pointer to an MT90503_CAS_CHANGE structure. The user allocates this structure. The definitions of the structure's elements are listed below.

2.9.1.1 Structure MT90503_CAS_CHANGE

reset_buffers TRUE / FALSE

If set to TRUE, the hardware and software buffers for the CAS changes will be emptied. When set to TRUE the function will not return a CAS change message, and **more_messages** will be set to FALSE.

Direction: IN Type: ULONG

Default: FALSE

ch_hndl identifier

The handle of the TDM channel containing the TSST that generated this CAS change message.

Direction: OUT Type: ULONG

Default: MT90503_INVALID_HANDLE

tx_rx MT90503_DIRECTION_TX
MT90503_DIRECTION_RX

The direction of TSST. Note that these directions are with respect to the UTOPIA ports. Thus, a TX TSST enters the chip on the TDM bus, and an RX TSST exits.

Direction: OUT Type: ULONG

Default: MT90503_DIRECTION_TX

timeslot 0 – 127 for stream frequency of 8 MHz
0 – 63 for stream frequency of 4 MHz
0 – 31 for stream frequency of 2 MHz

The timeslot of the TSST on which the CAS change occurred.

Direction: OUT Type: ULONG

Default: MT90503_INVALID_TIMESLOT

stream 0 – 31

The stream of the TSST on which the CAS change occurred.

Direction: OUT Type: ULONG

Default: MT90503_INVALID_STREAM

new_cas_value 4 bit field

The new CAS value.

Direction: OUT Type: ULONG

Default: MT90503_INVALID_CAS

old_cas_value 4 bit field

The previous CAS value.

Direction: OUT Type: ULONG

Default: MT90503_INVALID_CAS

more_messages TRUE / FALSE

True if there are more messages buffered to be read.

Direction: OUT Type: ULONG

Default: FALSE

2.9.2 mt90503_change_tx_cpu_cas

This function changes the CPU CAS value inserted in cells, in the TX direction, of the CBR VC to which the indicated channel is allocated. The current CAS value is changed to the value specified by **tx_cas_value**.

The mt90503_change_tx_cpu_cas_def function inserts default values into the MT90503_TX_CPU_CAS structure. The default value of a structure field is indicated following the field's description.

Usage

```
#include "mt90503_api.h"

ULONG mt90503_change_tx_cpu_cas_def( MT90503_INSTANCE_API* pmt90503_api,
            MT90503_TX_CPU_CAS* ptx_cpu_cas );

ULONG mt90503_change_tx_cpu_cas( MT90503_INSTANCE_API* pmt90503_api,
            MT90503_TX_CPU_CAS* ptx_cpu_cas );
```

Return Values

MT90503ER_GENERIC_OK	Indicates success

Also see **Section "4.0 Return Codes"** for non-successful codes.

Parameters

pmt90503_api pointer to an API instance structure of the chip

ptx_cpu_cas pointer to an MT90503_TX_CPU_CAS structure. The definitions of the structure's elements are listed below.

2.9.2.1 Structure MT90503_TX_CPU_CAS

ch_hndl identifier

The handle which was returned from the call to **mt90503_open_channel_in_vc** for the VC to which **tx_cas_value** will be applied.

Direction: IN Type: ULONG

Default: MT90503_INVALID_HANDLE

tx_cas_value 4 bit field

The new CAS value to be inserted in the TX direction of the VC.

Direction: IN Type: ULONG

Default: MT90503_INVALID_CAS

2.9.3 mt90503_change_rx_cpu_cas

This function changes the CPU CAS value that accompanies the data of the specified channel in the RX direction (i.e. exiting the chip) on the TDM bus. The current CAS value is changed to the value specified by **rx_cas_value**.

The mt90503_change_rx_cpu_cas_def function inserts default values into the MT90503_RX_CPU_CAS structure. The default value of a structure field is indicated following the field's description.

Usage

```
#include "mt90503_api.h"

ULONG mt90503_change_rx_cpu_cas_def( MT90503_INSTANCE_API* pmt90503_api,
            MT90503_RX_CPU_CAS* prx_cpu_cas );

ULONG mt90503_change_rx_cpu_cas( MT90503_INSTANCE_API* pmt90503_api,
            MT90503_RX_CPU_CAS* prx_cpu_cas );
```

Return Values

MT90503ER_GENERIC_OK	Indicates success

Also see **Section "4.0 Return Codes"** for non-successful codes.

Parameters

pmt90503_api pointer to an API instance structure of the chip

prx_cpu_cas pointer to an MT90503_RX_CPU_CAS structure. The definitions of the structure's elements are listed below.

2.9.3.1 Structure MT90503_RX_CPU_CAS

ch_hndl identifier

The handle which was returned from the call to **mt90503_open_channel_in_vc**. This handle is used to access the desired channel.

Direction: IN Type: ULONG

Default: MT90503_INVALID_HANDLE

rx_cas_value 4 bit field

The new CAS value to be inserted in the RX direction of the VC.

Direction: IN Type: ULONG

Default: MT90503_INVALID_CAS

2.9.3.2 mt90503_select_cas_source

This function allows the user to select the source of the CAS nibbles accompanying the voice on a specified channel. The source is selectable for both the RX and TX directions of the channel. The CAS nibbles can either be the nibble received from one interface and passed onto the other (i.e. from TDM to ATM in TX direction) or a user specified value. The nibble is changed via the **mt90503_change_tx_cpu_cas** and **mt90503_change_rx_cpu_cas** functions.

The mt90503_select_cas_source_def function inserts default values into the MT90503_CAS_SOURCE structure. The default value of a structure field is indicated following the field's description.

Usage

```
#include “mt90503_api.h”

ULONG mt90503_select_cas_source_def( MT90503_INSTANCE_API* pmt90503_api,
            MT90503_CAS_SOURCE* pcas_source );

ULONG mt90503_select_cas_source( MT90503_INSTANCE_API* pmt90503_api,
            MT90503_CAS_SOURCE* pcas_source );
```

Return Values

MT90503ER_GENERIC_OK	Indicates success

Also see **Section "4.0 Return Codes"** for non-successful codes.

Parameters

pmt90503_api pointer to an API instance structure of the chip

pcas_source pointer to an MT90503_CAS_SOURCE structure. The definitions of the structure’s elements are listed below.

2.9.3.3 Structure MT90503_CAS_SOURCE

ch_hndl identifier

The handle which was returned from the call to **mt90503_open_channel_in_vc**. This handle is used to access the desired channel.

Direction: IN Type: ULONG

Default: MT90503_INVALID_HANDLE

tx_cas_source

MT90503_SOURCED_CAS
MT90503_TDM_CAS
MT90503_UNMODIFIED

The source for cas nibbles inserted into outgoing ATM cells. The nibble can be generated by the chip or be latched from the TDM bus. Also, this field can be set so that the present settings are not modified.

Direction: IN Type: ULONG

Default: MT90503_UNMODIFIED

rx_cas_source

MT90503_SOURCED_CAS
MT90503_ATM_CAS
MT90503_UNMODIFIED

The source for cas nibbles driven on the TDM bus by the chip. The nibble can be generated by the chip or be taken from received ATM cells. Also, this field can be set so that the present settings are not modified.

Direction IN Type: ULONG

Default: MT90503_UNMODIFIED

2.10 Clock Recovery Functions

2.10.1 mt90503_get_clk_recovery_point

This function retrieves a clock recovery point from the API's soft buffer. A clock recovery point can be retrieved from either the A or B buffers. Each buffer can be configured to perform SRTS or Adaptive clock recovery. The configuration of each buffer is defined in the MT90503_CONF structure provided to the **mt90503_open** function.

The mt90503_get_clk_recovery_point_def function inserts default values into the MT90503_CLK_RECOV_PNT structure. The default value of a structure field is indicated following the field's description.

Usage

```
#include "mt90503_api.h"

ULONG mt90503_get_clk_recovery_point_def (
            MT90503_INSTANCE_API* pmt90503_api,
            MT90503_CLK_RECOV_PNT* pclk_recov_pnt );

ULONG mt90503_get_clk_recovery_point(
            MT90503_INSTANCE_API* pmt90503_api,
            MT90503_CLK_RECOV_PNT* pclk_recov_pnt );
```

Return Values

MT90503ER_GENERIC_OK	Indicates success

Also see **Section "4.0 Return Codes"** for non-successful codes.

Parameters

pmt90503_api pointer to an API instance structure of the chip

pclk_recov_pnt pointer to an MT90503_CLK_RECOV_PNT structure. The definitions of the structure's elements are listed below.

2.10.1.1 Structure MT90503_CLK_RECOV_PNT

reset_buffers TRUE/FALSE

If set to TRUE, the specified hardware and software buffers for the clock recovery points will be emptied. When set to TRUE the function will not return a point, and **more_points** will be set to FALSE. Which hardware and software buffers will be emptied is indicated by **buffer_select** and **srts_select**.

Direction: IN Type: ULONG

Default: FALSE

buffer_select MT90503_CLK_RECOV_BUF_A
MT90503_CLK_RECOV_BUF_B

Indicates which hardware and software buffers the function is to access.

Direction: IN Type: ULONG

Default: MT90503_CLK_RECOV_BUF_A

srts_select MT90503_REMOTE_SRTS
MT90503_LOCAL_SRTS

In the case where the selected buffer is used for SRTS clock recovery, this field indicates which SRTS nibble is requested: RX or local SRTS nibble.

Direction: IN Type: ULONG

Default: MT90503_REMOTE_SRTS

clk_recov_pnt[4] 4 element array of 32 bit fields

An array of bytes containing the fetched clock recovery point. The bytes are laid out in three different ways, depending on the type of clock recovery implemented by the selected buffer.

	b31 - b24	b23 - b16	b15 -b8	b7 - b0
payload[0]	pclk counter integer			
Payload[1]	Reserved		pclk counter fraction	
payload[2]	mclk counter			
payload[3]	cell counter			

Table 1 - Adaptive Clock Recovery Layout

	b31 – b4	b3 - b0
payload[0]	Reserved	nibble
payload[1]	Reserved	
payload[2]	Reserved	
payload[3]	Reserved	

Table 2 - RX SRTS Clock Recovery Layout

	b31 – b4	b3 - b0
payload[0]	Reserved	nibble
payload[1]	Reserved	
payload[2]	Reserved	
payload[3]	Reserved	

Table 3 - Local SRTS Clock Recovery Layout

Direction: OUT Type: ULONG[4]

Default: 0

more_points TRUE/FALSE

Indicates whether there are more clock recovery points of the specified type pending in either the chip’s buffer or the soft buffer maintained by the API.

Direction: IN Type: ULONG

Default: FALSE

2.11 GPIO Functions

2.11.1 mt90503_set_gpio_value

This function sets the output enable of **gpio_pin** to **gpio_oe** and the driven value of **gpio_pin** to **gpio_value**.

The mt90503_set_gpio_value_def function inserts default values into the MT90503_SET_GPIO_PARMS structure. The default value of a structure field is indicated following the field's description.

Usage

```
#include "mt90503_api.h"

ULONG mt90503_set_gpio_value_def( MT90503_INSTANCE_API* pmt90503_api,
            MT90503_SET_GPIO_PARMS* pset_gpio_parms );

ULONG mt90503_set_gpio_value( MT90503_INSTANCE_API* pmt90503_api,
            MT90503_SET_GPIO_PARMS* pset_gpio_parms );
```

Return Values

MT90503ER_GENERIC_OK	Indicates success

Also see **Section "4.0 Return Codes"** for non-successful codes.

Parameters

pmt90503_api pointer to an API instance structure of the chip

pset_gpio_parms pointer to an MT90503_SET_GPIO_PARMS structure. The definitions of the structure's elements are listed below.

2.11.1.1 Structure MT90503_SET_GPIO_PARMS

gpio_pin

MT90503_GPIO_INMO_D8
MT90503_GPIO_INMO_D9
. . .
MT90503_GPIO_INMO_D15

MT90503_GPIO_PHYA_RX_LED
MT90503_GPIO_PHYA_TX_LED
MT90503_GPIO_PHYB_RX_LED
MT90503_GPIO_PHYB_TX_LED

MT90503_GPIO_TXA_DATA8
MT90503_GPIO_TXA_DATA9
. . .
MT90503_GPIO_TXA_DATA15

MT90503_GPIO_TXB_DATA8
MT90503_GPIO_TXB_DATA9
. . .
MT90503_GPIO_TXB_DATA15

The pin to which the output enable and value are to be set.

Direction: IN Type: ULONG

Default: MT90503_INVALID_GPIO

gpio_oe TRUE/FALSE

If TRUE the output will be enabled. If FALSE the pin is tri-stated.

Direction: IN Type: ULONG

Default: FALSE

gpio_value 0 / 1

The value to be driven on the pin specified by **gpio_pin**. If the pin output is not enabled by **gpio_oe**, it will remain tri-stated regardless of the value set.

Direction: IN Type: ULONG

Default: 0

2.11.2 mt90503_get_gpio_value

This function returns the value, as well as the rise and fall values of the GPI/GPIO specified by **gpio_pin**.

The mt90503_get_gpio_value_def function inserts default values into the MT90503_GET_GPIO_PARMS structure. The default value of a structure field is indicated following the field's description.

