MT90528
API Software Design Spec. and Porting Guide

Part Number:	MT90528
Revision Number:	**1.1**
Issue Date:	**September 2003**

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

1.0 General . . . 5
1.1 Software Revision . . . 5
1.2 Environment . . . 5
1.2.1 Language . . . 5
1.2.2 Compiler . . . 5
2.0 System Overview . . . 5
3.0 MT90528 API Overview . . . 8
3.1 General . . . 8
3.2 Service Provided . . . 8
3.2.1 Available MT90528 Registers . . . 8
3.2.2 Available MT90528 Bit Fields . . . 8
3.2.3 User Macros . . . 9
4.0 Porting Software . . . 10
4.1 Porting Overview . . . 10
4.2 Getting Started . . . 11
4.2.1 Installation . . . 11
4.2.2 Study SARs and the ATM model . . . 11
4.2.3 Study MT90528 Software . . . 11
4.2.4 Design . . . 11
4.2.5 Coding . . . 11
4.2.6 Makefiles . . . 11
4.3 Requirements . . . 12
4.3.1 Development System . . . 12
4.3.2 Directory Structure . . . 12
4.3.3 Files to Port . . . 13
4.4 Compile Time Configurations . . . 14
4.5 Application Memory Requirements for API Arrays . . . 15
4.6 External Interfaces . . . 18
4.6.1 MIB Interface . . . 18
4.6.2 Alarm Processing . . . 18
4.6.3 Performance Processing . . . 19
4.6.4 Operating System Interface . . . 19
4.6.5 Hardware I/O Interface . . . 19
4.6.6 Initialization and Shutdown Interfaces . . . 20
4.6.7 ATM Interface . . . 20
4.7 Porting Process . . . 20
4.8 Porting Files . . . 20
4.8.1 drv90528.c . . . 21
4.8.2 drv90528.h . . . 21
4.8.3 drv90528.ini . . . 21
4.8.4 drv90528.rc . . . 21
4.8.5 zarlinkv.h . . . 21
4.8.6 drvwin95.c . . . 21
4.8.7 drvwin95.h . . . 22
4.8.8 low90528.c . . . 22
4.8.9 ioaccess.h . . . 22
4.8.10 platform.h . . . 22
4.8.11 brd90528.c . . . 22
4.8.12 brd90528.h . . . 23
4.8.13 isthread.c . . . 23
4.8.14 userapp.c . . . 23
5.0 Software Design Specification (Zarlink Evaluation Board) . . . 23

Table of Contents

5.1 General . . . 23
5.2 Environment . . . 23
5.2.1 Operating System . . . 23
5.2.2 Language . . . 23
5.2.3 Compiler . . . 23
5.2.4 Co-existence . . . 24
5.3 Hardware . . . 24
5.3.1 General . . . 24
5.3.2 Service Provided . . . 24
5.4 Windows NT Device Drivers . . . 24
5.4.1 General . . . 24
5.4.2 Service Provided . . . 24
5.4.3 Details . . . 24
5.4.4 Portability . . . 25
5.5 Interrupt Service Routine . . . 25
5.5.1 General . . . 25
5.5.2 Design Overview . . . 26
5.5.2.1 Interrupt Service Routine (Windows NT kernel device driver) . . . 26
5.5.2.2 User Application Interrupt Service Thread (Win32) . . . 26
5.6 Board Low Level I/O Interface . . . 27
5.6.1 General . . . 27
5.6.2 Service Provided . . . 27
5.6.3 Details . . . 27
5.6.4 Portability . . . 28
5.7 Evaluation Board Access . . . 28
5.7.1 Evaluation Board Registers . . . 28
5.8 Application Programs . . . 29
5.8.1 General . . . 29
5.8.2 Details . . . 29
6.0 Glossary . . . 30

1.0 General

It is assumed there is a level of software porting necessary to use the MT90528 family of devices in a customer's configuration. This guide highlights the changes required to port to a customer's platform. The guide explains the method of porting in general terms so as not to be platform dependent.

The necessary infrastructure is available to allow state machines to be added by the customer for complete Primary Rate Circuit Emulation AAL1 SAR specification conformance.

Both source code and executable (object) code are supplied to the customer on CD-ROM or diskette for porting purposes.

While best efforts have been made to make the code as portable as possible, programmer's notes are included as comments or pseudo code to indicate possible design issues or concerns.

1.1 Software Revision

The MT90528 software API referred to in this document is at release 3.0.

1.2 Environment

1.2.1 Language

All code is written in 'C', conforms to the ANSI C Specification (Oct. 1988) and follows the ANSI-C Style guide (Dec. 1990).

1.2.2 Compiler

Any ANSI "C" compatible compiler can be used. The API was developed using Microsoft Visual C++ version 6.0. Compiler-specific code has been identified. The source code for the API was also compiled and linked using the GNU compiler GCC with compiler definitions "TARGET_OS" for PLATFORM in platform.h and "TARGET_DRIVER" for DRIVER in brd90528.h.

NOTE: Setting the correct compile time definitions will allow any ANSI "C" compatible compiler to be used.

2.0 System Overview

Customers will most likely be porting the software to another platform (e.g., processor, bus etc.) and operating system (embedded, RTOS etc.). As such, the design attempts to make the code as operating system and platform independent as possible.

The following diagram (Figure 1 on page 6) represents a high level view of the software. Each of the following sections describes a component and the services that it provides the other components.

