AcQuisition Technology bv

Headquarters:
Raadhuislaan 27a
5341 GL OSS
THE NETHERLANDS

Postal address:
P.O. Box 627
5340 AP OSS
THE NETHERLANDS

Phone: +31-412-651055
Fax: +31-412-651050
Email: info@acq.nl
WEB: http://www.acq.nl

APIS

AcQ Platform Interface Software

Programmer's Manual

Version 2.1

Disclaimer:

The information in this document has been carefully checked and is believed to be entirely reliable. However, no responsibility is assumed for inaccuracies. AcQuisition Technology does not assume any liability arising out of the application or use of any product or circuit described herein; neither does it convey any license under its patent rights nor the rights of others. AcQuisition Technology products are not designed, intended, or authorized for use as components in systems intended to support or sustain life, or for any other application in which the failure of an AcQuisition Technology product could create a situation where personal injury or death may occur, including, but not limited to AcQuisition Technology products used in defence, transportation, medical or nuclear applications. Should the buyer purchase or use AcQuisition Technology products for any such unintended or unauthorized application, the buyer shall indemnify and hold AcQuisition Technology and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that AcQuisition Technology was negligent regarding the design or manufacture of the part.

Printed in The Netherlands.

CONTENTS

PREFACE

Providing good and consisting software support for standard products is essential for application programmers and system integrators in accomplishing their task.

The use of M-modules is not restricted to one specific environment, but is extended to numerous combinations of buses and operating systems. Standard software for these products must be useable on many platforms and easy to maintain.

In order to provide a good, consistent, maintainable and fast way to support M-modules on many different platforms, AcQuisition Technology has developed APIS.

APIS is short for AcQ's Platform Interface Software. Its mere task is to provide a standard API (Application Program Interface) to the application while taking care of all hardware related matters for the platform involved.

APIS provides an open software standard which is available to every interested user. The use of APIS is not restricted to M-modules, but can be applied in each situation where standard equipment (mezzanine card, plug-in adapter, silicon chip, etc.) has to be controlled over a variable hardware interface.

To ensure coherency when making proprietary platform support, AcQ distributes the APIS Programmers Manual. This manual contains all information needed by programmers to build their own platform support.

This page contains no essential data.

1. INTRODUCTION

1.1. VALIDITY OF THE MANUAL

This manual is of revision 2.1 and describes AcQ Platform Interface Software v2.x.

1.2. PURPOSE

This document describes the concept of APIS, AcQ Platform Interface Software. Furthermore the implementation of APIS for the i4000 in an OS-9 environment is described and information on APIS software distribution is given. This document can serve as an information resource for programming APIS support for a specific platform and for writing of APIS based application software.
The software described in this document handles physical accesses to either a mezzanine module, an on-board chip or any other form of direct memory and register accesses. The goal is that software supplied along with standard hardware (e.g. M-modules), can be used regardless of the platform (e.g. i4000/OS-9) provided that the independent software interface for the specific platform is available.
The main audience are programmers and users of APIS related software.

1.3. SCOPE

The scope of this manual is APIS: AcQ Platform Interface Software. APIS provides a standard interface for application software to access memory and hardware registers. Furthermore APIS offers some basic functions needed to make hardware related software platform independent (e.g. interrupt handling).

1.4. DEFINITIONS, ACRONYMS AND ABBREVIATIONS

AcQ	AcQuisition Technology bv
APIS	AcQ Platform Interface Software
M-module	Mezzanine I/O concept according to the M-module specification
Platform	Combination of hardware and operating system
API	Application Program Interface
OS-9	Realtime operating system from Microware
Windows NT	Operating system by Microsoft
Windows 95/98	Operating system by Microsoft
VxWorks	Realtime operating system by Windriver Systems
Linux	Operating system open source
i4000	AcQ M-module carrier for VMEbus
i2000	AcQ M-module carrier for PCI
i3000	AcQ M-module carrier for CompactPCI
i6030	AcQ VMEbus Processor board with M-module interface
PCI	Peripheral Component Interconnect, a computer expansion standard
CompactPCI	PCI Industrial computer bus standard
VMEbus	Versa Module Eurocard bus, an industrial computer bus standard
IP-module	Industry Pack I/O module

1.5. NOTES CONCERNING THE NOMENCLATURE

Hex numbers are marked with a leading “0x”-sign: for example: 0x20 or 0xff.

File names are represented in italic: *filename.txt*

Code examples are printed in `courier` .

1.6. OVERVIEW

In the next chapter APIS is described in general and an overview of the main features is listed. Chapter 3 contains a functional description of APIS platform support, necessary for writing APIS related software. An example of an APIS platform implementation is covered in chapter 4. Chapter 5 is the annex which contains a bibliography, document history and source code listings of the example software.

2. PRODUCT OVERVIEW

This chapter contains a general overview of APIS.

2.1. INTRODUCTION

AcQ produces and supports a large number of standard M-modules varying from networking and process I/O to motion control applications. Physically, the M-modules are supported by a large number of hardware platforms: VMEbus, PCI, CompactPCI as well as a wide variety of operating systems: OS-9, Windows NT, Linux etc.

APIS offers a way to program platform independent applications, example- and test software for controlling hardware. Application software written for APIS only needs re-compiling for a particular platform and is operational with little effort (provided that the application is operating system independent).

Although APIS is described as a software interface for M-modules it is applicable in a wide variety of implementations, whenever hardware has to be controlled by software. For example PCI cards, IP modules or an integrated device on a processor board.