Usage

```
#include "mt90503_api.h"

ULONG mt90503_get_gpio_value_def( MT90503_INSTANCE_API* pmt90503_api,
            MT90503_GET_GPIO_PARMS* pget_gpio_parms );

ULONG mt90503_get_gpio_value( MT90503_INSTANCE_API* pmt90503_api,
            MT90503_GET_GPIO_PARMS* pget_gpio_parms );
```

Return Values

`MT90503ER_GENERIC_OK`	Indicates success

Also see **Section "4.0 Return Codes"** for non-successful codes.

Parameters

pmt90503_api pointer to an API instance structure of the chip

pget_gpio_parms pointer to an MT90503_GET_GPIO_PARMS structure. The user allocates this structure. The definitions of the structure's elements are listed below.

2.11.2.1 Structure MT90503_GET_GPIO_PARMS

gpio_pin

MT90503_GPIO_INMO_A0
MT90503_GPIO_INMO_A1
. . .
MT90503_GPIO_INMO_A14

MT90503_GPIO_INMO_D8
MT90503_GPIO_INMO_D9
. . .
MT90503_GPIO_INMO_D15

MT90503_GPIO_PHYA_ALM
MT90503_GPIO_PHYB_ALM

MT90503_GPIO_PHYA_RX_LED
MT90503_GPIO_PHYA_TX_LED
MT90503_GPIO_PHYB_RX_LED
MT90503_GPIO_PHYB_TX_LED

MT90503_GPIO_TXA_DATA8
MT90503_GPIO_TXA_DATA9
. . .
MT90503_GPIO_TXA_DATA15

MT90503_GPIO_RXA_DATA8
MT90503_GPIO_RXA_DATA9
. . .
MT90503_GPIO_RXA_DATA15

MT90503_GPIO_TXB_DATA8
MT90503_GPIO_TXB_DATA9
. . .
MT90503_GPIO_TXB_DATA15

MT90503_GPIO_RXB_DATA8
MT90503_GPIO_RXB_DATA9
. . .
MT90503_GPIO_RXB_DATA15

Specifies the pin to be sampled.

Direction: IN Type: ULONG

Default: MT90503_INVALID_GPIO

input_value 0 / 1

The value currently received on **gpio_pin**.

Direction: OUT Type: ULONG

Default: 0

rise TRUE / FALSE

If TRUE, the pin has transitioned from ‘0’ to ‘1’ since the last time this function was called.

Direction: OUT Type: ULONG

Default: FALSE

fall TRUE / FALSE

If TRUE, the pin has transitioned from ‘1’ to ‘0’ since the last time this function was called.

Direction: OUT Type: ULONG

Default: FALSE

2.12 Interrupt Functions

See **Section "1.5 System Architecture"** for the flow of interrupt treatment.

The interrupts are divided into five categories:

Fatal – indicates that the chip has encountered a fatal error, and must be reset to operate correctly once again.

Data Error – indicates that the chip has detected an error that leads to bad data integrity. The chip and all connections will continue to operate.

Error – indicates that the chip has detected an error that must be handled by the user application. There is no recovery required by the chip, the severity and or recovery, if any, can only be determined by the application.

H100 Error – indicates an error with the H100 bus clock or frame signals. If the bus is configured for recovery, it will happen automatically and these interrupts are informational. If auto recovery is not configured these errors flag a continuous data integrity disruption on all voice connections if it is indicated on the current master clock or frame signal.

Alarm – indicates that a buffer in the control memory is half-full, or the buffer has an element that has been pending for more than the specified amount of time. An example of such a buffer is the RX data cell FIFO. See the rx_data_cell_fifo_stale_time, cas_change_fifo_stale_time and rx_vc_event_fifo_stale_time parameters.

API Sync – this interrupt is used by the API to maintain synchronization with the chip. This is provided for information only and there is no user action required. If disabled the API ISR must be called a minimum of every 20 sec. to prevent complete loss of synchronization which can lead to system failure if the API attempts any operations against the device. (i.e. upon loss of synchronization, channels will remain open and functioning but the API will no longer be able to open or close channels without the risk of corrupting the chip.)

The category to which an interrupt belongs is indicated by the interrupt's name's prefix.

The behavior of all of the chip's interrupts can be configured independently. An interrupt can be enabled or disabled. In the case where an interrupt is enabled, the interrupt can behave in one of three ways once it is active. The interrupt can remain active and will not be reset by the APIISR. The interrupt can be kept enabled and be reset immediately by the APIISR. Or, the interrupt can be reset and temporarily disabled by the APIISR. See **5.1.2 Interrupt Configuration Parameters**.

For the alarms, an additional interrupt configuration mode exists. The interrupt can be disabled, and the servicing of the FIFO to which the alarm is tied can also be disabled. That is, the APIISR will not service the FIFO, regardless of the state of the FIFO.

2.12.1 mt90503_interrupt_service_routine

It is to be called by the user provided function mt90503_access_apiisr to service interrupts. This function lies in the APIISR code entity (see **Section "1.5 System Architecture"**). Because this function can be called by both the OS ISR and the API code entity, accesses to the function must be serialized. This function will take the appropriate action to treat any active interrupts when called by the OS ISR.

All interrupts are enabled by the user via the mt90503_open function call. Disabled interrupts are still serviced by this routine but they will not generate a hardware interrupt on the interrupt pin of the device.

If the user wants to create an entirely polled system, all interrupts can be set to "disabled" and the user becomes responsible for calling this routine often enough for proper operation of the device.

This function will reset all conditions causing the interrupt such that the interrupt pin typically will be inactive when it returns.

The mt90503_interrupt_service_routine_def function can be used by the OS ISR to request typical APIISR operation. The function will insert the appropriate values into the fields of the MT90503_INT_STRUCT structure.

Usage

```
#include “mt90503_apiisr.h”

void mt90503_interrupt_service_routine(
            MT90503_INSTANCE_APIISR* pmt90503_apiisr,
            MT90503_INT_STRUCT* pint_strct );

void mt90503_interrupt_service_routine_def(
            MT90503_INSTANCE_APIISR* pmt90503_apiisr,
            MT90503_INT_STRUCT* pint_strct );
```

Parameters

pmt90503_apiisr a pointer to the MT90503_INSTANCE_APIISR structure of the chip to be serviced.

pint_strct a structure indicating the type of servicing to be performed. The parameter is used to differentiate between OS ISR calls and various API calls to this function. The API may need to perform tasks which are normally performed by the interrupt service routine. For example, the API may need to empty the chip’s RX data cell FIFO. Or, the API may need a resource which is kept in the APIISR instance structure (for example, retrieving a data cell from the software FIFO). Thus, this parameter permits the API to force certain operations to be performed by the ISR. Other parameters within the structure are used to provide information needed by the API ISR to perform an operation. These parameters are only used by the API code entity. The definitions of the structure’s elements are supplied below.

2.12.1.1 Structure MT90503_INT_STRUCT

The structure is composed of two sub-structures. The first is the structure used by the API to access the APIISR block via a communication pipe (see **Section "3.3.1 mt90503_access_apiisr"**). The second is used to retrieve indications, from the APIISR block, of events which were flagged by the chip.

ppipe_strct MT90503_PIPE_STRUCT

See **Section "3.3.1.1 Structure MT90503_PIPE_STRUCT"**.

Direction: IN/IO Type: POINTER

Default: NULL

pint_flags MT90503_INT_FLAGS

Pointer to a structure indicating the events flagged by the chip. See **Section "2.12.1.2 Structure MT90503_INT_FLAGS"**.

Direction: IN, IN/IO Type: POINTER

Default: NULL

2.12.1.2 Structure MT90503_INT_FLAGS

The following parameters indicate what events were detected during the operation of the ISR. All events in the list below are evaluated during every call of the ISR, with the exception of the clk recovery events. Because each clock recovery buffer, A and B, can be configured to support either adaptive or SRTS clock recovery methods, events are supplied for both. However, only one event can be set per buffer depending on the configuration of that buffer.

fatal_general TRUE / FALSE

If TRUE an internal fatal chip error has been detected.

Direction: OUT Type: ULONG

Default: FALSE

fatal_cmem_parity TRUE / FALSE

If TRUE a parity error has been detected on the control memory interface.

Direction: OUT Type: ULONG

Default: FALSE

data_err_dmem_parity TRUE / FALSE

If TRUE a parity error has been detected on the data memory interface.

Direction: OUT Type: ULONG

Default: FALSE

data_err_utopia_parity_a TRUE / FALSE

If TRUE a parity error has been detected on the receive direction of UTOPIA port A.

Direction: OUT Type: ULONG

Default: FALSE

data_err_utopia_parity_b TRUE / FALSE

If TRUE a parity error has been detected on the receive direction of UTOPIA port B.

Direction: OUT Type: ULONG

Default: FALSE

data_err_utopia_parity_c TRUE / FALSE

If TRUE a parity error has been detected on the receive direction of UTOPIA port C.

Direction: OUT Type: ULONG

Default: FALSE

data_err_scheduler_bw TRUE / FALSE

If TRUE the schedulers have run out of bandwidth. This indicates that the current wheel configuration has caused more events to complete within one frame than the TXSAR can treat. The wheel mappings would need to be modified to avoid future occurrences. The chip will continue to operate, but the schedulers will have skipped some frames.

Direction: OUT Type: ULONG

Default: FALSE

error_phy_alarm_a TRUE / FALSE

If TRUE PHY device A has generated an alarm via the phya_alm pin.

Direction: OUT Type: ULONG

Default: FALSE

error_phy_alarm_b TRUE / FALSE

If TRUE PHY B device has generated an alarm via the phyb_alm pin.

Direction: OUT Type: ULONG

Default: FALSE

error_rxsar_cell_loss TRUE / FALSE

If TRUE then one or many cells have been lost at the internal RX SAR cell FIFO of the UTOPIA module due to an overflow.

Direction: OUT Type: ULONG

Default: FALSE

error_txa_cell_loss TRUE / FALSE

If TRUE then one or many cells have been lost at the internal TX A cell FIFO of the UTOPIA module due to an overflow.

Direction: OUT Type: ULONG

Default: FALSE

error_txb_cell_loss TRUE / FALSE

If TRUE then one or many cells have been lost at the internal TX B cell FIFO of the UTOPIA module due to an overflow.

Direction: OUT Type: ULONG

Default: FALSE

error_txc_cell_loss TRUE / FALSE

If TRUE then one or many cells have been lost at the internal TX C cell FIFO of the UTOPIA module due to an overflow.

Direction: OUT Type: ULONG

Default: FALSE

error_cas_change_fifo TRUE / FALSE

If TRUE the CAS change message FIFO in the external control memory has overflowed, causing some CAS change messages to be lost.

Direction: OUT Type: ULONG

Default: FALSE

error_data_cell_fifo TRUE / FALSE

If TRUE the RX data cell FIFO in the external control memory has overflowed, causing some data cells to be lost.

Direction: OUT Type: ULONG

Default: FALSE

error_vc_event_fifo TRUE / FALSE

If TRUE the RX VC event FIFO in the external control memory has overflowed, causing some statistics to not be updated.

Direction: OUT Type: ULONG

Default: FALSE

error_clk_recov_a_adap_fifo TRUE / FALSE

If TRUE the clock recovery A FIFO in the external control memory has overflowed, causing some clock recovery points to be lost. This event will never be set if clock recovery A FIFO is configured to contain SRTS points.

Direction: OUT Type: ULONG

Default: FALSE

error_clk_recov_a_remote_fifo TRUE / FALSE

If TRUE the clock recovery A FIFO in the external control memory has overflowed, causing some clock recovery points to be lost. This event will never be set if clock recovery A FIFO is configured to contain adaptive points.

Direction: OUT Type: ULONG

Default: FALSE

error_clk_recov_a_local_fifo TRUE / FALSE

If TRUE the clock recovery A FIFO in the external control memory has overflowed, causing some clock recovery points to be lost. This event will never be set if clock recovery A FIFO is configured to contain adaptive points.

Direction: OUT Type: ULONG

Default: FALSE

error_clk_recov_b_adap_fifo see **error_clk_recov_b_adap_fifo**

Default: FALSE

error_clk_recov_b_remote_fifo see **error_clk_recov_b_remote_fifo**

Default: FALSE

error_clk_recov_b_local_fifo see **error_clk_recov_b_local_fifo**

Default: FALSE

h100_error_clk_a TRUE / FALSE

If TRUE the clock CT_C8_A is not behaving in accordance to the H100 specification.

Direction: OUT Type: ULONG

Default: FALSE

h100_error_clk_b TRUE / FALSE

If TRUE the clock CT_C8_B is not behaving in accordance to the H100 specification.

Direction: OUT Type: ULONG

Default: FALSE

h100_error_frame_a TRUE / FALSE

If TRUE the clock CT_FRAME_A is not behaving in accordance to the H100 specification.

Direction: OUT Type: ULONG

Default: FALSE

h100_error_frame_b TRUE / FALSE

If TRUE the clock CT_FRAME_B is not behaving in accordance to the H100 specification.