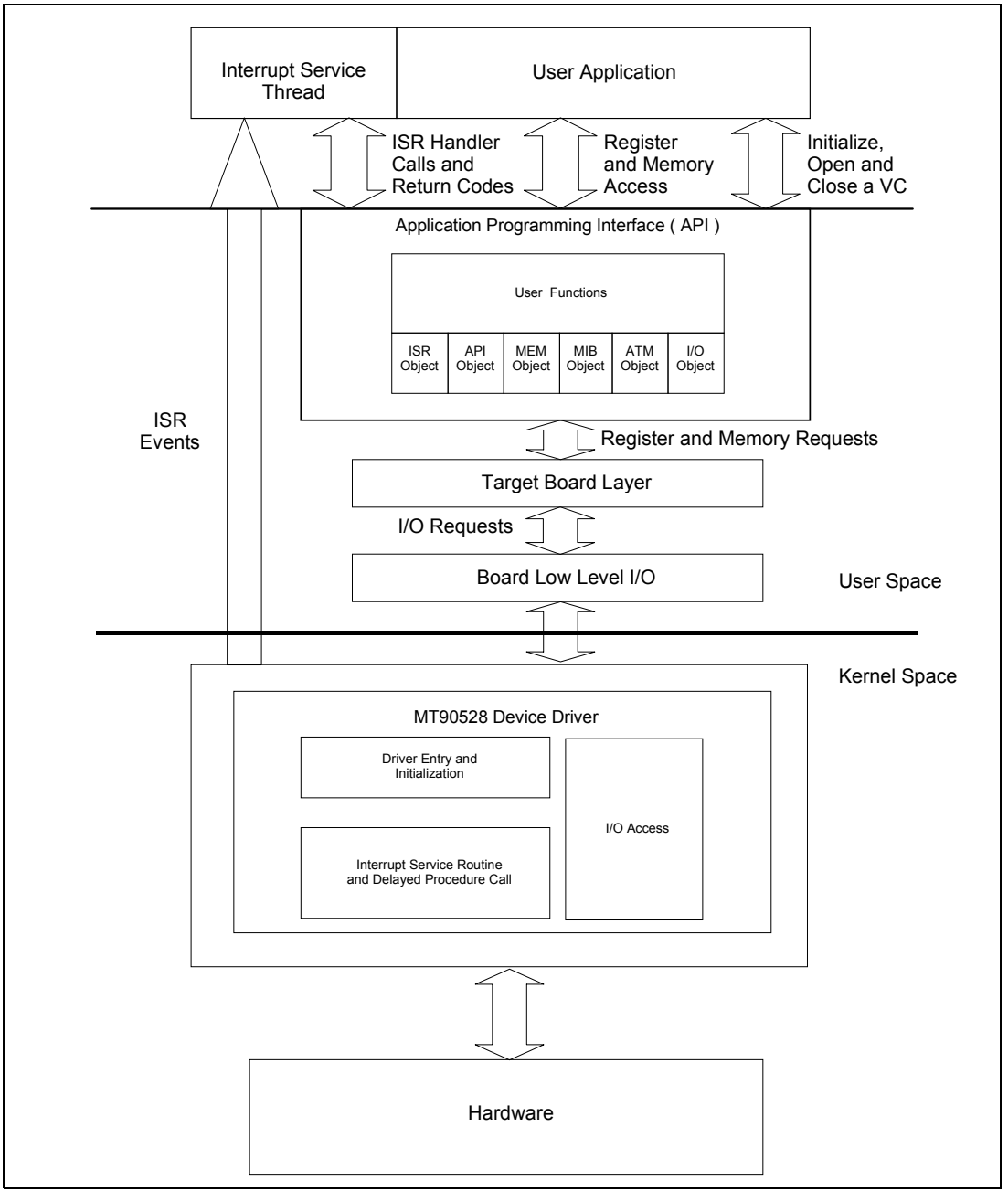

Figure 1 - System Overview

Module	Description
User Application	The user application code has to be developed by the user of the API. Example code has been provided on how to initialize the MT90528 device, how to open a VC or multiple VCs, how to close a VC, how to spawn an interrupt service thread and how to shut down the device.
Interrupt Service Thread	The interrupt service thread code has to be developed by the user of the API. Example code has been provided on how to create plus wait for an interrupt event and how to service the interrupt events. Some of this code may need to be moved down to the device driver interrupt service routine due to performance issues for real time operating systems.
User Functions	These are the API calls that the user application is using. This module is the external interface of the API.
API Object	This module in the API initializes the MT90528 device to allow writing to and reading from registers and memory. Specific interrupts are enabled. The module also shuts down the MT90528 device (closes the device driver).
ATM Object	This module in the API initializes the device and TDM ports. The module also opens and closes CBR and Data VCs.
MIB Object	This module deals with the statistics structures and MIB counters and flags.
MEM Object	This module is internal to the API and is used to manage internal and external memory.
ISR Object	This module in the API has routines that handle the MT90528 device interrupts for each of the functional modules on the device (e.g. TXSAR, SDT RXSAR)
I/O Object	This module in the API handles the register and memory input/output accesses on the MT90528 device.
Target Board Layer	This layer handles the input and output accesses for the MT90528 device and the board on which the MT90528 device is used. All of the board-specific definitions reside in this layer.
Board Low Level I/O	This layer handles the board address-specific input and output accesses to and from any customer device driver. It also has utility routines for opening and closing a device driver, creating and closing interrupt events on the device driver and acquiring a device driver handle/descriptor.
Device Driver Entry and Initialization	The device driver is initialized to reset the board and MT90528 device. The board and MT90528 device interrupts are disabled. The device driver interrupt service routine is enabled.
Device Driver I/O Access	The device driver input/output access module provides reading from and writing to the hardware on the board and MT90528 device either using port I/O or memory I/O ISA access.

Table 1 - System Overview Module Description

Module	Description
Device Driver Interrupt Service Routine	The device driver interrupt service routine services the board and MT90528 device hardware interrupts, checking for valid MT90528 device interrupts and initiating an interrupt event to the user's interrupt service thread.
Hardware	Hardware accesses to the board and MT90528 device.

Table 1 - System Overview Module Description

3.0 MT90528 API Overview

3.1 General

The Zarlink Semiconductor API ("MT90528") layer is provided in source code format. The API provides a high level interface for the programming of the MT90528 device. It is assumed that most application programs could be built using this API.

3.2 Service Provided

The API provides a user interface to the MT90528 device. This API could be ported to a RTOS with minimal effort.

The MT90528 API must serve several distinct purposes:

1. Allow the user to access multiple devices on a single card, and arbitrate resource conflicts
2. Allow the user access to information (**all** registers and memory) on an MT90528 device
3. Abstract the common tasks necessary to initialize, configure and utilize a given device (thus allowing access to the device at a higher level than pure register access).
4. Allow registration, binding and reporting of asynchronous events to the user application

In short, the API must convert register or memory locations and data to I/O requests. It must then pass the requests to the lower layer interface.

See the Zarlink Semiconductor C313BCS3 MT90528 API Specification document for a list of API function calls.

3.2.1 Available MT90528 Registers

See C313BCS3 MT90528 API Specification concerning the MT90528 Register names.

3.2.2 Available MT90528 Bit Fields

All register bit fields are available through the `Mt90528ReadRegister` and `Mt90528WriteRegister` API calls.

Each bit field in a register can be manipulated using `InsertBitPattern` before calling `Mt90528WriteRegister` or `ReadBitPattern` after calling `Mt90528ReadRegister.`

There are definitions available for "Position of Bits" and "Number of bits to shift the value left".

For example:

Position of bits: `MSR_RX_TIMEOUT_SRV`

Number of bits to shift the value left: `BS_MSR_RX_TIMEOUT_SRV`

For brevity, the values have not been included in this document. They are included in the dev90528 header file in the source code distribution.

3.2.3 User Macros

User macros that encapsulate common register and bitfield manipulation are provided in the MT90528 API library.

```
InsertBitPattern ( ... )
ReadBitPattern ( ... )
```