Direction: OUT Type: ULONG

Default: FALSE

alarm_cas_change_fifo TRUE / FALSE

If TRUE the CAS change FIFO has reached half of its fill, or a CAS change message has been pending in the external control memory for more than X μs since the last CAS change message was retrieved, The amount of time (X) is specified at startup in the parameter cas_change_fifo_stale_time.

Direction: OUT Type: ULONG

Default: FALSE

alarm_data_cell_fifo TRUE / FALSE

If TRUE the RX data cell FIFO has reached half of its fill, or a data cell has been pending in the external control memory for more than X μs since the last data cell was retrieved, The amount of time (X) is specified at startup in the parameter rx_data_cell_fifo_stale_time.

Direction: OUT Type: ULONG

Default: FALSE

alarm_rx_vc_event_fifo TRUE / FALSE

If TRUE the RX VC event FIFO has reached half of its fill, or such an event has been pending in the external control memory for more than X μs since the last event was retrieved, The amount of time (X) is specified at startup in the parameter rx_vc_event_fifo_stale_time. These events are serviced by the API to maintain statistics of the VCs that can be obtained by calls to the various statistics routines.

Direction: OUT Type: ULONG

Default: FALSE

alarm_clk_recov_a_adap_fifo TRUE / FALSE

If TRUE the clock recovery A FIFO has reached half of its fill. This event will never be set if clock recovery A FIFO is configured to contain SRTS points.

Direction: OUT Type: ULONG

Default: FALSE

alarm_clk_recov_a_remote_fifo TRUE / FALSE

If TRUE the clock recovery A FIFO has reached half of its fill. This event will never be set if clock recovery A FIFO is configured to contain adaptive points.

Direction: OUT Type: ULONG

Default: FALSE

alarm_clk_recov_a_local_fifo TRUE / FALSE

If TRUE the clock recovery A FIFO has reached half of its fill. This event will never be set if clock recovery A FIFO is configured to contain adaptive points.

Direction: OUT Type: ULONG

Default: FALSE

alarm_clk_recov_b_adap_fifo see **alarm_clk_recov_a_adap_fifo**

Default: FALSE

alarm_clk_recov_b_remote_fifo see **alarm_clk_recov_a_remote_fifo**

Default: FALSE

alarm_clk_recov_b_local_fifo see alarm_clk_recov_a_local_fifo

Default: FALSE

alarm_cut_vcs_detected TRUE / FALSE

If TRUE then there is at least one open CBR VC which is considered cut (i.e. no cells have been received on VC for more than the specified time). The function **mt90503_get_handle_list** can be called to determine which VCs are cut. This alarm will not generate an interrupt. The alarm is flagged, however, when the VC statistics are polled (i.e. a call to **mt90503_poll_vc_stats**).

Direction: OUT Type: ULONG

Default: FALSE

api_sync TRUE / FALSE

If TRUE the chip interrupted for purposes of maintaining synchronization with the API.

Direction: OUT Type: ULONG

Default: FALSE

2.12.2 mt90503_mask_interrupt

This function is to be used by the operating system's interrupt service routine. This function disables the chip's interrupt pin. When the chip generates an interrupt, the OS starts its interrupt service routine (see **Section "1.5 System Architecture"**). The API's ISR must be called to treat the interrupt. Either the OS calls the API's ISR directly from its ISR, or it defers the treatment of the ISR to a later time, and at a lower CPU priority level. In the latter case, the interrupt pin of the chip must be disabled until the current interrupt has been treated. This function serves this purpose. The function first performs a read to the chip's interrupt register to determine if the chip is the source of the interrupt (many devices can share the same interrupt line). If the chip is the source of the interrupt, the function performs a single write to the chip's interrupt register, which disables the interrupt pins from generating another interrupt for up to 16 ms. After the disable timer has expired the interrupt pin will function normally. If the conditions causing the original interrupt still exist or a new event has occurred, the chip will interrupt immediately

when the timer expires. The API's ISR will re-enable the interrupt pin when it completes allowing new interrupts to occur in potentially less than 16ms.

The mt90503_mask_interrupt_def function inserts default values into the MT90503_MASK_INT_PARMS structure. The default value of a structure field is indicated following the field's description.

Usage

```
#include "mt90503_apimi.h"

ULONG mt90503_mask_interrupt_def(MT90503_MASK_INT_PARMS* pmask_int_parms );

ULONG mt90503_mask_interrupt( MT90503_MASK_INT_PARMS* pmask_int_parms );
```

Return Values

MT90503ER_GENERIC_OK
: Indicates success.

MT90503ER_INT_NOT_ACTIVE
: Indicates that the interrupt of the pin was not active.

MT90503ER_INT_RW_ERROR
: Indicates that an error occurred while trying to read from / write to the chip.

Parameters

pmask_int_parms pointer to an MT90503_MASK_INT_PARMS structure. The definitions of the structure's elements are listed below.

2.12.2.1 Structure MT90503_MASK_INT_PARMS

user_chip_number 0 – ??

The chip identifier parameter provided to the mt90503_open function.
(see **Section "1.5 System Architecture"**).

Direction: IN Type: ULONG

Default: UNDEFINED

2.12.3 mt90503_configure_interrupts

This function is used to change the current configuration of interrupt servicing. Before calling this function the mt90503_configure_interrupts_def function should be called. This function will insert the current interrupt configuration into the MT90503_CONF_INTERRUPTS structure. From this, only the fields corresponding to the desired interrupts need be changed.

Usage

```
#include "mt90503_api.h"

void mt90503_configure_interrupts(
                        MT90503_INSTANCE_API* pmt90503_api,
                        MT90503_CONF_INTERRUPTS* pconf_interrupts );

void mt90503_configure_interrupts_def(
                        MT90503_INSTANCE_API* pmt90503_api,
                        MT90503_CONF_INTERRUPTS* pconf_interrupts );
```

Parameters

pmt90503_api pointer to the MT90503_INSTANCE_API structure of the chip for which the interrupts are to be reconfigured.

pconf_interrupts pointer to an interrupt configuration structure. See **5.3 Structure MT90503_CONF_INTERRUPTS**.

2.13 Polling Functions

These functions are called periodically by the user application. They are used to update and extend statistics counters in the chip. Thus, these functions must be called regularly to not allow the counters to wrap and lead to bad extended statistics counters.

2.13.1 mt90503_poll_chip_stats

This function is to be called by the user to update the internal extended copy of the chip counters. The extended counters must be updated on a regular basis so to not allow the chip counters to wrap.

The maximum allowable time between two calls to this function is 20 s. If this maximum time is exceeded an error will be returned by the function.

The statistics can be reset via the reset_statistics parameter.

The mt90503_poll_chip_stats_def function inserts default values into the fields of the MT90503_POLL_CHIP_STATS structure. The default value of a structure field is indicated below the field's description.

Usage

```
#include “mt90503_api.h”

ULONG mt90503_poll_chip_stats_def( MT90503_INSTANCE_API* pmt90503_api,
            MT90503_POLL_CHIP_STATS *ppoll_chip );

ULONG mt90503_poll_chip_stats( MT90503_INSTANCE_API* pmt90503_api,
            MT90503_POLL_CHIP_STATS *ppoll_chip );
```

Return Values

MT90503ER_GENERIC_OK	Indicates success

Also see **Section "4.0 Return Codes"** for non-successful codes.

Parameters

pmt90503_api a pointer to the MT90503_INSTANCE_API structure of the chip for which the stats are to be polled.

ppoll_chip a pointer to a chip statistics poll structure. The definitions of the elements of the structure are provided below.

2.13.1.1 Structure MT90503_POLL_CHIP_STATS

reset_statistics TRUE / FALSE

Resets the global chip statistics counters.

Direction: IN Type: ULONG

Default: FALSE

2.13.2 mt90503_poll_vc_stats

This function is to be called by the user to update the internal extended copy of the statistics counters for each CBR VC, and to monitor the current VC CDV. The extended counters must be updated on a regular basis to not allow the chip counters to wrap. If the counters wrap, the statistics become invalid. However, this has no other effect on the operation of the device. The rate at which VC stats must be polled to avoid losing counts is dependent on the amount of activity on a given VC.

The statistics of each VC can be reset via the reset_statistics parameter. If the function is called with this parameter set to TRUE then the statistics of all VCs are reset.

This function will update only the statistics of max_vc number of VCs during a call to limit the amount of processor resources required. The VCs for which the statistics are updated depend on the time elapsed between two calls to the function. The function can only update the statistics for max_vcs number of VCs during a call (when less VCs are open than max_vcs, only those will be serviced).

The mt90503_poll_vc_stats_def function inserts default values into the fields of the MT90503_POLL_VC_STATS structure. The default value of a structure field is indicated below the field's description.

Usage

```
#include "mt90503_api.h"

ULONG mt90503_poll_vc_stats_def( MT90503_INSTANCE_API* pmt90503_api,
            MT90503_POLL_VCS_STATS *ppoll_vcs );

ULONG mt90503_poll_vc_stats( MT90503_INSTANCE_API* pmt90503_api,
            MT90503_POLL_VCS_STATS *ppoll_vcs );
```

Return Values

`MT90503ER_GENERIC_OK`	Indicates success

Also see **Section "4.0 Return Codes"**for non-successful codes.

Parameters

pmt90503_api a pointer to the MT90503_INSTANCE_API structure of the chip on which the VC stats are to be polled.

ppoll_vcs a pointer to a VC statistics poll structure. The definitions of the elements of the structure are provided below.

2.13.2.1 Structure MT90503_POLL_VCS_STATS

reset_statistics TRUE / FALSE

Resets the statistics counters of all CBR VCs.

Direction: IN Type: ULONG

Default: FALSE

max_vcs 0 - 2048

the maximum number of VCs for which the statistics will be updated during one call to this function. If the value is greater than the number of CBR VCs opened, it will be reduced to that number.

Direction: IN Type: ULONG

Default: 2048

3.0 User Supplied Function Descriptions

The API functions make all accesses to the physical device through user supplied functions. This gives the user full control over how accesses to the device are accomplished in any given implementation. In order to write these access routines specific implementation details of the device are given here.

Details to be supplied.

The performance of the CPU accesses depends on the **cache_cpu_accesses** parameter of the MT90503_CONF structure. This parameter is provided to the **mt90503_open** function call at start up. If this parameter is set to TRUE then no caching of the CPU accesses will be done in the chip. This results in a higher average access time, but a lower worst-case access time. If the parameter is set to FALSE, then the CPU accesses are cached.

3.1 Write Functions

3.1.1 mt90503_driver_write_api, _apiisr, _osisr

Performs a single word write to the chip. Any error returned by this function is considered a fatal error. Two or three versions of the function are needed because the function may be accessed from two or three different software layers depending on the user system architecture, See **Section "1.5 System Architecture".** Thus, each function must have a different name, but the functionality remains identical.

Usage

```
#include “mt90503_apiud.h”

ULONG mt90503_driver_write_api( ULONG user_chip_number,
            MT90503_WRITE_PARMS* pwrite_parms );

ULONG mt90503_driver_write_apiisr(ULONG user_chip_number,
            MT90503_WRITE_PARMS* pwrite_parms );

ULONG mt90503_driver_write_osisr(ULONG user_chip_number,
            MT90503_WRITE_PARMS* pwrite_parms );
```

Return Values

`MT90503ER_GENERIC_OK` Indicates success

`MT90503ER_DRIVER_WRITE_FAILED`
return values from 0xFFFF0000-0xFFFF000F are reserved for write routine return values. This return value will be passed to by the API function to the calling user routine. Any error returned by this function is considered a fatal error.

Parameters

user_chip_number The chip identifier parameter provided to the mt90503_open function. (see **Section "1.5 System Architecture"**).

pwrite_parms pointer to an MT90503_WRITE_PARMS structure. The definitions of the structure’s elements are listed below.

3.1.1.1 Structure MT90503_WRITE_PARMS

write_address 0 – 0x007FFFFE

Start address of the word access. This is a byte address that always points to words and must be even.

Direction: IN Type: ULONG

Default: 0

write_data 16 bit field

The word to be written.

Direction: IN Type: ULONG

Default: 0

3.1.2 mt90503_driver_write_smear_api, _apiisr, osisr

Performs a write of a data word to multiple addresses of the chip. Any error returned by this function is considered a fatal error. Two or three versions of the function are needed because the function may be accessed from two or three different software layers depending on the user system architecture, See **Section "1.5 System Architecture"** Thus, each function must have a different name, but the functionality remains identical.

Usage

```
#include “mt90503_apiud.h”

ULONG mt90503_driver_write_smear_api( ULONG user_chip_number,
            MT90503_WRITE_SMEAR_PARMS* write_smear_parms );

ULONG mt90503_driver_write_smear_apiisr( ULONG user_chip_number,
            MT90503_WRITE_SMEAR_PARMS* write_smear_parms );

ULONG mt90503_driver_write_smear_osisr( ULONG user_chip_number,
            MT90503_WRITE_SMEAR_PARMS* write_smear_parms );
```

Return Values

`MT90503ER_GENERIC_OK` Indicates success

`MT90503ER_DRIVER_WRITE_FAILED`
return values from 0xFFFF0000-0xFFFF000F are reserved for write routine return values. This return value will be passed to by the API function to the calling user routine. Any error returned by this function is considered a fatal error.