4.0 Porting Software

4.1 Porting Overview

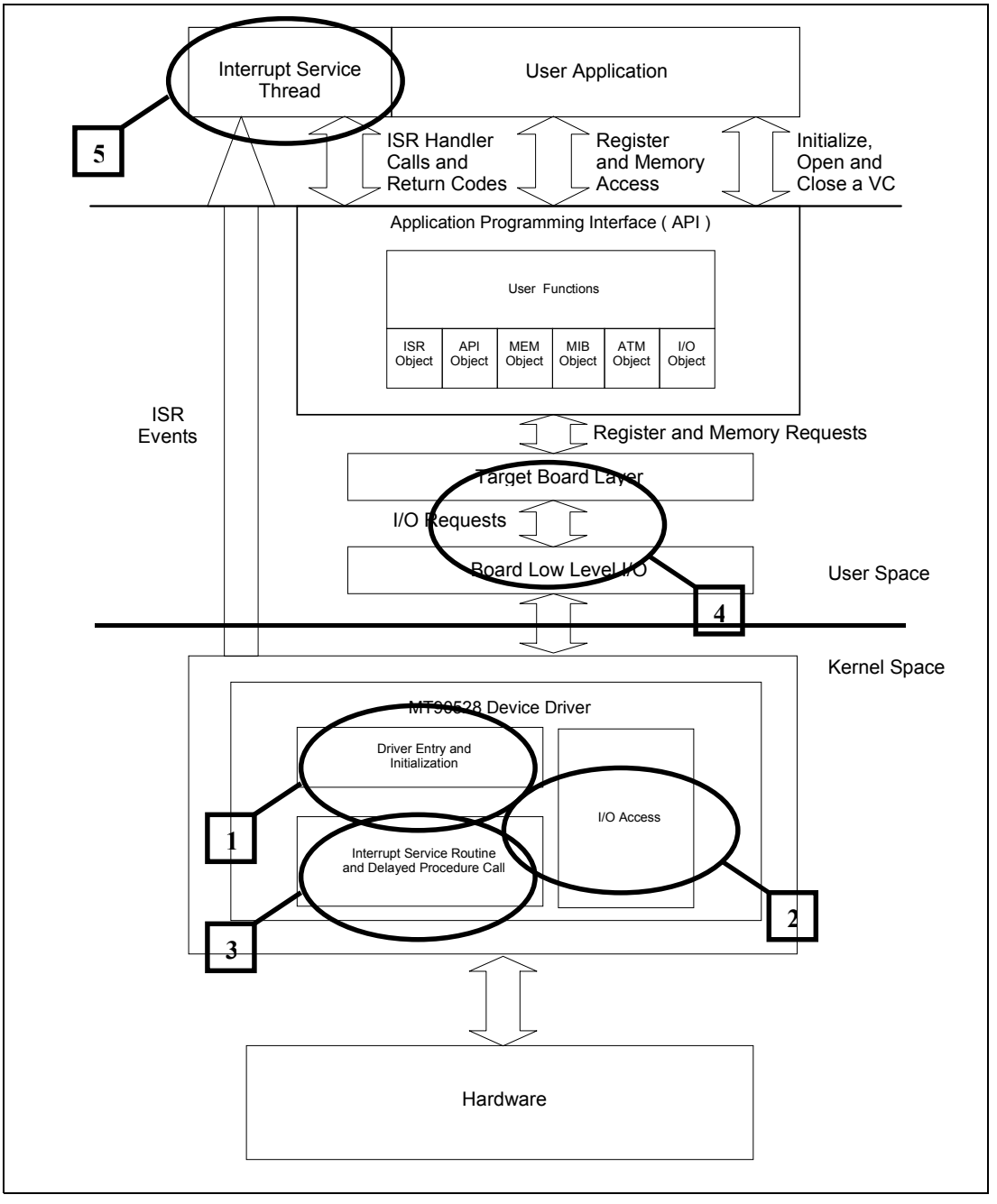

Figure 2 - Porting Steps

A number of functional layers need to be modified to port to embedded systems or real time O/Ss. The layers are device driver, low level I/O interface and the user application layers.

In “Figure 2, “Porting Steps” on page 10” the modules requiring modification are listed in the order of modification.

1. Driver entry and initialization (driver and MT90528 device)
2. Driver input / output access to the hardware, and interface to the low level layer
3. Driver interrupt service routine
4. Board-specific low level I/O access to the device driver
5. User application interrupt service thread

NOTE: Customers may already have a device driver developed for their target operating system and therefore steps 1, 2, 3 and 5 may not be required.

The API calls are documented in the C313BCS3 MT90528 API Specification and therefore are not documented here as part of the porting requirements/guide.

4.2 Getting Started

This section describes the process of porting and integrating the MT90528 software to your target system. The process involves the following steps:

4.2.1 Installation

Install the software source-tree into the target development system. The software is packaged as a ZIP file. Winzip, Pkunzip, or Unzip can be used to unzip the source-tree. The source tree is a project in Microsoft Visual C++ but can be pulled into other IDEs (Integrated Development Environments).

4.2.2 Study SARs and the ATM model

Study the SAR technology/concepts. You should be familiar with the ATM Forum’s ATM model and how a Segmentation and Reassembly device fits into that model. You should be familiar with UDT and SDT modes as well as SNMP (simple network management protocol) which MIB is part of. An understanding of E1 and T1 signaling/framing is required.

4.2.3 Study MT90528 Software

Study Zarlink Semiconductor’s MT90528 software layers (user, API, Brd, lowlevel, driver) and API external interfaces.

4.2.4 Design

Design the overall architecture to integrate the MT90528 API software with the target system. Decide if the MT90528 API software should be created as a single task or multiple tasks.

4.2.5 Coding

Design and develop the required user code and modify the required port files for the target system.

4.2.6 Makefiles

Modify the appropriate makefiles to compile the MT90528 source files and link the objects. Sample makefiles are provided for reference.

4.3 Requirements

4.3.1 Development System

The MT90528 source code does not put any particular requirements on the target development system. The target development system should have enough free disk space. The source code has been developed in the ANSI-standard C language. The source tree contains the set of header (.h) and C source (.c) files.

4.3.2 Directory Structure

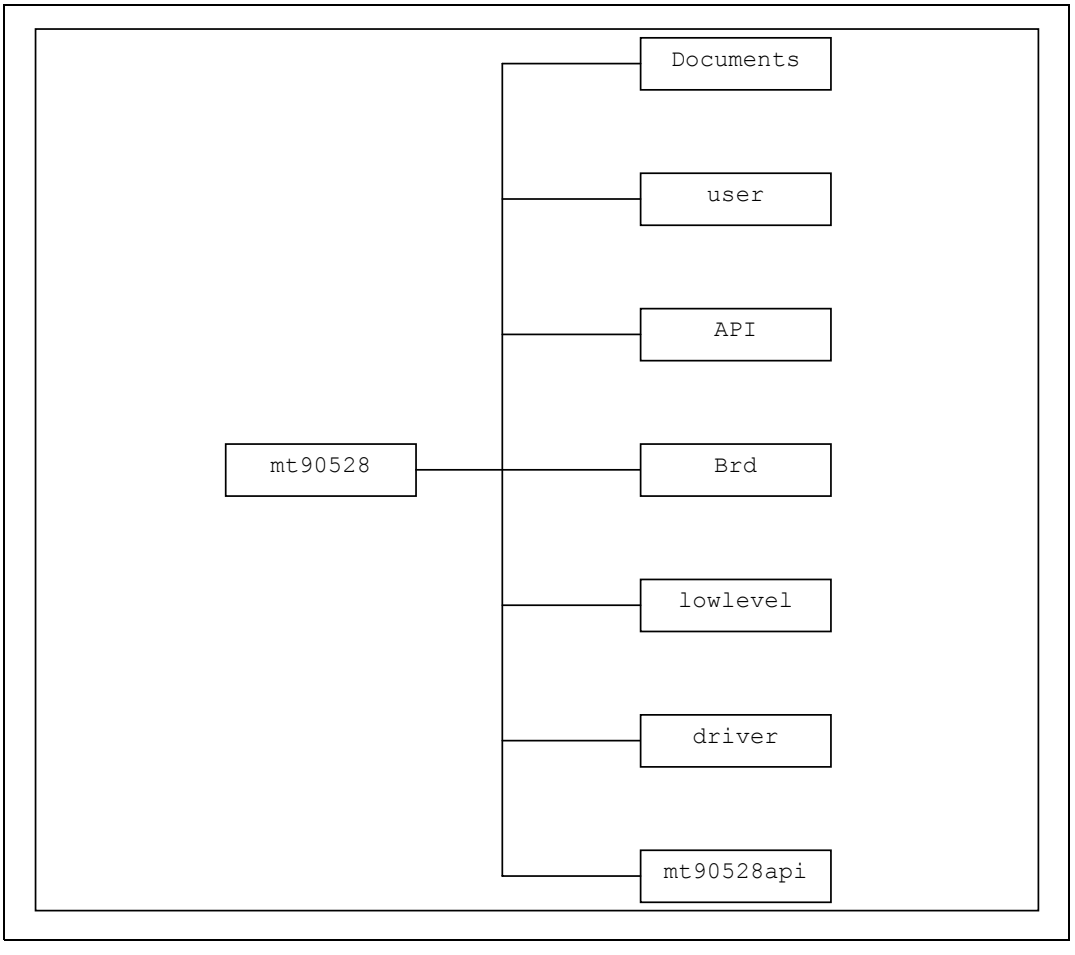

Figure 3 - Directory Structure

The MT90528 distribution floppy or CD or downloaded zip file extracts the software into the *mt90528* directory. The *mt90528* directory contains the following sub-directories:

- *Documents*: This sub-directory contains all the user documents (e.g. MT90528 API document, Porting Guide)
- *user*: This sub-directory contains example user application and test programs. It contains the source files as “C” programs and headers. See README.txt in this directory concerning the userapp.c program.
- *API*: This sub-directory contains the API programs and interfaces (API, ATM, I/O, ISR, MIB and MEM). It contains the source files as “C” programs and headers.

- *Brd*:This sub-directory contains the device and board-level interfaces and definitions. It contains the source files as “C” programs and headers.
- *lowlevel*:This sub-directory contains the board I/O interfaces to the device driver. It contains the source files as “C” programs and headers.
- *driver*:This sub-directory contains the device driver programs and routines. It contains the source files as “C” programs and headers. The driver files, which make up a Windows NT driver, must be compiled using the Windows NT DDK and nmake. See README.txt in this directory for instructions on how to compile and build the drv90528.sys device driver and how to install it. For other operating systems this driver will have to be replaced or reworked.
- *mt90528api*:This sub-directory contains the project and files for Microsoft Visual C++ and a release and debug directory of the mt90528 executable. The makefiles are in this directory.

4.3.3 Files to Port

Certain files need to be modified to port to a different target system. Those files that are not part of the API layer are described here in Table 2.

Location/File	Description
user/**userapp.c**	This is the mainline program that makes calls to the MT90528 API. This is an example program that probably won’t be used in an embedded or real-time O/S. This program is useful as an example for initializing the MT90528 device and initializing the interrupt service thread.
user/**isthread.c**	This is the interrupt service thread that contains the initialize service threads routine, the service thread routines and the ISR handler routine. This is an example thread that probably won’t be used in an embedded or real-time O/S. This program may require modifications to the initialize service thread and service thread routines if platforms other than Windows NT are used.
Brd/**brd90528.c**	This file contains the read and write (register, internal and external memory) routines and some utility routines. The address calculation may have to change depending on how the AEM pin is implemented on the target board for the MT90528 device. Code may have to be written for the routines `mt90528_current_milliseconds` and `GoToSleep` if an OS other than Windows95/NT is used.
Brd/**brd90528.h**	This file contains all of the board-specific definitions and addresses. This file has to be modified for the target board to change the board addresses and the device start addresses. It also contains important compile time flags such as `DRIVER` and `EMULATE_DRIVER`.
Brd/**platform.h**	This file contains platform-specific definitions and utility macro definitions. It also contains an important compile time flag, `PLATFORM`.
lowlevel/**low90528.c**	This file contains the routines for reading from and writing to the device driver as well as opening and closing the device driver. The I/O access lock and device interrupt event creation routines plus the associated closing routines are provided as well. The lowest layer routines need to be modified to port to other target O/Ss.

Table 2 - Files to be modified by user

Location/File	Description
lowlevel/**ioaccess.h**	This file contains the device driver-specific definitions and the compile time flag PORT_IO_CT_FLAG for selecting memory I/O or port I/O access. Target-specific device driver definitions should go in this file.
driver/**drv90528.h**	This file contains WindowsNT-specific device driver definitions. The definitions are discarded if porting to other target O/Ss.
driver/**drv90528.c**	This file contains the WindowsNT-specific code for the device driver. The driver code has to be modified or more likely replaced for other target O/Ss.

Table 2 - Files to be modified by user

4.4 Compile Time Configurations

Compiler Option	Defined In	Description
PLATFORM	platform.h	Three possible operating systems can be selected (WINDOWSNT, WINDOWS95 and TARGET_OS) NOTE: the TARGET_OS option requires code to be written in all places where the PLATFORM constant is checked in the software
DRIVER	brd90528.h	Two possible device drivers can be selected (ZARLINK_EVAL and TARGET_DRIVER) NOTE: the TARGET_DRIVER option requires code to be written in all places where the DRIVER constant is checked in the software, if a device driver is required
EMULATE_DRIVER	brd90528.h	If defined, the device driver is emulated in the PC's memory in the low90528 module for I/O reads and writes.
MEMORY_ACCESS	brd90528.h	Two possible memory accesses can be selected (MEM_ACC_DIRECT and MEM_ACC_INDIRECT) for direct or indirect access to the MT90528 device internal and external memory.
MEM_WRITE_VERIFY	brd90528.h	If defined, enables verification of the data during writes to the device. If undefined, provides a performance improvement but writes to the device will not be verified. The constant is undefined by default.
BOARD_MACR_MCFG	brd90528.h	Five possible external memory configurations can be selected (MACR_NO_EXTERNAL_MEMORY, MACR_1MEG_BANK, MACR_512K_BANKS, MACR_256K_BANKS and MACR_128K_BANKS) depending on what external memory configuration is required on the board.

Table 3 - Compile Time Options

Compiler Option	Defined In	Description
BOARD_MACR_MTYP	brd90528.h	Two possible external memory types can be selected (MACR_FLOW_THROUGH, MACR_PIPELINED)
EXT_MEMORY_PARITY_TYPE	brd90528.h	Two possible external memory parity types can be selected (MEM_NON_PARITY, MEM_PARITY)
BOARD_TDM1_TDM_LOS_POL	brd90528.h	Two possible TDM LOS polarity can be selected for UDT mode only (TDM1_NEGATIVE_POLARITY, TDM1_POSITIVE_POLARITY)
MAX_PORTS [1]	brd90528.h	Up to 28 TDM ports
MAX_VC[2]	brd90528.h	Up to 896 CBR VCs
MAX_DATA_VC	brd90528.h	Up to 65536 Data VCs
MAX_MT90528_DEVICES	brd90528.h	Maximum MT90528 devices on target board
BOARD_BASE_ADDRESS	brd90528.h	Base address of target board
MT90528_BASE_ADDRESS	brd90528.h	Base address of MT90528 devices
EXT_MEMORY_BASE_ADDRESS	brd90528.h	Base address of MT90528 external memory
MAX_ADDRESS_BRD_MEMORY	brd90528.h	Maximum address of target board memory
MAX_ADDRESS_EXT_MEMORY	brd90528.h	Maximum address of MT90528 external memory
PORT_IO_CT_FLAG	ioaccess.h	If defined, the PC ISA bus I/O access is port I/O access else if not defined, memory I/O access

Table 3 - Compile Time Options

1. MAX_PORTS must be changed to 8 if using the MT90520 8 port device
2. MAX_VC must be changed to 256 or less if using the MT90520 8 port device

4.5 Application Memory Requirements for API Arrays

The following is a table of memory usage by the MIB statistics, APP statistics and memory manager arrays based on MAX_PORTS and MAX_VC constant values. This table gives an idea of how much application memory should be allocated for these arrays. The memory usage doesn't include the executable code memory usage or the parameter structures memory usage as these are variable based on the customer's application. **NOTE**: these memory requirements are per device.

MAX_VC	MAX_PORTS	MIB Statistics	APP Statistics	Memory Manager	Total (in bytes)
16	8	4343	1280	2312	7935
32	8	7543	2560	3464	13567
48	8	10743	3840	4616	19199
64	8	13943	5120	5768	24831

Table 4 - Application Memory Requirements for API Arrays

MAX_VC	MAX_PORTS	MIB Statistics	APP Statistics	Memory Manager	Total (in bytes)
80	8	17143	6400	6920	30463
96	8	20343	7680	8072	36095
112	8	23543	8960	9224	41727
128	8	26743	10240	10376	47359
144	8	29943	11520	11528	52991
160	8	33143	12800	12680	58623
176	8	36343	14080	13832	64255
192	8	39543	15360	14984	69887
208	8	42743	16640	16136	75519
224	8	45943	17920	17288	81151
240	8	49143	19200	18440	86783
256	8	52343	20480	19592	92415
16	28	6923	1280	3272	11475
32	28	10123	2560	4424	17107
48	28	13323	3840	5576	22739
64	28	16523	5120	6728	28371
80	28	19723	6400	7880	34003
96	28	22923	7680	9032	39635
112	28	26123	8960	10184	45267
128	28	29323	10240	11336	50899
144	28	32523	11520	12488	56531
160	28	35723	12800	13640	62163
176	28	38923	14080	14792	67795
192	28	42123	15360	15944	73427
208	28	45323	16640	17096	79059
224	28	48523	17920	18248	84691
240	28	51723	19200	19400	90323
256	28	54923	20480	20552	95955
272	28	58123	21760	21704	101587
288	28	61323	23040	22856	107219

Table 4 - Application Memory Requirements for API Arrays

MAX_VC	MAX_PORTS	MIB Statistics	APP Statistics	Memory Manager	Total (in bytes)
304	28	64523	24320	24008	112851
320	28	67723	25600	25160	118483
336	28	70923	26880	26312	124115
352	28	74123	28160	27464	129747
368	28	77323	29440	28616	135379
384	28	80523	30720	29768	141011
400	28	83723	32000	30920	146643
416	28	86923	33280	32072	152275
432	28	90123	34560	33224	157907
448	28	93323	35840	34376	163539
464	28	96523	37120	35528	169171
480	28	99723	38400	36680	174803
496	28	102923	39680	37832	180435
512	28	106123	40960	38984	186067
528	28	109323	42240	40136	191699
544	28	112523	43520	41288	197331
560	28	115723	44800	42440	202963
576	28	118923	46080	43592	208595
592	28	122123	47360	44744	214227
608	28	125323	48640	45896	219859
624	28	128523	49920	47048	225491
640	28	131723	51200	48200	231123
656	28	134923	52480	49352	236755
672	28	138123	53760	50504	242387
688	28	141323	55040	51656	248019
704	28	144523	56320	52808	253651
720	28	147723	57600	53960	259283
736	28	150923	58880	55112	264915
752	28	154123	60160	56264	270547
768	28	157323	61440	57416	276179