Parameters

user_chip_number
The chip identifier parameter provided to the mt90503_open function. (see **Section "1.5 System Architecture"**).

pwrite_smear_parms
pointer to an MT90503_WRITE_SMEAR_PARMS structure. The definitions of the structure’s elements are listed below.

3.1.2.1 Structure MT90503_WRITE_SMEAR_PARMS

write_address 0 – 0x007FFFFE

Start address of the writes. This is a byte address that always points to words and must be even. This is the address of the first location to write to. For each subsequent word the address is incremented by two.

Direction: IN Type: ULONG

Default: 0

write_data 16 bit field

The word to be written.

Direction: IN Type: ULONG

Default: 0

auto_parity TRUE / FALSE

When true the write is to a structure which can generate parity automatically. If TRUE the **write_parity** pointer is null.

Direction: IN Type: ULONG

Default: FALSE

write_parity 2 bit field

The parity bits to be written. This buffer contains the parity bits to be written with the word if required (see **auto_parity**).

Direction: IN Type: ULONG

Default: 0

write_length 1 – ??

The number of locations to write the data to. (length in words).

Direction: IN Type: ULONG

Default: 0

we 2 bit field

The write enable bits to be applied to every write. This is a two-bit field. Bit 1 enables the writing of MSB byte of the data word, and bit 0 enables LSB byte. The enables are active high.

Direction: IN Type: ULONG

Default: 0

3.2 Read Functions

3.2.1 mt90503_driver_read_api, _apiisr, _osisr

Performs a single word read from the chip. Any error returned by this function is considered a fatal error. Two or three versions of the function are needed because the function may be accessed from two or three different software layers depending on the user system architecture, See **Section "1.5 System Architecture".** Thus, each function must have a different name, but the functionality remains identical.

Usage

```
#include “mt90503_apiud.h”

ULONG mt90503_driver_read_api( ULONG user_chip_number,
            MT90503_READ_PARMS* pread_parms );

ULONG mt90503_driver_read_apiisr( ULONG user_chip_number,
            MT90503_READ_PARMS* pread_parms );
```

```
ULONG mt90503_driver_read_osisr( ULONG user_chip_number,
            MT90503_READ_PARMS* pread_parms );
```

Return Values

MT90503ER_GENERIC_OK Indicates success

MT90503ER_DRIVER_READ_FAILED
return values from 0xFFFF0010-0xFFFF001F are reserved for read routine return values. This return value will be passed to by the API function to the calling user routine. Any error returned by this function is considered a fatal error.

Parameters

user_chip_number The chip identifier parameter provided to the mt90503_open function. (see **Section "1.5 System Architecture"**).

pread_parms pointer to an MT90503_READ_PARMS structure. The definitions of the structure's elements are listed below.

3.2.1.1 Structure MT90503_READ_PARMS

read_address 0 – 0x007FFFFE

Start address of the read. This is a byte address that always points to words and must be even. This is the address of the word to be read.

Direction: IN Type: ULONG

Default: 0

pread_data Pointer to a single ULONG to receive the read data.

Direction: IN/OUT Type: POINTER

Default: NULL

3.2.2 mt90503_driver_read_burst_api, _apiisr, _osisr

Performs a burst of reads from the chip. Any error returned by this function is considered a fatal error. Two or three versions of the function are needed because the function may be accessed from two or three different software layers depending on the user system architecture, See **Section "1.5 System Architecture".** Thus, each function must have a different name, but the functionality remains identical.

Usage

```
#include "mt90503_apiud.h"

ULONG mt90503_driver_read_burst_api( ULONG user_chip_number,
                        MT90503_READ_BURST_PARMS* pread_burst_parms );

ULONG mt90503_driver_read_burst_apiisr( ULONG user_chip_number,
                        MT90503_READ_BURST_PARMS* pread_burst_parms );

ULONG mt90503_driver_read_burst_osisr( ULONG user_chip_number,
                        MT90503_READ_BURST_PARMS* pread_burst_parms );
```

Return Values

MT90503ER_GENERIC_OK Indicates success

`MT90503ER_DRIVER_READ_FAILED`
return values from 0xFFFF0010-0xFFFF001F are reserved for read routine return values. This return value will be passed to by the API function to the calling user routine. Any error returned by this function is considered a fatal error.

Parameters

user_chip_number The chip identifier parameter provided to the mt90503_open function. (see **Section "1.5 System Architecture"**).

pread_burst_parms pointer to an MT90503_READ_BURST_PARMS structure. The definitions of the structure's elements are listed below.

3.2.2.1 Structure MT90503_READ_BURST_PARMS

read_address 0 – 0x007FFFFE

Start address of the burst. This is a byte address that always points to words and must be even. This is the address of the first word in the burst. For each subsequent word the address is incremented by two.

Direction: IN Type: ULONG

Default: 0

pread_data Pointer to a list of ULONGs to receive the read data. Each element is one word.

Direction: IN/OUT Type: POINTER

Default: NULL

read_length 1 – ??

Length of the **pread_data** (burst length in words).

Direction: IN Type: ULONG

Default: 0

3.2.3 mt90503_driver_read_debug_api, _apiisr, _osisr

Performs a burst of reads from the chip with parity. Any error returned by this function is considered a fatal error. Two or three versions of the function are needed because the function may be accessed from two or three different software layers depending on the user system architecture, See **Section "1.5 System Architecture".** Thus, each function must have a different name, but the functionality remains identical.

Usage

```
#include “mt90503_apiud.h”

ULONG mt90503_driver_read_debug_api( ULONG user_chip_number,
            MT90503_READ_DEBUG_PARMS* read_debug_parms );

ULONG mt90503_driver_read_debug_apiisr(ULONG user_chip_number,
            MT90503_READ_DEBUG_PARMS* read_debug_parms );

ULONG mt90503_driver_read_debug_osisr(ULONG user_chip_number,
            MT90503_READ_DEBUG_PARMS* read_debug_parms );
```

Return Values

`MT90503ER_GENERIC_OK` Indicates success

`MT90503ER_DRIVER_READ_FAILED`
return values from 0xFFFF0010-0xFFFF001F are reserved for read routine return values. This return value will be passed to by the API function to the calling user routine. Any error returned by this function is considered a fatal error.

Parameters

user_chip_number
The chip identifier parameter provided to the mt90503_open function. (see **Section "1.5 System Architecture"**).

pread_debug_parms
pointer to an MT90503_READ_DEBUG_PARMS structure. The definitions of the structure's elements are listed below.

3.2.3.1 Structure MT90503_READ_DEBUG_PARMS

read_address 0 – 0x007FFFFE

Start address of the burst. This is a byte address that always points to words and must be even. This is the address of the first word in the burst. For each subsequent word the address is incremented by two.

Direction: IN Type: ULONG

Default: 0

pread_data Pointer to a list of ULONGs to receive the read data. Each element is one word.

Direction: IN/OUT Type: POINTER

Default: NULL

pread_parity Pointer to a list of ULONGs to receive the associated parity of each word. Each element is 2 bits where the most significant bit is the parity of the associated word's most significant byte.

Direction: IN/OUT Type: POINTER

Default: NULL

read_length 1 – ??

Length of the **pread_data** and **pread_parity** buffers (burst length in words).

Direction: IN Type: ULONG

Default: 0

3.3 Interrupt Service Routine Called From API

The system architecture is depicted in **Section "1.5 System Architecture"**. As illustrated, the API's interrupt service routine can be accessed by both the OS's interrupt service routine or deferred procedure call, and other functions within the API.

3.3.1 mt90503_access_apiisr

Because the API's ISR is accessed by the OS in kernel space (in the case of the embedded system), the code for the API's ISR must lie in the kernel's space. However, the API in the user's space must also have access to the API's ISR. This routine serves as the bridge for the API from user space to kernel space.

The API calls this function, and passes to it a chip number (user_chip_number) and a pointer to a structure (ppipe_strct). This structure must be passed to the interrupt service routine, along with the pointer to the APIISR structure corresponding to the chip number provided. The APIISR instance structure pointer, can be retrieved from an instance structure pointer array maintained by the user, using the chip number as the index. For the structure passed via the pointer pint_strct, one of two actions must be taken:

- If the API's ISR is located in user space, then the memory pointed to by ppipe_strct is accessible, and the pointer can be passed directly.
- If the API's ISR is located in kernel space, then the memory pointed to by ppipe_strct is inaccessible because it points to memory in the user space. In this case, the contents of the structure are copied into a new kernel-space memory buffer. The pointer to this new buffer is passed to the interrupt service routine.

The definitions of the contents of the structure pointed to by pint_strct are provided in **Section "2.11.1.1 Structure MT90503_SET_GPIO_PARMS"**. Within the MT90503_INT_STRUCT structure is a void pointer pint_buf. This pointer points to a buffer of contiguous memory. The size of the buffer is indicated by pipe_buf_size (in bytes). The contents of this buffer must also be passed (copied if necessary) to the interrupt service routine.

Before the function terminates, the contents of the kernel space buffer must be copied back to the user space buffer passed to this function.

Some sample code implementing this piping mechanism follows:

```
ULONG mt90503_access_apiisr( ULONG user_chip_number,
                  MT90503_PIPE_STRUCT* puser_pipe_strct)

{

   MT90503_INSTANCE_APIISR*   pmt90503_apiisr;
   MT90503_PIPE_STRUCT        kernel_pipe_strct;
   void*                      puser_int_buf;
   ...
   // Copy the contents of the structure if the interrupt service routine
   // is located inkernel space.
   memcpy(&kernel_pipe_strct, puser_pipe_strct,
                        sizeof(MT90503_PIPE_STRUCT));

   // Copy the contents of the buffer pointer within the structure if the
   // API's ISR is located in kernel space.
   if (puser_pipe_strct->ppipe_buf_size > 0)
   {
            kernel_pipe_strct.ppipe_buf = malloc(puser_pipe_strct->ppipe_buf_size);
            memcpy(kernel_pipe_strct.ppipe_buf,
                  puser_pipe_strct->ppipe_buf,
                  puser_pipe_strct->ppipe_buf_size);
   }

   ...

   // Select the corresponding APIISR instance structure according to chip number.
   pmt90503_apiisr = user_pointer_array[user_chip_number];
   ...
   // Call the serialization function. This function will then call the API's ISR.
   apiisr_serialization(pmt90503_apiisr, pkernel_pipe_buf);
```

```
   // The function has returned, so copy back buffers to user space memory buffer.
   // Keep a copy of the pointer to the user's void buffer pointer.
   puser_pipe_buf = puser_pipe_strct->ppipe_buf;
   memcpy(puser_pipe_strct, &kernel_pipe_strct,
                        sizeof(MT90503_PIPE_STRUCT));
   puser_pips_strct->pint_buf = puser_pips_buf;
   if (kernel_pips_strct.pipe_buf_size > 0)
   {
                  memcpy(puser_pipe_strct->ppipe_buf,
                        kernel_pipe_strct.ppipe_buf,
                        kernel_pipe_strct.ppipe_buf_size);
                  free(kernel_pipe_strct.ppipe_buf);
   }

}
```

Once the buffers are copied, the mt90503_serialize_interrupt_service_routine function must be called, which in turn calls the API's ISR. See **Section "1.5 System Architecture"**.

Usage

```
   #include "mt90503_apiud.h"

   void mt90503_run_interrupt_service_routine(ULONG user_chip_number,
              MT90503_PIPE_STRUCT* ppipe_strct );
```

Parameters

user_chip_number The chip identifier parameter provided to the mt90503_open function. (see **Section "1.5 System Architecture"**)

ppipe_strct Pointer to an **MT90503_PIPE_STRUCT** structure indicating what servicing is to be performed by the APIISR. The definitions of the structure's fields are provided below.

3.3.1.1 Structure MT90503_PIPE_STRUCT

The following parameters are used to determine what operations are to be performed by the API's ISR.

isr_type MT90503_ISR_TYPE_NORMAL

Must be set to MT90503_ISR_TYPE_NORMAL by the OS ISR calling this function.

Direction: IN Type: ULONG

Default: MT90503_ISR_TYPE_NORMAL

result This field is used by the APIISR block to return the error code of the functions performed by it due to this pipe message. This field is set to MT90503ER_GENERIC_OK if no errors occurred.

Direction: OUT Type: ULONG

Default: MT90503ER_GENERIC_OK

ppipe_buf Pointer to a buffer of bytes (unsigned 8-bit fields) used by the APIISR code entity to perform the operations indicated by the isr_type parameter. The pointer points to a block of contiguous memory. This parameter is only used if isr_type is not set to MT90503_ISR_TYPE_NORMAL.

Direction: IN, IN/IO Type: POINTER

Default: NULL

pipe_buf_size 0 – ??

Indicates the number of bytes pointed to by ppipe_buf.

Direction: IN Type: ULONG

Default: 0

4.0 Return Codes

The description for error return codes can be found in the mt90503_def.h file of the API release.

Errors in the range 0x1000-0x1FFF indicate the API software has detected an internal fatal error. These errors should be reported to the vendor for resolution.