Table 4 - Application Memory Requirements for API Arrays

MAX_VC	MAX_PORTS	MIB Statistics	APP Statistics	Memory Manager	Total (in bytes)
784	28	160523	62720	58568	281811
800	28	163723	64000	59720	287443
816	28	166923	65280	60872	293075
832	28	170123	66560	62024	298707
848	28	173323	67840	63176	304339
864	28	176523	69120	64328	309971
880	28	179723	70400	65480	315603
896	28	182923	71680	66632	321235

Table 4 - Application Memory Requirements for API Arrays

4.6 External Interfaces

The MT90528 API is a modular architecture with well-defined external (user) interfaces in each of the modules. A certain amount of data coupling exists for user-supplied data or passing back info data in data structures. These structures are documented in the C313BCS3 MT90528 API Specification for each of the appropriate interfaces (function calls). The MIB implementation provides statistical information through the MIB interfaces outlined in the API specification document.

4.6.1 MIB Interface

This interface is through the Interrupt Service Handlers. The handlers are:

Interface	Description
Mt90528IsrHandler	handles all device interrupts and updates MIB statistics
Mt90528IsrHandlerWith2ndThresholds	handles all device interrupts and updates MIB statistics plus tracks a second set of thresholds.

Table 5 - MIB interfaces

Typically these interfaces would be called from a service thread or process when an interrupt has occurred on the MT90528 device. The Main Status Register on the device should be queried to determine which module triggered the interrupt and the interrupt should be serviced using the appropriate handler. Information about the interrupt (counters, overflows etc…) is passed back to the user in the statistics. All handlers provide a return code if errors occurred during the handling of the interrupt. If an error occurred, processing is terminated and returned back to the user. Therefore the sub-interrupt on the device may not have been cleared and the user needs to handle the error and clear the sub-interrupt. See C313ACS3 API Specification document for a detailed description of the MIB statistics structure and MIB functions to retrieve the statistics.

4.6.2 Alarm Processing

Available using the MIB Interface (Interrupt Handlers).

4.6.3 Performance Processing

The interrupt interfaces require the most care in integrating into your target code. This is because you may have to place code directly in the interrupt routines or high priority tasks and pass information as efficiently as possible (real time impact).

Performance code is any code that requires real time actions and responses such as interrupt handling to correct data errors and buffer overflows or underflows.

4.6.4 Operating System Interface

These interfaces are at the lowest layer for input/output access to the device driver, mutex (mutual exclusion) locks and interrupt events. Most of the porting occurs at this interface. Care must be taken to ensure that real time performance is not affected by your design. The target operating system event handlers should be used for notification of interrupt events.

Interface	Description
`IoMemAccess`	Provides memory based I/O accesses to the device driver (NT OS specific).
`IoPortAccess`	Provides port based I/O accesses to the device driver (NT OS specific).
`OpenDeviceDriver`	Opens the device driver for access and provides a handle.
`CloseBoardDriver`	Closes the device driver using the handle provided by `OpenDeviceDriver`.
`CreateIoLock`	Creates an I/O mutex lock and provides a handle (a constructor).
`CloseIoLock`	Closes the mutex lock handler using the handle provided by `CreateIoLock` (a destructor).
`AcquireIoLock`	Wait for and acquire an I/O mutex lock using the handle provided by `CreateIoLock`.
`ReleaseIoLock`	Releases an I/O mutex lock using the handle provided by `CreateIoLock`.
`CreateBoardIrqEvent`	Creates an event handler for interrupts and provides a handle.
`CloseBoardIrqEvent`	Closes the event handler using the handle provided by `CreateBoardIrqEvent`.
`SavePortDeviceAddress`	Saves the device address in the device driver extension

Table 6 - Operating System Interfaces

4.6.5 Hardware I/O Interface

This interface is between the device driver and the hardware layer of the target operating system. As mentioned before, the device driver that includes the interrupt service handler has to be redesigned if WindowsNT O/S is not used.

4.6.6 Initialization and Shutdown Interfaces

These interfaces initialize or shut down the board and device. Porting should not be required.

Interface	Description
`Mt90528InitializeApi`	Initializes the device, external memory memory manager and statistics structure
`Mt90528ShutDownApi`	Cleans up the memory manager and statistics structure

Table 7 - Initialization and Shutdown Interfaces

4.6.7 ATM Interface

This interface provides the ability to set up the TDM and UTOPIA ports plus open and close CBR and Data VCs. No porting is required. The setting up of the ports requires data structures to be filled and passed in by the user.

See C313BCS3 MT90528 API Specification concerning the API function calls.

4.7 Porting Process

The following steps should be followed to port the MT90528 software to an embedded processor or different real time operating system (RTOS):

- Port specific initialization code from the Windows NT kernel mode device driver to RTOS environment (if required)
- Port specific hardware I/O access routines from the Windows NT kernel mode device driver to RTOS environment (if required)
- Port specific interrupt service routine code from the Windows NT kernel mode device driver to RTOS environment (if required) (replace WinNT user events with O/S-specific event handler)
- Change low layer interface to use O/S-specific I/O calls
- Replace interface to Windows NT Registry with appropriate calls to another database application (if required)
- Port/Replace all compiler-specific and processor-specific code in layers outside of the API
- Port the interrupt and timer service thread code at the user application layer (if required)
- Convert, if required, the device driver and user interrupt service thread to a state machine
- Recompile user API for new environment
- Recompile test application as the main task in the RTOS
- Run reference scripts against new environment to ensure compatibility

NOTE: The customer may have existing drivers or hardware I/O access that may only require porting of the lowlevel layer (brd90528.h, lowlevel.h and lowlevel.c).

4.8 Porting Files

A number of definitions may have to be added or modified in both brd90528.h and platform.h to accommodate the target platform and board.

The MT90528 API was designed in layers to ensure that the core of the API is not affected and that minimal porting of the upper and lower layers is required.

4.8.1 drv90528.c

This file is the Windows NT device driver for the MT90528 device. The code is tailored to the Zarlink Semiconductor MT90528 evaluation board. If another board is used with a Windows NT platform, the only change required is the board reset register location. Do not use this file on other operating systems.

In the driver entry code, the `BOARD_CONTROL_STATUS_REG` location would have to change and `BOARD_ASSERT_SW_RESET` may have to change depending on how the reset is asserted. If the port is for a port I/O or memory I/O design, the appropriate code has to be modified for the type of board I/O access.

If the target platform is not Windows NT, the driver can be discarded and a new device driver including the interrupt service routine (ISR) must be created for the target platform if not already available. drv90528.c is not part of the API (resides in the driver layer).

4.8.2 drv90528.h

This is the associated header file for the Windows NT MT90528 device driver. No changes are required if another board is used with a Windows NT platform.

If the target platform is not Windows NT, the header can be discarded. drv90528.h is not part of the API (resides in the driver layer).

4.8.3 drv90528.ini

This is the Windows NT registry initialization file for assigning an I/O address, memory base and interrupt number. The settings are based on the Zarlink Semiconductor MT90528 evaluation board. The settings may have to change if any of the defaults are different on another board with a Windows NT platform.

If the target platform is not Windows NT, then the registry ini file can be discarded. drv90528.ini is not part of the API (resides in the driver layer).

4.8.4 drv90528.rc

This is the resource file based on Windows NT. No changes should be required.

If the target platform is not Windows NT, then the resource file can be discarded. drv90528.rc is not part of the API (resides in the driver layer).

4.8.5 zarlinkv.h

This is the vendor info file for Windows NT device driver builds. Changes are required if porting.