5.0 Configuration Structures

5.1 Structure MT90503_CONF

5.1.1 General Parameters

user_chip_number identifier

This number is carried down to the user-supplied read/write routines to distinguish which chip the API is servicing. This can be used as an array index of the chip to be serviced to retrieve the correct instance pointer. If only one chip is being serviced by the API, then this parameter can be ignored. See **Section "1.5 System Architecture"**.

Direction: IN Type: ULONG

Default: UNDEFINED

max_cbr_vc 1 – 2048

The maximum number of CBR VCs that this chip instance will open concurrently. This parameter is used for optimizing structure sizes.

Direction: IN Type: ULONG

Default: 2048

max_data_vc_a 1 – ??

The maximum number of data VCs that this chip instance will open concurrently on port A. This parameter is used for optimizing structure sizes. The maximum number of data VCs which can be opened on port A is determined by the number of bits used for header concatenation by the LUT. The maximum number is determined as follows:

$$max\_data\_vcs = 2^{(u\_rxa\_lut\_index\_vpi\_bits + u\_rxa\_lut\_index\_vci\_bits)}.$$

See parameters **u_rxa_lut_index_vpi_bits** and **u_rxa_lut_index_vci_bits**.

Direction: IN Type: ULONG

Default: 128

max_data_vc_b 1 – ??

The maximum number of data VCs that this chip instance will open concurrently on port B. This parameter is used for optimizing structure sizes. The maximum number of data VCs which can be opened on port B is determined by the number of bits used for header concatenation by the LUT. The maximum number is determined as follows:

$$max\_data\_vcs = 2^{(u\_rxb\_lut\_index\_vpi\_bits + u\_rxb\_lut\_index\_vci\_bits)}.$$

See parameters **u_rxb_lut_index_vpi_bits** and **u_rxb_lut_index_vci_bits**.

Direction: IN Type: ULONG

Default: 128

max_data_vc_c 1 – ??

The maximum number of data VCs that this chip instance will open concurrently on port C. This parameter is used for optimizing structure sizes. The maximum number of data VCs which can be opened on port C is determined by the number of bits used for header concatenation by the LUT. The maximum number is determined as follows:

$$max\_data\_vcs = 2^{(u\_rxc\_lut\_index\_vpi\_bits + u\_rxc\_lut\_index\_vci\_bits)}.$$

See parameters **u_rxc_lut_index_vpi_bits** and **u_rxc_lut_index_vci_bits**.

Direction: IN Type: ULONG

Default: 128

max_stream {4, 8, 16, 32}

The maximum number of H100 streams that this chip instance will allocate timeslots on concurrently. This parameter is used to allow the device to operate at lower clock frequencies. When less than 32 streams are specified, the most significant streams are remove first. For example, in **max_stream** = 4, ct_d[3:0] streams are used only.

Direction: IN Type: ULONG

Default: 32

tx_data_buffer_size 4 – 16384

The required size of TX data buffer contained in SSRAM (in units of one cell). The TX data buffer is filled by the **mt90503_send_data_cell** function and emptied by the device as bandwidth is available on the required UTOPIA port. If this buffer is full the **mt90503_send_data_cell** function will return an MT90503ER_SEND_DATA_CELL_BUFFER_FULL result. This parameter is used for optimizing structure sizes.

Direction: IN Type: ULONG

Default: 32

rx_data_buffer_size 4 – 16384

The required size of RX data buffer contained in SSRAM (in units of one cell). The RX data cell buffer is filled by received cells that are routed to the CPU interface. The device will assert an interrupt when the buffer is ½ full and the API will transfer the cells to the soft RX data buffer during interrupt servicing. The required size is determined by the maximum rate cells destined to the CPU interface are expected to be received and the amount of time after the interrupt is asserted until the API interrupt service routine is called by the user software. If this buffer

overflows cells will be dropped and the **rx_data_buffer_overflow** parameter will be set in the MT90503_CHIP_STATS structure returned by the function **mt90503_get_chip_statistics**. This parameter is used for optimizing structure sizes.

Direction: IN Type: ULONG

Default: 128

soft_rx_data_buffer_size 4 – 16384

The required size of the soft RX data buffer contained in program memory in cells. The soft RX data buffer is filled from the RX data buffer in SSRAM by the API interrupt service routine and emptied by user calls to the **mt90503_receive_data_cell** function. The required size of the buffer is determined by the rate cells destined to the CPU interface are expected to be received and the frequency of user software removing cells from the buffer. If this buffer overflows, cells will be dropped and the **soft_rx_data_buffer_overflow** parameter will be set in the MT90503_CHIP_STATS structure returned by the function **mt90503_get_chip_statistics**. This parameter is used for optimizing structure sizes. This soft buffer must be at least as large as the buffer in the device (**soft_rx_data_buffer_size** >= **rx_data_buffer_size**)

Direction: IN Type: ULONG

Default: 1024

cas_data_buffer_size 4 – 16384

The required size of CAS data buffer contained in SSRAM in CAS change events. The CAS change buffer is filled by CAS change events that occur on open channels. The device will assert an interrupt when the buffer is ½ full and the API will transfer the events to the soft CAS data buffer during interrupt servicing. The required size is determined by the maximum rate of CAS change events expected and the amount of time after the interrupt is asserted until the API interrupt service routine is called by the user software. If this buffer overflows CAS change events will be dropped and the **cas_data_buffer_overflow** parameter will be set in the MT90503_CHIP_STATS structure returned by the function **mt90503_get_chip_statistics**. This parameter is used for optimizing structure sizes.

Direction: IN Type: ULONG

Default: 256

soft_cas_data_buffer_size 4 – 16384

The required size of the soft CAS data buffer contained in program memory in CAS change events. The soft CAS data buffer is filled from the CAS change event data buffer in SSRAM by the API interrupt service routine and emptied by user calls to the **mt90503_get_cas_change** function. The required size of the buffer is determined by the rate of CAS change events expected to be received and the frequency of user software removing events from the buffer. If this buffer overflows CAS changes will be lost and the **soft_cas_data_buffer_overflow** parameter will be set in the MT90503_CHIP_STATS structure returned by the function **mt90503_get_chip_statistics**. This parameter is used for optimizing structure sizes. This soft buffer must be at least as large as the buffer in the device (**soft_cas_data_buffer_size** >= **cas_data_buffer_size**)

Direction: IN Type: ULONG

Default: 1024

rx_vc_event_buffer_size 4 – 16384

The required size of RX VC event buffer contained in SSRAM in VC events. The buffer is filled by events that occur on open CBR VCs. The device will assert an interrupt when the buffer is ½ full and the API will remove the events and update the CBR VC statistics during interrupt servicing. The required size is determined by the maximum rate of events expected and the amount of time after the interrupt is asserted until the API interrupt service routine is called by the user software. If this buffer overflows RX VC events will be dropped and the **rx_vc_event_buffer_overflow** parameter will be set in the MT90503_CHIP_STATS structure returned by the function **mt90503_get_chip_statistics**. This parameter is used for optimizing structure sizes.

Direction: IN Type: ULONG

Default: 1024

clk_recov_a_enb TRUE / FALSE

Whether clock recovery A FIFO is to be enabled. If set to TRUE, then the type of clock recovery points contained by the FIFO is indicated by **clk_recov_a_type**.

Direction: IN Type: ULONG

Default: FALSE

clk_recov_b_enb see **clk_recov_a_enb**

Default: FALSE

clk_recov_a_type MT90503_CLK_RECOV_ADAPTIVE
MT90503_CLK_RECOV_SRTS

The type of clock recovery points stored in buffer A. For Adaptive clock recovery, each point requires 16 bytes, and 2 bytes for SRTS.

Direction: IN Type: ULONG

Default: MT90503_CLK_RECOV_ADAPTIVE

clk_recov_b_type see **clk_recov_a_type**

Default: MT90503_CLK_RECOV_ADAPTIVE

clk_recov_a_buffer_size 4 – 65536

The required size of clock recovery buffer A contained in SSRAM in units of one clock recovery point. The size of a clock recovery point is determined by the type of clock recovery the buffer is used for (see **clk_recov_type_a**). Adaptive clock recovery requires 16 bytes per point, and 2 bytes for SRTS. The buffer is filled by points generated from cells received on the selected clock recovery VC. The selection of the VC is done when the VC is opened. The device will assert an interrupt when the buffer is ½ full and the API will transfer the points to the soft clock recovery A buffer during interrupt servicing. The required size is determined by the maximum rate cells on the selected VC are expected to be received and the amount of time after the interrupt is asserted until the API interrupt service routine is called by the user software. If this buffer overflows points will be dropped and the **clk_recov_a_buffer_overflow** parameter will be set in the MT90503_CHIP_STATS structure returned by the function **mt90503_get_chip_statistics**. This parameter is used for optimizing structure sizes.

Direction: IN Type: ULONG

Default: 128

clk_recov_b_buffer_size see **clk_recov_a_buffer_size**

Default: 128

soft_clk_recov_a_buffer_size 4 – 65536

The required size of the soft clock recovery A buffer contained in program memory, in clock recovery points. The soft clock recovery A buffer is filled from the clock recovery A buffer in SSRAM by the API interrupt service routine and emptied by user calls to the **mt90503_get_clk_recovery_point** function. The required size is determined by the maximum rate cells on the selected VC are expected to be received and the frequency of user software removing points from the buffer. If this buffer overflows clock recovery points will be lost and the **soft_clk_recov_a_buffer_overflow** parameter will be set in the MT90503_CHIP_STATS structure returned by the function **mt90503_get_chip_statistics**. This parameter is used for optimizing structure sizes. This soft buffer must be at least as large as the buffer in the device (**soft_clk_recov_a_buffer_size** >= **clk_recov_a_buffer_size**)

Direction: IN Type: ULONG

Default: 1024

soft_clk_recov_b_buffer_size see **soft_clk_recov_a_buffer_size**

Default: 1024

silent_tone_length 0 – 65536

The length of the chip's silent tone buffers, in bytes. The silent tones are used for padding TDM channels in the case of losses of data due to underruns. The silent tones are used if the channel is configured to use them. There are 2 silent tone buffers, A and B. This field indicates the size of each buffer. The contents of the silent tone buffers are specified via the field **psilent_tone_a** and **psilent_tone_b**. The length of each array is the value of this field. If this field is set to 0 then the fields **psilent_tone_a** and **psilent_tone_b** will be ignored.

Direction: IN Type: ULONG

Default: 0

psilent_tone_a pointer to bytes.

A pointer to an array of bytes (unsigned 8-bit fields). Each element of the array contains a byte of the silent tone A. The length of this array is specified by the field **silent_tone_length**. If **silent_tone_length** is set to 0 then this field is ignored. The memory containing the bytes can be discarded once **mt90503_open** has returned successfully.

Direction: IN/IN Type: POINTER

Default: NULL

psilent_tone_b pointer to bytes.

A pointer to an array of bytes (unsigned 8-bit fields). Each element of the array contains a byte of the silent tone B. The length of this array is specified by the field **silent_tone_length**. If **silent_tone_length** is set to 0 then this field is ignored. The memory containing the bytes can be discarded once **mt90503_open** has returned successfully.

Direction: IN/IN Type: POINTER

Default: NULL

soft_console_buffer_size 0 – 262144

The required size of the soft console message buffer contained in program memory in bytes. Messages are stored as text in a circular buffer. They are retrieved by the function **get_console_messages**. If this size is configured as 0 console messaging will be disabled. When the buffer becomes full newer messages are lost until it is emptied by the function **get_console_messages**.

Direction: IN Type: ULONG

Default: 16384

cpu_type

MT90503_INTEL_16BIT_NON_MUXED
MT90503_INTEL_16BIT_MUXED
MT90503_INTEL_8BIT_NON_MUXED
MT90503_INTEL_8BIT_MUXED

MT90503_MOTO_16BIT_NON_MUXED
MT90503_MOTO_16BIT_MUXED
MT90503_MOTO_8BIT_NON_MUXED
MT90503_MOTO_8BIT_MUXED

The CPU interface selected for the chip. The chip can interface with an Intel/Motorola CPU, with 8 or 16 data lines, and the possibility of multiplexing the data lines to carry both addresses and data.

Direction: IN Type: ULONG

Default: MT90503_INTEL_16BIT_NON_MUXED

upclk_freq 25000000 - 50000000

The frequency of upclk, in Hz.

Direction: IN Type: ULONG

Default: 40000000 (40 MHz)

mclk_freq 12500000 – 100000000

The frequency of mclk in Hz. Generated by feeding upclk into the PLL.

Direction: IN Type: ULONG

Default: 80000000 (80 MHz)

mclk_type

MT90503_MCLK_TYPE_TTL
MT90503_MCLK_TYPE_PECL

The type of clk used for mclk.

Direction: IN Type: ULONG

Default: MT90503_MCLK_TYPE_TTL

led_flash_freq 1 - 10

LED flashing frequency in Hz. This is used to indicate link activity.

Direction: IN Type: ULONG

Default: 2 (2 Hz)

write_cache_ena TRUE / FALSE

Configures whether the device will cache write accesses or not. Enabling the write cache will allow the device be up to 128 writes late. However, the write cache must be empty for a read access to be performed. In general if the device is being read using direct accesses this should be set to TRUE to avoid CPU access time-outs. If reads are indirect this parameter in general should be set to FALSE to increase the performance of the device. See **Section "3.0 User Supplied Function Descriptions"** for more information.