If the target platform is not Windows NT, the header file can be discarded. zarlinkv.h is not part of the API (resides in the driver layer).

4.8.6 drvwin95.c

This is the Windows 95 device driver source file. It's not a true driver file but a utility for accessing the ISA bus for port I/O only. No interrupt or timer events are available with this module. If another board with port I/O access is used with a Windows 95 platform, no changes are required. The base ISA address defined as `ISA_BASE_ADDRESS` may have to be changed in brd90528.h

If the target platform is not Windows 95, the source file can be discarded. drvwin95.c is not part of the API (resides in the driver layer).

4.8.7 drvwin95.h

This is the associated header file for the Windows 95 device driver. No changes are required if another board is used with a Windows 95 platform.

If the target platform is not Windows 95, the header can be discarded. drvwin95.h is not part of the API (resides in the driver layer).

4.8.8 low90528.c

If the target platform is Windows NT or Windows 95, it is unlikely that any changes are required in this file.

If the target platform is a non-Windows O/S, a number of routines require code added for I/O access to the target device driver, opening and closing the device driver, creating and closing an interrupt event handler and creating and closing the timer event handler. The I/O mutex locks will have to be modified for other target OSs, creating, waiting for, acquiring, releasing and closing an I/O lock.

IoMemAccess or **IoPortAccess** functions require code where indicated to provide I/O access plus create and close an IRQ event handler in the target device driver.

OpenBoardDriver and **CloseBoardDriver** functions require code where indicated to open the target device driver plus acquire a driver handle/descriptor and close the target device driver.

CreateIoLock, **CloseIoLock**, **AcquireIoLock** and **ReleaseIoLock** functions require code where indicated to create, wait for, acquire, release and close an I/O locking mechanism to prevent out of sequence reads and writes between processes/threads.

Another option is to bypass the above mentioned routines if direct hardware access is required. Replace the code in the functions **ReadBoardAddress** and **WriteBoardAddress** to provide direct I/O access.

low90528.c is not part of the API (resides in the lowlevel layer).

4.8.9 ioaccess.h

If the target platform is Windows NT or Windows 95, it is unlikely that any changes are required in this file.

If the target platform is a non-Windows O/S, a new message structure, where indicated in the source file, may have to be created to communicate with the target device driver through the target operating system. If a message structure is not required then this file can be discarded. ioaccess.h is not part of the API (resides in the lowlevel layer).

4.8.10 platform.h

The `PLATFORM` compile time flag may have to be changed depending on the target O/S. platform.h is not part of the API (resides in the Brd layer).

4.8.11 brd90528.c

The read and write routines (registers, internal memory, and external memory) may require modification if the target board is not the Zarlink Semiconductor MT90528 evaluation board. Where indicated in the source code (if NOT `ZARLINK_EVAL`), the AEM (Access External Memory) bit may have to be toggled for internal device access or external memory access depending on how the AEM pin on the device is wired on the board. Typically it should be connected to the high-order CPU address line. Currently the non-Zarlink evaluation board code ignores the AEM bit when calculating the board address. Code may have to be written for mt90528_current_milliseconds to replace the existing code if the ANSI clock() function is not available on the target system. brd90528.c is not part of the API (resides in the Brd layer).

4.8.12 brd90528.h

This header file contains the majority of definitions that require changing depending on how the target board is set up. See “Table 3” concerning the compile time flags. A number of board-based addresses may have to be changed depending on how the target board is memory mapped. brd90528.h is not part of the API (resides in the Brd layer).

4.8.13 isthread.c

This module, as part of the user application, contains the service thread creation, the interrupt service handler routines (that get an interrupt event from the device driver and services, ALL active interrupts on the MT90528 device) plus the timer service handler that asserts and de-asserts alarms. The code in this module was developed for the Windows NT O/S environment and has to be modified or replaced for any other target platforms. It is intended as an example of how interrupts should be serviced on the MT90528 device. Embedded systems and real-time O/Ss may require the interrupt handler code in the device driver. isthread.c is not part of the API (resides in the user layer).

4.8.14 userapp.c

This mainline program is only an example of how the MT90528 API can be used plus how interrupt and timer service threads can be initiated. It is unlikely this mainline would be used in any implementation. userapp.c is not part of the API (resides in the user layer). See C313BCS3 MT90528 API Specification for a detailed description of the API calls.

5.0 Software Design Specification (Zarlink Evaluation Board)

5.1 General

The MT90528 software demonstrates and tests the MT90528 device on the Zarlink MT90528 engineering evaluation board. This section of the document highlights the software design of layers outside the MT90528 software API specific to the Zarlink engineering evaluation board and may or may not be of interest to customers.

5.2 Environment

5.2.1 Operating System

The target operating system is Microsoft Windows NT Version 4.0 Workstation (Build 1381 or newer). It is assumed that the code also works on Windows NT 4.0 Server, but testing was limited to the Workstation release. Older versions of Windows NT (pre-4.0) are not supported. Operating system-dependent code has been identified in this porting guide, however it should be noted that most of the Kernel Mode device driver, especially the interrupt service routine (ISR) is Windows NT-specific.

5.2.2 Language

All code is written in ‘C’, conforms to the ANSI C Specification (Oct. 1988) and follows the ANSI-C Style guide (Dec. 1990).

5.2.3 Compiler

The compiler used was the Microsoft Visual C++ version 6.0. Compiler-specific code has been identified. The Microsoft Win32 Software Development Kit (Win32 SDK), Device Driver Development Kit (NT DDK), SoftICE for Windows NT and related tools were used for development and testing. Nmake was used to compile and link the device driver.

5.2.4 Co-existence

The MT90528 evaluation board coexists with other Zarlink Semiconductor evaluation boards and devices. The MT90528 complements, but does not interfere with existing Zarlink Semiconductor software in both design and function.

5.3 Hardware

5.3.1 General

The target system consists of an MT90528 evaluation board running in an ISA standard chassis. The minimum host processor is a single Intel Pentium™ running at 233 MHz. Slower processors may be used, but it is assumed there may be some performance issues. The MT90528 board has no local processor.

5.3.2 Service Provided

The hardware provides the Windows NT device driver's register and interrupt access. The register access may be either an I/O port access or I/O memory mapped access. A single (standard ISA) interrupt is provided on the evaluation board. The evaluation board uses I/O port access (high order address, low order address and data location).

5.4 Windows NT Device Drivers

5.4.1 General

The Windows NT device driver is a kernel mode driver.

5.4.2 Service Provided

The device driver provides the interface between the Windows NT software and evaluation board hardware and has two main purposes:

- accept and process I/O requests from user space
- service interrupts (see Interrupt Service Routine)

One MT90528 device is supported on each evaluation board and the device driver supports only one evaluation board per system. The MT90528 API does have the capability of supporting multiple devices.

5.4.3 Details

The Windows NT device driver design is as simple as possible - effectively being a dispatcher of port and memory mapped I/O requests. Board IDs, location and other configuration information is obtained from either the NT registry entry, or default (compile time) values.

Each NT device driver supports the following I/O functions:

Function	Description
IoPortAccess()	• access an I/O port
IoMemAccess()	• access a memory mapped region
OpenBoardDriver()	• open the MT90528 board and device

Table 8 - I/O Functions

Function	Description
	• obtain an NT device driver handle
CloseBoardDriver()	• close the MT90528 board and device
	• release the NT device driver handle
CreateBoardIrqEvent()	• create an MT90528 board interrupt event
	• obtain an NT interrupt event handle
CloseBoardIrqEvent()	• close an MT90528 board interrupt event
	• release the NT interrupt event handle
SavePortDeviceAddress()	• save the device address in the device driver extension

Table 8 - I/O Functions

The device driver interrupt service routine sets an event, which the user application interrupt service thread waits for and then acts upon (see isthread.c in the mt90528api user directory as an example).

5.4.4 Portability

The device drivers are WinNT-specific. The I/O access capabilities should be ported at the low-level interface layer, as opposed to the device driver level, as the WinNT device drivers simply process I/O requests.

The device driver interrupt service routine needs to be rewritten when moving to a different operating system. The supplied ISR serves as an example of how interrupts may be serviced. The majority of work to move the device drivers to a different operating system takes place in the interrupt service routine and related code.

5.5 Interrupt Service Routine

5.5.1 General

The interrupt service routine for the MT90528 device exists partly in the "DRV90528" kernel device driver and partly in the user application. Consequently, the kernel portion is Windows NT-specific. The code and related comments, however, serve as an example of how to process MT90528 interrupts. The concept and design is useful to designers using other operating systems, but the Windows NT kernel code is not directly portable.

The MT90528 ISR code is designed for MT90528 evaluation boards with an ISA interrupt scheme. Again, the code may be used as an example of how to use the device in this physical setup.

There are two distinct parts to the interrupt servicing mechanism - the kernel device driver interrupt service routine (ISR) and a user application interrupt service thread. The device driver ISR signals the user application interrupt service thread when action is required. The user application interrupt service thread then handles the MT90528 device interrupt.

The device driver ISR masks interrupt sources until the user application interrupt service thread can service and unmask them.

The speed of the target WinNT system is sufficient to service interrupts for the testing and verification of the MT90528 device. Further work may be required to tune the system (both device drivers and user application) to account for a fully loaded NT system (for example a fileserver configuration) and a SAR implementation under heavy traffic and fault conditions.

The user application interrupt service thread services all known interrupts before re-enabling the main board interrupt.

5.5.2 Design Overview

5.5.2.1 Interrupt Service Routine (Windows NT kernel device driver)

The interrupt service routine is found in the MT90528 kernel device driver and in the sample user application (isthread.c). The user application service thread makes calls to ISR handler functions in the API layer.

The MT90528 ISR also assumes only one MT90528 device exists on one MT90528 evaluation board. No facility is provided to handle multiple interrupts on multiple devices, or for cascaded interrupts (shared interrupt for multiple devices). The selected (by jumper on the ISA interface board on the PC) ISA IRQ value is manually set in the registry - no auto sensing "plug and play" capability is provided. The MT90528 ISR co-exists with other Zarlink Semiconductor cards, but does not share the same interrupt or the Zarlink Semiconductor Board Interrupt driver.

The device driver ISR is called when the MT90528 interrupt line is asserted. It is a level triggered interrupt.

The interrupt service sequence is as follows:

1. The evaluation board interrupt enable bit is checked for enabled board interrupts.
2. The evaluation board interrupt is verified as a valid MT90528 device interrupt.
3. The Main Status and Interrupt Enable Registers on the MT90528 device are checked to determine if a device module interrupt has occurred and device interrupts are enabled.
4. If a valid device interrupt has occurred, the evaluation board interrupts are disabled and the kernel ISR sends an event to the waiting user application interrupt service thread.
5. The user interrupt service thread checks the Main Status Register and calls the API ISR handler routines that are associated to the type of interrupts on the MT90528 device.
6. Each of the ISR handler routines determines what sub interrupts occurred for a specific module on the MT90528 device.
7. The execution returns back to the user application interrupt service thread immediately with an error return code, if an error occurs during the handling of the sub interrupts.
8. If no error, sub interrupt information is filled into an info structure and the sub interrupts are de-asserted.
9. Control is returned back to the user application interrupt service thread.
10. All current sub interrupts are handled and cleared for each of the affected device modules.
11. The user ISR synchronous thread then enables the evaluation board interrupts and the ISR IRQ thread waits for the next IRQ event from the kernel ISR.
12. This process continues as long as interrupt events come in from the kernel ISR and the user ISR IRQ/Sync threads and the user application remain running.

The internal design of the device driver ensures that the actual device driver interrupt service routine is called at a high priority. The ISR starts a deferred procedure call - DPCforIsr (which executes at a system-defined priority) which, in turn starts a DPC thread, whose priority level may be altered. This DPC thread priority allows the system to be tuned if necessary.

5.5.2.2 User Application Interrupt Service Thread (Win32)

The user application must first unmask the device interrupt sources by calling the API initialization routine (Mt90528InitializeApi). User application interrupt and synchronous service threads are created which wait for a single IRQ event to occur.

The Win32 function WaitforSingleObject (or other associated Win32 functions) should be used to wait for the interrupt signal.

Upon receiving the event signal, the application ISR IRQ/Sync threads:

1. determine the exact source(s) of the interrupt by examining the appropriate IRQ status registers
2. handles the type of interrupt (changes a read pointer or writes data into an info structure).
3. clears the interrupt source(s) (if necessary)
4. re-enables the board interrupts after the IRQ thread enters WaitforSingleObject.

Handling the interrupt and clearing the interrupt source are provided by ISR routines in the API. See C313BCS3 MT90528 API Specification concerning the interrupt handlers.

5.6 Board Low Level I/O Interface

5.6.1 General

The board low-level I/O interface is provided as a single source code file, separate from the API layer.

5.6.2 Service Provided

The board low-level interface is provided as a porting point for other operating systems and I/O access methods. The board low-level I/O interface contains the calls to the Windows NT device drivers.

The lower layer file contains a driver emulation to test the upper layer software when hardware does not exist or is not available (e.g. a memory map on the PC).

5.6.3 Details

The following functions are provided in the board low-level I/O interface:

Function	Description
IoMemAccess	used to access (read or write) a memory mapped region
IoPortAccess	used to access (read or write) an I/O port
ReadFromDriver	calls either IoMemAccess or IoPortAccess for reading
WriteToDriver	calls either IoMemAccess or IoPortAccess for writing
ReadBoardAddress	processes the board address and calls ReadFromDriver
WriteBoardAddress	processes the board address and calls WriteToDriver
OpenDeviceDriver	calls IoMemAccess or IoPortAccess to open a device driver
OpenBoardDriver	error checking and calls OpenDeviceDriver
CloseBoardDriver	error checking and calls IoMemAccess or IoPortAccess to close a device driver
GetBoardDriverHandle	passes back the opened device driver handle
CreateIoLock	creates and passes back an I/O lock handle
CloseIoLock	closes an I/O lock
AcquireIoLock	waits for and acquires an I/O lock

Table 9 - Low-Level Functions

Function	Description
ReleaseIoLock	releases an I/O lock
GetIoLockHandle	passes back the created I/O lock handle
CreateBoardIrqEvent	creates and passes back an interrupt event handle
CloseBoardIrqEvent	closes the interrupt event
EnableBoardInterrupts	enables the board interrupts
SavePortDeviceAddress	saves the device address in the device driver extension

Table 9 - Low-Level Functions

All simple I/O requests use IoMemAccess or IoPortAccess. The WIN32 API DeviceIoControl function is called from both functions to pass data to/from the device driver. A Windows NT handle passed to these functions defines which driver is accessed.

5.6.4 Portability

To use the low level interface in a different O/S other than Windows NT, the DeviceIoControl calls should be replaced with appropriate I/O access functions provided by the target O/S. For example, in MS-DOS inp() and outp() functions could be substituted. (Important Note: The DeviceIoControl function is a part of the Win32 API - and therefore is supported by Windows 95. However, the Windows NT drv90528 device driver (to which the DeviceIOControl function passes information) is NOT compatible with Windows 95. Customers using Windows 95 need to select the compile time "PLATFORM" flag as "WINDOWS95" for drvwin95 I/O functions.

The drv90528.sys device driver must not be started for Windows 95 systems. The drv90528.sys device driver could be converted to a Windows 98 device driver but would require porting work. The low layer interface is most useful to those porting to embedded or real time operating systems).

5.7 Evaluation Board Access

5.7.1 Evaluation Board Registers

There are registers that are board-specific that may or may not control the MT90528 device. Listed are MT90528 device-specific board registers:

Register	Description
Control and Status Registers	
`BOARD_CONTROL_STATUS_REG`	control and status register
IRQ Registers	
`BOARD_IRQ_STATUS_REG`	IRQ status register
`BOARD_IRQ_CONTROL_REG`	IRQ control register

Table 10 - Evaluation Board Registers

Please note that evaluation board registers are written to and read from using the `WriteBoardAddress` and `ReadBoardAddress` routines instead of the read and write memory routines since the addresses are board addresses and not device addresses.