Direction: IN Type: ULONG

Default: TRUE

cdv_min_monitor_period 1-20000 ms

The minimum amount of time elapsed between two CDV monitoring periods. This parameter is applied to all CBR VCs. The CDV is monitored by calling the mt90503_poll_vc_stats function. If the mt90503_poll_vc_stats function is called more frequently than the specified period, the CDV monitoring statistics will not be updated. (i.e. MAX and MIN CDV will be monitored for at least the monitor period).

Direction: IN Type: ULONG

Default: 10000 (10 sec)

5.1.2 Interrupt Configuration Parameters

interrupt_period_granularity 10 - 1000 ms

The granularity of the specified minimum period for an internally generated interrupt, in ms. An interrupt can be disabled for a short period of time following its activation. If configured for a given interrupt this field indicates the granularity of time for the timeout period. For example, if 10 ms is chosen then an interrupt can be disabled for 10, 20, 30, … ms.

Direction: IN Type: ULONG

Default: 10 (10 ms)

interrupt_polarity MT90503_INT_ACTIVE_LOW_OC
MT90503_INT_ACTIVE_HIGH_OC

Polarity and active status of interrupt line 1. The line can be active high or low and is in tri-state (open collector) when not active. Interrupt line 2 is not used by the chip to signal interrupts.

Direction: IN Type: ULONG

Default: MT90503_INT_ACTIVE_LOW_OC

interrupt_configuration

see 5.3 Structure MT90503_CONF_INTERRUPTS.

Direction: IN Type: MT90503_CONF_INTERRUPTS

Default: see structure

cas_data_fifo_stale_time 0 – 1048575 us

The maximum time, in us, a CAS change message can remain pending in the CAS change FIFO before an interrupt is generated.

Direction: IN Type: ULONG

Default: 500 (500 us)

rx_data_fifo_stale_time 0 – 1048575 us

The maximum time, in us, a cell can remain pending in the RX data cell FIFO before an interrupt is generated.

Direction: IN Type: ULONG

Default: 500 (500 us)

rx_vc_event_fifo_stale_time 0 – 1048575 us

The maximum time, in us, a VC event message can remain pending in the RX VC event FIFO before an interrupt is generated.

Direction: IN Type: ULONG

Default: 500 (500 us)

5.1.3 Memory Configuration Parameters

mem_type

MT90503_MEM_TYPE_FLOWTHROUGH_ZBT
MT90503_MEM_TYPE_FLOWTHROUGH_SSRAM
MT90503_MEM_TYPE_PIPELINED_ZBT
MT90503_MEM_TYPE_PIPELINED_SSRAM

The type of memory used for the data and control memory.

Direction: IN Type: ULONG

Default: MT90503_MEM_TYPE_PIPELINED_ZBT

cmem_chip_size

MT90503_MEM_CHIP_SIZE_128KB
MT90503_MEM_CHIP_SIZE_256KB
MT90503_MEM_CHIP_SIZE_512KB
MT90503_MEM_CHIP_SIZE_1MB

Indicates the size of each memory chip of the control memory.

Direction: IN Type: ULONG

Default: MT90503_MEM_CHIP_SIZE_512KB

cmem_num_chips 1, 2

The number of memory chips used to form control memory. The total amount of memory used (**cmem_chip_size** * **cmem_num_chips**) must not exceed 1 Megabyte.

Direction: IN Type: ULONG

Default: 2

dmem_chip_size

MT90503_MEM_CHIP_SIZE_128KB
MT90503_MEM_CHIP_SIZE_256KB
MT90503_MEM_CHIP_SIZE_512KB
MT90503_MEM_CHIP_SIZE_1MB

Indicates the size of the memory chips of the data memory.

Direction: IN Type: ULONG

Default: MT90503_MEM_CHIP_SIZE_512KB

dmem_num_chips 1 - 4

The number of memory chips used to form data memory.

Direction: IN Type: ULONG

Default: 4

5.1.4 Utopia Port Physical Configuration Parameters

u_pa_address_ena TRUE / FALSE

If TRUE, UTOPIA port A (TX and RX) is configured to operate with address select lines as described in UTOPIA level 2 specification (PHY mode only). The address is determined by **u_pa_address**. Note that in this mode port A TX and RX address pins use the rxb_data[15:8] pins and the txb_data[15:14] pins so **u_txb_width** and **u_rxb_width** needs to be 8 bits. UTOPIA port A can be 8 or 16 bits wide. While **u_txa_multiphy** may be TRUE or FALSE, using UTOPIA level 2 addresses is only useful if it is set TRUE.

Direction: IN Type: ULONG

Default: FALSE

u_pa_address 0 - 30

This is the address used by port A when operating under UTOPIA level 2 multi-phy mode with address lines. (**u_pa_address_ena** = TRUE).

Direction: IN Type: ULONG

Default: 0

u_pa_enb TRUE / FALSE

Determines whether UTOPIA port A is enabled.

Direction: IN Type: ULONG

Default: TRUE

u_pa_clk_oe TRUE / FALSE

Determines whether the outputs of the signals txa_clk and rxa_clk are enabled.

Direction: IN Type: ULONG

Default: TRUE

u_txa_clk_select MT90503_UTOPIA_CLK_DIVIDER_1
MT90503_UTOPIA_CLK_DIVIDER_2
MT90503_UTOPIA_CLK_DIVIDER_3

Selects which clk divider circuit to use for txa_clk signal. See **5.1.3.1 Utopia Clock Divider Configuration Parameters**.

Direction: IN Type: ULONG

Default: MT90503_UTOPIA_CLK_DIVIDER_1

u_rxa_clk_select MT90503_UTOPIA_CLK_DIVIDER_1
MT90503_UTOPIA_CLK_DIVIDER_2
MT90503_UTOPIA_CLK_DIVIDER_3

Selects which clk divider circuit to use for rxa_clk signal. See **5.1.3.1 Utopia Clock Divider Configuration Parameters**.

Direction: IN Type: ULONG

Default: MT90503_UTOPIA_CLK_DIVIDER_1

u_txa_multiphy TRUE / FALSE

If TRUE, the data, parity, and soc lines are tri-stated when UTOPIA TX port A is not selected. If FALSE, these signals are always driven.

Direction: IN Type: ULONG

Default: FALSE

u_txa_sar_mode MT90503_PHY_LAYER
MT90503_ATM_LAYER

Determines whether UTOPIA TX port A is in PHY or ATM mode.

Direction: IN Type: ULONG

Default: MT90503_ATM_LAYER

u_rxa_sar_mode MT90503_PHY_LAYER
MT90503_ATM_LAYER

Determines whether UTOPIA RX port A is in PHY or ATM mode.

Direction: IN Type: ULONG

Default: MT90503_ATM_LAYER

u_txa_width MT90503_UTOPIA_PORT_WIDTH_8
MT90503_UTOPIA_PORT_WIDTH_16

Determines the number of data lines used on UTOPIA TX port A.

Direction: IN Type: ULONG

Default: MT90503_UTOPIA_PORT_WIDTH_8

u_rxa_width MT90503_UTOPIA_PORT_WIDTH_8
MT90503_UTOPIA_PORT_WIDTH_16

Determines the number of data lines used on UTOPIA RX port A.

Direction: IN Type: ULONG

Default: MT90503_UTOPIA_PORT_WIDTH_8

u_txa_led_conf MT90503_PHY_LED_CONF_GPIO
MT90503_PHY_LED_CONF_LED

Determines if the phya_tx_led pin is used to drive an LED or for GPIO.

Direction: IN Type: ULONG

Default: MT90503_PHY_LED_CONF_LED

u_rxa_led_conf MT90503_PHY_LED_CONF_GPIO
MT90503_PHY_LED_CONF_LED

Determines if the phya_rx_led pin is used to drive an LED or for GPIO.

Direction: IN Type: ULONG

Default: MT90503_PHY_LED_CONF_LED

u_pb_enb see **u_pa_enb**

Default: FALSE

u_pb_clk_oe see **u_pa_clk_oe**

Default: FALSE

u_txb_clk_select see **u_txa_clk_select**

Default: MT90503_UTOPIA_CLK_DIVIDER_2

u_rxb_clk_select see **u_rxa_clk_select**

Default: MT90503_UTOPIA_CLK_DIVIDER_2

u_txb_multiphy see **u_txa_multiphy**

Default: FALSE

u_txb_sar_mode see **u_txa_sar_mode**

Default: MT90503_ATM_LAYER

u_rxb_sar_mode see **u_rxa_sar_mode**

Default: MT90503_ATM_LAYER

u_txb_width see **u_txa_width**

Default: MT90503_UTOPIA_PORT_WIDTH_8

u_rxb_width see **u_rxa_width**

Default: MT90503_UTOPIA_PORT_WIDTH_8

u_txb_led_conf see **u_txa_led_conf**

Default: MT90503_PHY_LED_CONF_LED

u_rxb_led_conf see **u_rxa_led_conf**

Default: MT90503_PHY_LED_CONF_LED

u_pc_enb see **u_pa_enb**

Default: TRUE

u_pc_clk_oe see **u_pa_clk_oe**

Default: FALSE

u_txc_clk_select see **u_txa_clk_select**

Default: MT90503_UTOPIA_CLK_DIVIDER_3

u_rxc_clk_select see **u_rxa_clk_select**

Default: MT90503_UTOPIA_CLK_DIVIDER_3

u_txc_multiphy see **u_txa_multiphy**

Default: FALSE

u_txc_sar_mode see **u_txa_sar_mode**

Default: MT90503_PHY_LAYER

u_rxc_sar_mode see **u_rxa_sar_mode**

Default: MT90503_PHY_LAYER

5.1.4.1 Utopia Clock Divider Configuration Parameters

The UTOPIA clocks can be generated from any one of three clock divider circuits or be configured to receive the clock from the bus. The source clock and divisor of each clock divider circuit are specified in these parameters.

u_div1_clk_src

MT90503_UDIV_SRC_TXA_CLK
MT90503_UDIV_SRC_TXB_CLK
MT90503_UDIV_SRC_TXC_CLK
MT90503_UDIV_SRC_RXA_CLK
MT90503_UDIV_SRC_RXB_CLK
MT90503_UDIV_SRC_RXC_CLK
MT90503_UDIV_SRC_MCLK

Selects the source for the UTOPIA clock divider 1. The source can be: TX clock from UTOPIA ports A, B, or C; RX clock from UTOPIA ports A, B, or C; mclk of the chip. Care must be taken that the divider does not drive the same clock source that feeds it.

Direction: IN Type: ULONG

Default: MT90503_UDIV_SRC_TXC_CLK

u_div1_clk_div 1 - 16

Integer divisor for UTOPIA clock divider 1.

Direction: IN Type: ULONG

Default: 1

u_div1_clk_inv TRUE / FALSE

Inverts the output of UTOPIA clock divider 1.

Direction: IN Type: ULONG

Default: TRUE

u_div2_clk_src see **u_div1_clk_src**

Default: MT90503_UDIV_SRC_TXC_CLK

u_div2_clk_div see u_div1_clk_div

Default: 1

u_div2_clk_inv see **u_div1_clk_inv**

Default: TRUE

u_div3_clk_src see **u_div1_clk_src**

Default: MT90503_UDIV_SRC_TXC_CLK

u_div3_clk_div see **u_div1_clk_div**

Default: 1

u_div3_clk_inv see **u_div1_clk_inv**

Default: TRUE

5.1.5 UTOPIA Operational Characteristics Parameters

5.1.5.1 General

The general parameters apply to all UTOPIA ports.

u_null_cell_elim TRUE / FALSE

If TRUE, null cells (vpi=0,vci=0) received on any RX UTOPIA port will be eliminated upon reception. Cells associated with a loopback VC will not be eliminated.

Direction: IN Type: ULONG

Default: TRUE

u_hec_mask 8 bit field

This mask is XORed with the accumulated header CRC result to form the HEC value of all cells sent on a UTOPIA TX port.

Direction: IN Type: ULONG

Default: 0x55

u_lut_entry_size MT90503_LUT_ENTRIES_SIZE_32_BITS
MT90503_LUT_ENTRIES_SIZE_64_BITS

Determines the size of each LUT entry. LUT entries of 32 bits use less memory than 64 bit entries and thus provide the possibility of supporting a greater number of VCs. This reduction in size does not affect the routing of CBR or data VCs. However, header replacement cannot be performed on any VCs.

LUT entries of 64 bits need only be selected if header replacement must be performed.

Direction: IN Type: ULONG

Default: MT90503_LUT_ENTRIES_SIZE_64_BITS

u_phy_alarm_pol MT90503_PHY_ALARMS_DISABLED
MT90503_PHY_ALARMS_ACTIVE_HIGH
MT90503_PHY_ALARMS_ACTIVE_LOW

Determines the polarity of UTOPIA PHY alarms.