5.8 Application Programs

5.8.1 General

Application programs are provided to the customer to serve as an example of the use of the MT90528 device.

5.8.2 Details

A console-based test application is provided that allows the user to exercise all register and memory accesses. The app (as a reference file) is provided to the customer.

For example (in an NT Command Window):

```
C:\MT90528API\USER> userapp

API Initialization
```

Setting up Utopia for UDT and SDT

```
Setting up TDM port 0 for UDT
Setting up TDM port 1 for SDT


Configuring Data Txsar

Configuring Data Rxsar
TDM monitoring header configured to monitor port 1 as low port
SDT MODE GENERIC E1
STiCLK source for ports 0 to 13: C2
Edges used by BERT 0:
        - sampling: rising edge
        - driving: falling edge
BERT 0 configured to transmit 32-bit pattern: aaaaaaaah
TDM monitoring header configured to monitor port 15 as high port
SDT MODE GENERIC E1
STiCLK source for ports 14 to 27: C2
Edges used by BERT 1:
        - sampling: rising edge
        - driving: falling edge
BERT 1 configured to transmit 32-bit pattern: aaaaaaaah

Press enter to start tests
```

The console-based application tests both device operation and limits and error handling of the user API. See README.txt in the mt90528api/user directory for a more detailed description of userapp.c.

The `userapp` program will be updated to include new tests as the MT90528 API evolves.

An interrupt service thread application (`isthread`) provides the ability to service interrupts in real time. It is based upon Win32 threads. `isthread` is invoked from `userapp` as spawned threads. The interrupt service IRQ/Sync threads may have to be tuned for performance depending on the criticality of servicing interrupts. The code provided is an example of how to service interrupts.

6.0 Glossary

AAL - ATM Adaptation Layer; standardized protocols used to translate higher layer services from multiple applications into the size and format of an ATM cell.

AAL1 - ATM Adaptation Layer 1 used for the transport of constant bit rate, time-dependent traffic (e.g., voice, video); requires transfer of timing information between source and destination; maximum of 47-bytes of user data permitted in payload as an additional header byte is required to provide sequencing information.

Asynchronous - 1. Not synchronous; not periodic. 2. The temporal property of being sourced from independent timing references. Asynchronous signals have different frequencies, and no fixed phase relationship. 3. In telecom, data which is not synchronized to the public network clock. 4. The condition or state when an entity is unable to determine, prior to its occurrence, exactly when an event transpires.

ATM - Asynchronous Transfer Mode; a method in which information to be transferred is organized into fixed-length cells; asynchronous in the sense that the recurrence of cells containing information from an individual user is not necessarily periodic. (While ATM cells are transmitted synchronously to maintain clock between sender and receiver, the sender transmits data cells when it has something to send and transmits empty cells when idle, and is not limited to transmitting data every Nth cell.)

Cell - Fixed-size information package consisting of 53 bytes (octets) of data; of these, 5 bytes represent the cell header and 48 bytes carry the user payload and required overhead. **Note:** If a 16-bit UTOPIA interface is used, each cell is 54 bytes long, composed of 6 bytes of cell header and 48 bytes carrying user payload and required overhead.

CBR - Constant Bit Rate; an ATM service category supporting a constant or guaranteed rate, with timing control and strict performance parameters. Used for services such as voice, video, or circuit emulation.

CDV - Cell Delay Variation; a QoS parameter that measures the peak-to-peak cell delay through the network; results from buffering and cell scheduling.

CES - Circuit Emulation Service; ATM Forum service providing a virtual circuit which emulates the characteristics of a constant bit rate, dedicated-bandwidth circuit (e.g., DS1).

CLP - Cell Loss Priority; a 1-bit field in the ATM cell header that corresponds to the loss priority of a cell; cells with CLP = 1 can be discarded in a congestion situation.

CSI - Convergence Sublayer Indication bit in the AAL1 header byte; when present in an even-numbered cell using SDT, indicates the presence of a pointer byte; used to transport RTS values in odd-numbered cells using SRTS for clock recovery.

DBCES - Dynamic Bandwidth Circuit Emulation Service (also referred to as Dynamic Bandwidth Utilization); ATM Forum service providing a virtual circuit which follows the CES specification for the transport of only the active time slots of a constant bit rate, dedicated-bandwidth circuit.

E1 carrier - Similar to the North American T1, E1 is the European format for digital transmission.

E1 carries signals at 2.048 Mbps (32 channels at 64kbps), versus the T1, which carries signals at 1.544 Mbps (24 channels at 64kbps). E1 and T1 lines may be interconnected for international use.

GFC - Generic Flow Control; 4-bit field in the ATM header used for local functions (not carried end-to-end); when cells are configured for NNI formatting, this value forms the four MSBs of the VPI field.

HEC - Header Error Control; using the fifth octet in the ATM cell header, ATM equipment (usually the PHY) may check for an error and correct the contents of the header; CRC algorithm allows for single-error correction and multiple-error detection.

MIB – Management Information Base. A collection of data presented in objects to the network via SNMP about entities in the network.

NNI - ATM Network Node Interface.

OAM - Operations, Administration and Maintenance; MSB within the PTI field of the ATM cell header which indicates if the ATM cell carries management information such as fault indications.

PHY - Physical Layer; bottom layer of the ATM Reference Model; provides ATM cell transmission over the physical interfaces that interconnect the various ATM devices.

PTI - Payload Type Identifier; 3-bit field in the ATM cell header - MSB indicates if the cell contains OAM information or user data.

RTS - Residual Time Stamp; see SRTS.

SAR - Segmentation and Reassembly; method of partitioning, at the source, frames into ATM cells and reassembling, at the destination, these cells back into information frames; lower sublayer of the AAL which inserts data from the information frames into cells and then adds the required header, trailer, and/or padding bytes to create 48-byte payloads to be transmitted to the ATM layer.

SDT - Structured Data Transfer; format used within AAL1 for blocks consisting of N * 64 kbps channels; blocks are segmented into cells for transfer and additional overhead bytes (pointers) are used to indicate structure boundaries within cells (therefore aiding data recovery).

SNMP – Simple Network Management Protocol. This protocol uses the MIB information (objects).

SRTS - Synchronous Residual Time Stamp; method for clock recovery in which difference signals between a source clock and the network reference clock (time stamps) are transmitted to allow reconstruction of the source clock. The destination reconstructs the source clock based on the time stamps and the network reference clock. (Note that the same network reference clock is required at both ends.)

Synchronous - 1. The temporal property of being sourced from the same timing reference. Synchronous signals have the same frequency, and a fixed (often implied to be zero) phase offset. 2. A mode of transmission in which the sending and receiving terminal equipment are operating continually at the same rate and are maintained in a desired phase relationship by an appropriate means.

T1 carrier - A dedicated phone connection supporting data rates of 1.544Mbps. A T1 line actually consists of 24 individual channels, each of which supports 64kbps. Each 64kbps channel can be configured to carry voice or data traffic. Most telephone companies allow you to buy just some of these individual channels, known as fractional T1 access. T1 lines are sometimes referred to as DS1 lines.

UDT - Unstructured Data Transfer; format used within AAL1 for transmission of user data without regard for structure boundaries (e.g., circuit emulation).

UNI – ATM User Node Interface.

UTOPIA - Universal Test and Operations Physical Interface for ATM; a PHY-level interface to provide connectivity between ATM components.

VC - Virtual Channel; one of several logical connections defined within a virtual path (VP) between two ATM devices; provides sequential, unidirectional transport of ATM cells. Also Virtual Circuit.

VCI - Virtual Channel Identifier; 16-bit value in the ATM cell header that provides a unique identifier for the virtual channel (VC) within a virtual path (VP) that carries a particular cell.

VP - Virtual Path; a unidirectional logical connection between two ATM devices; consists of a set of virtual channels (VC).

VPI - Virtual Path Identifier; 8-bit value (in UNI, 12 bits in NNI) in the ATM cell header that indicates the virtual path (VP) to which a cell belongs.

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

www.zarlink.com

TECHNICAL DOCUMENTATION - NOT FOR RESALE