Direction: IN Type: ULONG

Default: MT90503_PHY_ALARMS_ACTIVE_HIGH

5.1.5.2 Cell Routing

u_txa_network_mask 28 bit field

This mask indicates which bits of a cell's GFC, VPI and VCI fields will be used to identify a VC connection by the network device connected to this UTOPIA port. All bits that are high will be used by the network device, and all bits that are low will be ignored.

This field is used by the API software to detect conflicts during the opening of a VC that will be exiting the chip through UTOPIA port A. If the requested header will not be unique to the network according to this mask the **mt90503_open_cbr_vc** function will be unsuccessful.

Direction: IN Type: ULONG

Default: 0x0000FFF

u_rxa_ncr 0, or the OR of any or all of:
MT90503_CELL_ROUTE_TXA
MT90503_CELL_ROUTE_TXB
MT90503_CELL_ROUTE_TXC
MT90503_CELL_ROUTE_DATA_FIFO

Determines how unknown non-OAM cells entering on UTOPIA RX port A will be routed. Cells can be routed to any or all of the ports by using the OR of the listed values. If set to 0, all unknown non-OAM cells will be discarded.

Direction: IN Type: ULONG

Default: 0

u_rxa_ocr 0, or the OR of any or all of:
MT90503_CELL_ROUTE_TXA
MT90503_CELL_ROUTE_TXB
MT90503_CELL_ROUTE_TXC
MT90503_CELL_ROUTE_DATA_FIFO

Determines how unknown OAM cells entering on UTOPIA RX port A will be routed. Cells can be routed to any or all of the ports by using the OR of the listed values. If set to 0, all unknown OAM cells will be discarded.

Direction: IN Type: ULONG

Default: 0

u_rxa_header_mask 28 bit field

This mask is used to determine which bits of a cell's GFC, VPI and VCI fields will be used with the **u_rxa_header_match** parameter to determine if a cell received on UTOPIA RX port A is known. If it is known then it will be passed to the LUT. Otherwise the cell will be routed according to the **u_rxa_ncr** and **u_rxa_ocr** fields.

The 28 bit field represents the GFC, VPI and VCI fields of the header where the GFC bits are the most significant. A "1" in a bit position indicates the corresponding bit position of the cell header should be compared with the corresponding bit of the **u_rxa_header_match** parameter. All compared bits must match the **u_rxa_header_match** parameter value to be passed to the LUT.

Direction: IN Type: ULONG

Default: 0xFFFF000

u_rxa_header_match 28 bit field

This parameter is used in conjunction with the **u_rxa_header_mask** parameter to determine the required value of selected bits of a cell's GFC, VPI and VCI fields for a cell received on UTOPIA RX port A to be passed to the LUT. If the corresponding bits of the GFC,VPI, or VCI of the received header do not match the value set in this parameter and the mis-matched bits are not masked by the **u_rxa_header_mask** parameter the cell is treated as unknown and routed via the **u_rxa_ncr** and **u_rxa_ocr** fields.

The 28 bit field represents the concatenated GFC, VPI and VCI fields of the header where the GFC bits are the most significant. Only the result of bits not masked by the **u_rxa_header_mask** parameter will determine if the cell should be passed to the LUT.

For example:

GFC | VPI | VCI (from cell header)	0010 10000000 00000000 10110010	0010 10000000 00000000 10110**1**10
u_rxa_header_match	0000 00000000 00000000 10110010	0000 00000000 00000000 1011**0**010
Match Result (1=mismatch)	0010 10000000 00000000 00000000	0010 10000000 00000000 0000**1**100
u_rxa_header_mask	1111 11000000 11111111 00000000	0000 00111111 00000000 1111**1**111
Masked result (1=mismatched cell)	0000 00000000 00000000 00000000	0000 00000000 00000000 0000**1**100
Result	Routed according to LUT entry	Routed as unknown cell

Direction: IN Type: ULONG

Default: 0x0000000

u_rxa_lut_index_vpi_bits 0-12

This parameter determines how many bits of the concatenated GFC and VPI fields are to be used to index the LUT entry for cells received on UTOPIA RX port A. The specified number of bits are selected from LSB to MSB of the 12 bit field formed by the concatenation of the GFC and VPI fields of the header where the GFC bits are the most significant. The selected bits of the cell header are used to form the index of the LUT entry. The LUT index can be a maximum of 16 bits so a maximum of 16 bits may be selected by the combination of this parameter and the **u_rxa_lut_index_vci_bits** parameter. The more total bits that are selected the larger the LUT structure is required to be. See **u_rxa_lut_index_vci_bits** for an example.

Direction: IN Type: ULONG

Default: 0

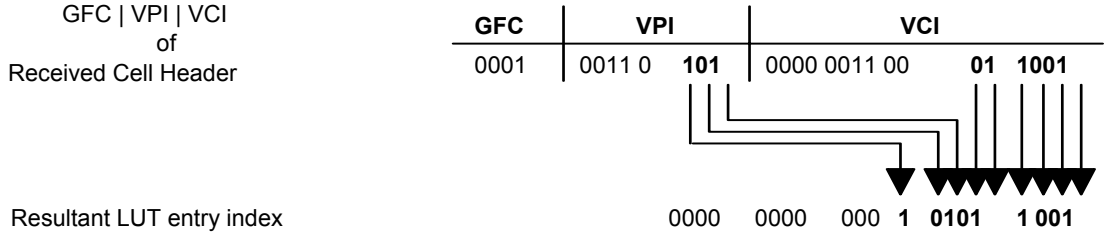
u_rxa_lut_index_vci_bits 0-16

This parameter determines how many bits of the VCI field are to be used to index the LUT entry for cells received on UTOPIA RX port A. The specified number of bits are selected from LSB to MSB of the 16 bit VCI field of the header. The selected bits of the cell header are used to form the index of the LUT entry. The LUT index can be a maximum of 16 bits so a maximum of 16 bits may be selected by the combination of this parameter and the **u_rxa_lut_index_vpi_bits** parameter. The more total bits that are selected the larger the LUT structure is required to be. See example.

Example LUT entry index:

u_rxa_lut_index_vpi_bits = 3

u_rxa_lut_index_vci_bits = 6



Direction: IN Type: ULONG

Default: 12

u_txb_network_mask see **u_txa_network_mask**

Default: 0x0000000

u_rxb_ncr see **u_rxa_ncr**

Default: 0

u_rxb_ocr see **u_rxa_ocr**

Default: 0

u_rxb_header_mask see **u_rxa_header_mask**

Default: 0xFFFFFFF

u_rxb_header_match see **u_rxa_header_match**

Default: 0x0000000

u_rxb_lut_index_vpi_bits see **u_rxa_lut_index_vpi_bits**

Default: 0

u_rxb_lut_index_vci_bits see **u_rxa_lut_index_vci_bits**

Default: 0

u_txc_network_mask see **u_txa_network_mask**

Default: 0x00003FF

u_rxc_ncr see **u_rxa_ncr**

Default: 0

u_rxc_ocr see **u_rxa_ocr**

Default: 0

u_rxc_header_mask see **u_rxa_header_mask**

Default: 0xFFFFC00

u_rxc_header_match see **u_rxa_header_match**

Default: 0x0000000

u_rxc_lut_index_vpi_bits see **u_rxa_lut_index_vpi_bits**

Default: 0

u_rxc_lut_index_vci_bits see **u_rxa_lut_index_vci_bits**

Default: 10

5.1.5.3 Flow Control

u_txa_rxa_cell_max 1 - 31
MT90503_NO_BACK_PRESSURE

If the cell fill of the UTOPIA TX port A output FIFO becomes greater than this value, cells from the UTOPIA RX port A input FIFO will be blocked. If MT90503_NO_BACK_PRESSURE is selected then the UTOPIA RX port A input FIFO will not be blocked by this FIFO (cells may be dropped if the TX FIFO is full).

Direction: IN Type: ULONG

Default: MT90503_NO_BACK_PRESSURE

u_txa_rxb_cell_max 1 - 31
MT90503_NO_BACK_PRESSURE

If the cell fill of the UTOPIA TX port A output FIFO becomes greater than this value, cells from the UTOPIA RX port B input FIFO will be blocked. If MT90503_NO_BACK_PRESSURE is selected then the UTOPIA RX port B input FIFO will not be blocked by this FIFO (cells may be dropped if the TX FIFO is full).

Direction: IN Type: ULONG

Default: MT90503_NO_BACK_PRESSURE

u_txa_rxc_cell_max 1 - 31
MT90503_NO_BACK_PRESSURE

If the cell fill of the UTOPIA TX port A output FIFO becomes greater than this value, cells from the UTOPIA RX port C input FIFO will be blocked. If MT90503_NO_BACK_PRESSURE is selected then the UTOPIA RX port C input FIFO will not be blocked by this FIFO (cells may be dropped if the TX FIFO is full).

Direction: IN Type: ULONG

Default: 4

u_txa_txsar_cell_max 1 - 31
MT90503_NO_BACK_PRESSURE

If the cell fill of the UTOPIA TX port A output FIFO becomes greater than this value, cells from the TX SAR output FIFO will be blocked. If MT90503_NO_BACK_PRESSURE is selected then UTOPIA TX port A input FIFO will not be blocked by this FIFO (cells may be dropped if the TX FIFO is full).

Direction: IN Type: ULONG

Default: 31

u_txb_rxa_cell_max see **u_txa_rxa_cell_max**

Default: MT90503_NO_BACK_PRESSURE

u_txb_rxb_cell_max see **u_txa_rxb_cell_max**

Default: MT90503_NO_BACK_PRESSURE

u_txb_rxc_cell_max see **u_txa_rxc_cell_max**

Default: MT90503_NO_BACK_PRESSURE

u_txb_txsar_cell_max see **u_txa_txsar_cell_max**

Default: MT90503_NO_BACK_PRESSURE

u_txc_rxa_cell_max see **u_txa_rxa_cell_max**

Default: MT90503_NO_BACK_PRESSURE

u_txc_rxb_cell_max see **u_txa_rxb_cell_max**

Default: MT90503_NO_BACK_PRESSURE

u_txc_rxc_cell_max see **u_txa_rxc_cell_max**

Default: MT90503_NO_BACK_PRESSURE

u_txc_txsar_cell_max see **u_txa_txsar_cell_max**

Default: MT90503_NO_BACK_PRESSURE

u_rxsar_rxa_cell_max see **u_txa_rxa_cell_max**

If the cell fill of the RX SAR input FIFO becomes greater than this value, cells from the UTOPIA RX port A input FIFO will be blocked. If MT90503_NO_BACK_PRESSURE is selected then the UTOPIA RX port A input FIFO will not be blocked by this FIFO (cells may be dropped if the TX FIFO is full).

Default: MT90503_NO_BACK_PRESSURE

u_rxsar_rxb_cell_max see **u_txa_rxb_cell_max**

Default: MT90503_NO_BACK_PRESSURE

u_rxsar_rxc_cell_max see **u_txa_rxc_cell_max**

Default: MT90503_NO_BACK_PRESSURE

u_rxsar_txsar_cell_max see **u_txa_txsar_cell_max**

Default: MT90503_NO_BACK_PRESSURE

5.1.6 TXSAR Scheduler Parameters

wheels[15] 15 element array of structure **MT90503_CONF_WHEEL**

This is an array of the configuration parameters for the 15 possible TX SAR Schedulers in the chip. Each TX SAR scheduler is controlled by a circular event list (or wheel). Every 125 μs a scheduler will complete all the events in a wheel frame.

Direction: IN Type:MT90503_CONF_WHEEL[15]

Default: see structure

num_mappings 0 – ???

Indicates the length of the **pmappings** parameter.

Direction: IN Type: ULONG

Default: 0

pmappings pointer to **MT90503_WHEEL_MAPPING** array

An array of CBR VC configurations for which the wheel mappings are to be determined at the time the **mt90503_open** function is called. The length of this array is determined by the **num_mappings** parameter. The description of each parameter of a node of the array is described in **Section "5.3 Structure MT90503_WHEEL_MAPPING"**. The memory pointed to is allocated by the user.

Direction: IN/IN Type: POINTER

Default: NULL

5.1.7 TDM configuration Parameters

null_byte 8 bit field

The byte with which the chip will pad if underruns occur and null byte padding is chosen in the TDM channel assignment structure.

Direction: IN Type: ULONG

Default: 0xFF

dstream_0_3_freq MT90503_TDM_STREAM_FREQ_2MHZ
MT90503_TDM_STREAM_FREQ_4MHZ
MT90503_TDM_STREAM_FREQ_8MHZ

The clock frequency of streams 0 - 3.

Direction: IN Type: ULONG

Default: MT90503_TDM_STREAM_FREQ_8MHZ

dstream_4_7_freq see **dstream_0_3_freq**

Default: MT90503_TDM_STREAM_FREQ_8MHZ

dstream_8_11_freq see **dstream_0_3_freq**

Default: MT90503_TDM_STREAM_FREQ_8MHZ

dstream_12_15_freq see **dstream_0_3_freq**

Default: MT90503_TDM_STREAM_FREQ_8MHZ

cas_enable_position 0 - 7

The bit position of the CAS enable bit in the TSST carrying CAS.

Direction: IN Type: ULONG

Default: 7

cas_enable_polarity MT90503_EN_ACTIVE_LOW
MT90503_EN_ACTIVE_HIGH

The polarity of the CAS enable bit.

Direction: IN Type: ULONG

Default: MT90503_EN_ACTIVE_HIGH

h100_sampling MT90503_H100_SAMPLE_AT_3_QUARTERS
MT90503_H100_SAMPLE_AT_RISING_EDGE
MT90503_H100_SAMPLE_AT_FALLING_EDGE

Determines at what point in the H.100 clock cycle a bit is sampled from the H100 bus.

Direction: IN Type: ULONG

Default: MT90503_H100_SAMPLE_AT_3_QUARTERS

h100_sclk_speed MT90503_SCLK_FREQ_2MHZ
MT90503_SCLK_FREQ_4MHZ
MT90503_SCLK_FREQ_8MHZ

The speed of sclk.

Direction: IN Type: ULONG

Default: MT90503_SCLK_FREQ_8MHZ

5.2 Structure MT90503_CONF_WHEEL

wheel_ena TRUE / FALSE

If TRUE this TX SAR scheduler is enabled and will attempt to process events.

Direction: IN Type: ULONG

Default: 0 => TRUE
1 => TRUE
2 => TRUE
3 => TRUE
4 => FALSE
5 => FALSE
6 => FALSE
7 => FALSE
8 => FALSE
9 => FALSE
10 => FALSE
11 => FALSE
12 => FALSE
13 => FALSE
14 => FALSE

wheel_num_events_per_frame MT90503_NUM_EVENTS_2
MT90503_NUM_EVENTS_4
MT90503_NUM_EVENTS_8
MT90503_NUM_EVENTS_16
MT90503_NUM_EVENTS_32
MT90503_NUM_EVENTS_64

The number of events that are to be mapped by this function and executed by the chip per frame of this wheel.

Direction: IN Type: ULONG

Default:
0 => MT90503_NUM_EVENTS_8
1 => MT90503_NUM_EVENTS_8
2 => MT90503_NUM_EVENTS_2
3 => MT90503_NUM_EVENTS_2
4 => UNDEFINED
5 => UNDEFINED
6 => UNDEFINED
7 => UNDEFINED
8 => UNDEFINED
9 => UNDEFINED
10 => UNDEFINED
11 => UNDEFINED
12 => UNDEFINED
13 => UNDEFINED
14 => UNDEFINED

wheel_num_frames 1 - 2048

The number of frames in the wheel.

In the case where E1 VCs are to be mapped, wheel_num_frames must be set to **MT90503_NUM_FRAMES_E1_CAS**. This will allocate 750 frames and set a special E1 mode for the wheel.

In the case where T1 VCs are to be mapped, wheel_num_frames must be set to **MT90503_NUM_FRAMES_T1_CAS**. This will allocate 1125 frames and set a special T1 mode for the wheel.

For structured AAL1 VCs with no CAS the value must be 375.

For unstructured AAL1, AAL0, and AAL5 VCs the following relationship must hold:

(wheel_num_frames * number_of_channels) = (n * payload_size)

where n is an integer.

Direction: IN Type: ULONG

Default:
0 => 48
1 => 375
2 => MT90503_NUM_FRAMES_E1_CAS
3 => MT90503_NUM_FRAMES_T1_CAS
4 => UNDEFINED
5 => UNDEFINED
6 => UNDEFINED
7 => UNDEFINED
8 => UNDEFINED
9 => UNDEFINED
10 => UNDEFINED
11 => UNDEFINED
12 => UNDEFINED
13 => UNDEFINED
14 => UNDEFINED

5.3 Structure MT90503_WHEEL_MAPPING

Every CBR VC requires a mapping in a scheduler wheel in external memory. Usually, this mapping is determined at the moment the **mt90503_open_cbr_vc** function is called. However, for VCs carrying CAS signaling bits, the determination of such a mapping is quite time consuming. To alleviate such heavy burden from the **mt90503_open_cbr_vc** function, several wheel mappings can be precalculated during the call to the **mt90503_open** function. The trade-off is between execution time of the open function, and memory requirements of the API instance structure.

Note that this structure is used to precalculate the mappings of structured AAL1 VCs (CAS or not) only. Calculating the wheel mapping of any other type of VC requires little calculation and thus is done at the time the **mt90503_open_cbr_vc** function is called.

vc_support_of_cas see MT90503_CBR_VC structure

Default: N/A

vc_cas_type see MT90503_CBR_VC structure

Default: N/A

number_of_channels see MT90503_CBR_VC structure

Default: N/A

5.4 Structure MT90503_CONF_INTERRUPTS

The following parameters determine which events will trigger an interrupt, and how that event will be treated by the API's ISR. See **Section "2.12.1.1 Structure MT90503_INT_STRUCT"** for descriptions of what the interrupts indicate.

fatal_general_conf MT90503_INT_DISABLE
MT90503_INT_NO_TIMEOUT
MT90503_INT_TIMEOUT

Indicates the configuration of the general fatal interrupt. The interrupt can be disabled from asserting the hardware interrupt pin (MT90503_INT_DISABLE). If the interrupt is enabled, it can behave in one of two ways once the interrupt has been treated. It can be reset and kept enabled (MT90503_INT_NO_TIMEOUT) or it can be cleared and disabled for a timeout period time (MT90503_INT_TIMEOUT). In the latter case, the timeout period is specified by the fatal_general_timeout parameter. When the specified timeout value is not an exact multiple of the interrupt_period_granularity it is rounded up to the next nearest multiple of interrupt_period_granularity before being applied. The configuration of this interrupt can be changed dynamically, see **Section "2.12.3 mt90503_configure_interrupts"**.

Direction: IN Type: ULONG

Default: MT90503_INT_NO_TIMEOUT

fatal_cmem_parity_conf see **fatal_general_conf**

Default: MT90503_INT_NO_TIMEOUT

data_err_dmem_parity_conf see **fatal_general_conf**

Default: MT90503_INT_TIMEOUT

data_err_utopia_parity_a_conf see **fatal_general_conf**

Default: MT90503_INT_TIMEOUT

data_err_utopia_parity_b_conf see **fatal_general_conf**

Default: MT90503_INT_TIMEOUT

data_err_utopia_parity_c_conf see **fatal_general_conf**

Default MT90503_INT_TIMEOUT

data_err_scheduler_bw_conf see **fatal_general_conf**

Default: MT90503_INT_TIMEOUT

error_phy_alarm_a_conf see **fatal_general_conf**

Default: MT90503_INT_TIMEOUT

error_phy_alarm_b_conf see **fatal_general_conf**

Default: MT90503_INT_TIMEOUT

error_rxsar_cell_loss_conf see **fatal_general_conf**

Default: MT90503_INT_TIMEOUT

error_txa_cell_loss_conf see **fatal_general_conf**

Default: MT90503_INT_TIMEOUT

error_txb_cell_loss_conf see **fatal_general_conf**

Default: MT90503_INT_TIMEOUT

error_txc_cell_loss_conf see **fatal_general_conf**

Default: MT90503_INT_TIMEOUT

error_cas_change_fifo_conf see **fatal_general_conf**

Default: MT90503_INT_TIMEOUT

error_rx_data_cell_fifo_conf see **fatal_general_conf**

Default: MT90503_INT_TIMEOUT

error_rx_vc_event_fifo_conf see **fatal_general_conf**

Default: MT90503_INT_TIMEOUT

error_clk_recov_a_adap_fifo_conf see **fatal_general_conf**

This parameter should be set to MT90503_INT_DISABLED if clock recovery A FIFO is configured to contain SRTS clock recovery points.

Default: MT90503_INT_TIMEOUT

error_clk_recov_a_remote_fifo_conf see **fatal_general_conf**

This parameter should be set to MT90503_INT_DISABLED if clock recovery A FIFO is configured to contain adaptive clock recovery points.

Default: MT90503_INT_TIMEOUT

error_clk_recov_a_local_fifo_conf see **fatal_general_conf**

This parameter should be set to MT90503_INT_DISABLED if clock recovery A FIFO is configured to contain adaptive clock recovery points.

Default: MT90503_INT_TIMEOUT

error_clk_recov_b_adap_fifo_conf see **fatal_general_conf**

This parameter should be set to MT90503_INT_DISABLED if clock recovery B FIFO is configured to contain SRTS clock recovery points.

Default: MT90503_INT_TIMEOUT

error_clk_recov_b_remote_fifo_conf see **fatal_general_conf**

This parameter should be set to MT90503_INT_DISABLED if clock recovery B FIFO is configured to contain adaptive clock recovery points.

Default: MT90503_INT_TIMEOUT

error_clk_recov_b_local_fifo_conf see **fatal_general_conf**

This parameter should be set to MT90503_INT_DISABLED if clock recovery B FIFO is configured to contain adaptive clock recovery points.

Default: MT90503_INT_TIMEOUT

h100_error_clk_a_conf see **fatal_general_conf**

Default: MT90503_INT_TIMEOUT

h100_error_frame_a_conf see **fatal_general_conf**

Default: MT90503_INT_TIMEOUT

h100_error_clk_b_conf see **fatal_general_conf**

Default: MT90503_INT_TIMEOUT

h100_error_frame_b_conf see **fatal_general_conf**

Default: MT90503_INT_TIMEOUT

alarm_cas_change_fifo_conf see **fatal_general_conf**

Default: MT90503_INT_TIMEOUT

alarm_data_cell_fifo_conf see **fatal_general_conf**

Default: MT90503_INT_TIMEOUT

alarm_rx_vc_event_fifo_conf see **fatal_general_conf**

Default: MT90503_INT_TIMEOUT

api_sync_conf MT90503_INT_DISABLE
MT90503_INT_NO_TIMEOUT

This configuration bit is provided for debug purposes. This interrupt is used by the API to maintain synchronization with the device.

Default: MT90503_INT_NO TIMEOUT

fatal_general_timeout 10 – 10000 ms

This parameter specifies the timeout period of the interrupt associated fatal_general_conf parameter. This parameter should be a multiple of the interrupt_period_granularity parameter. If not, it is rounded up to the next nearest multiple of interrupt_period_granularity before being applied.

Direction: IN Type: ULONG

Default: 10

fatal_cmem_parity_timeout see **fatal_general_timeout**

Default: 10

data_err_dmem_parity_timeout see **fatal_general_timeout**

Default: 10

data_err_utopia_parity_a_timeout see **fatal_general_timeout**

Default: 10

data_err_utopia_parity_b_timeout see **fatal_general_timeout**

Default: 10

data_err_utopia_parity_c_timeout see **fatal_general_timeout**

Default: 10

data_err_scheduler_bw_timeout see **fatal_general_timeout**

Default: 10

error_phy_alarm_a_timeout see **fatal_general_timeout**

Default: 10

error_phy_alarm_b_timeout see **fatal_general_timeout**

Default: 10

error_rxsar_cell_loss_timeout see **fatal_general_timeout**

Default: 10

error_txa_cell_loss_timeout see **fatal_general_timeout**

Default: 10

error_txb_cell_loss_timeout see **fatal_general_timeout**

Default: 10

error_txc_cell_loss_timeout see **fatal_general_timeout**

Default: 10

error_cas_change_fifo_timeout see **fatal_general_timeout**

Default: 10

error_rx_data_cell_fifo_timeout see **fatal_general_timeout**

Default: 10

error_rx_vc_event_fifo_timeout see **fatal_general_timeout**

Default: 10

error_clk_recov_a_adap_fifo_timeout see **fatal_general_timeout**

This parameter is irrelevant if clock recovery A FIFO is configured to contain SRTS clock recovery points.

Default: 10

error_clk_recov_a_remote_fifo_timeout see **fatal_general_timeout**

This parameter is irrelevant if clock recovery A FIFO is configured to contain adaptive clock recovery points.

Default: 10

error_clk_recov_a_local_fifo_timeout see **fatal_general_timeout**

This parameter is irrelevant if clock recovery A FIFO is configured to contain adaptive clock recovery points.

Default: 10

error_clk_recov_b_adap_fifo_timeout see **fatal_general_timeout**

This parameter is irrelevant if clock recovery B FIFO is configured to contain SRTS clock recovery points.

Default: 10

error_clk_recov_b_remote_fifo_timeout see **fatal_general_timeout**

This parameter is irrelevant if clock recovery B FIFO is configured to contain adaptive clock recovery points.

Default: 10

error_clk_recov_b_local_fifo_timeout see **fatal_general_timeout**

This parameter is irrelevant if clock recovery B FIFO is configured to contain adaptive clock recovery points.

Default: 10

h100_error_clk_a_timeout see **fatal_general_timeout**

Default: 10

h100_error_frame_a_timeout see **fatal_general_timeout**

Default: 10

h100_error_clk_b_timeout see **fatal_general_timeout**

Default: 10

h100_error_frame_b_timeout see **fatal_general_timeout**

Default: 10

alarm_cas_change_fifo_timeout	see **fatal_general_timeout**
Default:	10
alarm_data_cell_fifo_timeout	see **fatal_general_timeout**
Default:	10
alarm_rx_vc_event_fifo_timeout	see **fatal_general_timeout**
Default:	10

For more information about all Zarlink products
visit our Web Site at

www.zarlink.com

TECHNICAL DOCUMENTATION - NOT FOR RESALE