2.2. TECHNICAL OVERVIEW

! Hardware related software made platform independent

! Eliminates the need for custom device driver(s)

! Application software is portable over various platforms with little effort

! Open software concept

! Generic application program interface

! APIS support is available for a wide variety of platforms

! Supports interrupt handling

! Handles endian conflicts in multiple byte words

This page contains no essential data.

3. Functional Description

This chapter contains a detailed description of the product.

3.1. APIS Concept

Hardware accesses to registers and memory are handled by APIS. Some minor operating system dependent functions frequently used in hardware related software, such as interrupt handling and delay functions are also provided by APIS.
APIS platform support consists of an Application Programming Interface in the form of definition files coded in ANSI-C and platform dependent modules e.g. source files, libraries and/or drivers.
In the most simple outline, a platform dependent APIS module consist of nothing more then macro definitions in which APIS calls are substituted by direct hardware accesses. But in most cases an APIS module will consist of a library with interface routines and in some implementations a device driver is needed for interaction with the operating system.

3.2. Block Diagram

The following block diagram illustrates a simplified APIS based application.

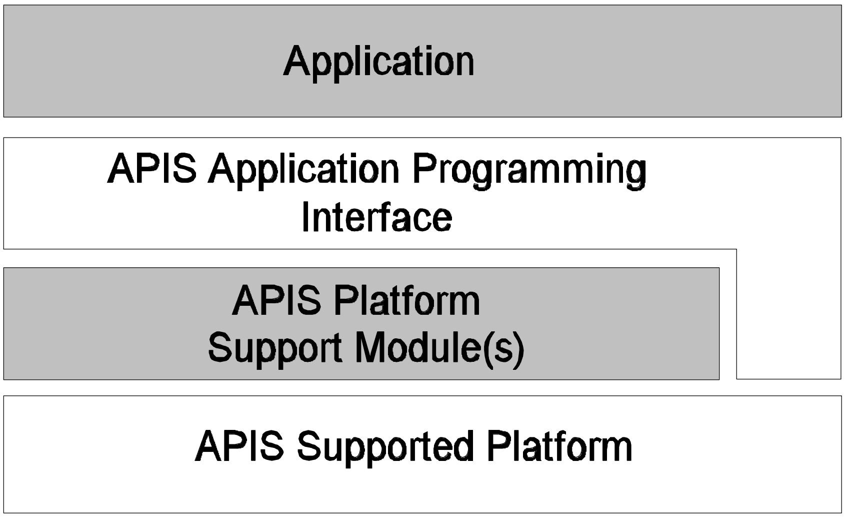
Application
APIS Application Programming Interface
APIS Platform Support Module(s)
APIS Supported Platform

Figure 1 APIS Overview

The block diagram above shows the APIS concept divided in layers. The top layer is the application program. The application program interfaces to the APIS Application Programming Interface (API). The API can call functions of the APIS Platform Support Modules, it can interact with the Operating System and it can directly access the hardware. The APIS Platform Support Module(s) provides functions to the API for accessing the hardware and use of Operating System functions. The bottom layer in the diagram is defined as the APIS platform which is a combination of the hardware and the Operating System.
In the diagram there is no interface between the Application and the Operating System; However the application program can use Operating System functions but in distribution software for AcQ products this has to be avoided.

3.3. APIS Application Programming Interface

In this section the application programming interface (API) for APIS is described.

The Application Programming Interface for APIS is implemented in two ANSI-C coded definition files: *apis.h* which contains general definitions and *platform_apis.h* which contains platform specific definitions and references to the APIS function calls.

The application source file must include the APIS header file *apis.h*. Porting of the application to a platform, consists of re-compiling the source code with a defined pre-processor macro for selection of the used platform. The APIS header file contains generic APIS definitions and includes a platform specific header file according to the platform selection macro.
API calls are translated to the platform specific calls in the APIS header file and the platform specific definition file *platform_apis.h* (*platform* is a name that identifies a hardware and operating system combination, e.g. i4000os9).

Although *apis.h* contains general APIS definitions, it can contain specific definitions that are somehow related between a range of platforms, e.g. M-module carrier boards.
The macro PLATFORM must be defined, either via a pre-processor definition provided at compile time or via a macro-definition in the application source. A valid platform name must be assigned to the macro PLATFORM, the platform name must be defined as a unique decimal number in the file *apis.h*. The macro PLATFORM ensures that the correct platform dependent definition file is included.

3.3.1. Platform Identifier

The platform identifier is a name that is used to refer to a platform, defined as the combination of a hardware product and the operating system environment in which it is used. The platform identifier is used in file names and directories as well as in function names. The platform name used in a function name starts with an underscore (`_platform_function()`).
The platform name must be a unique string of maximal 10 characters, for instance i4000os9 for the platform consisting of the i4000 M-module carrier board and the operating system OS-9.

3.3.2. TYPE DEFINITIONS

Below you can find a table containing type definitions available to APIS based applications and APIS platform implementations.

Name	Type	Description
INT8	char	8-bit signed data
UINT8	unsigned char	8-bit unsigned data
INT16	short	16-bit signed data
UINT16	unsigned short	16-bit unsigned data
INT32	long	32-bit signed data
UINT32	unsigned long	32-bit unsigned data
PHA8	volatile unsigned char*	8-bit physical access
PHA16	volatile unsigned short*	16-bit physical access
PHA32	volatile unsigned long*	32-bit physical access
APIS_PATH	unsigned long	APIS physical path ID
APIS_HANDLE	void *	APIS physical path handle
APIS_WIDTH	int	APIS access size in bytes

Note: This list can be expanded at any time however existing types must NOT be removed or changed.

For details refer to *apis.h* in the appendix.

3.3.3. APIS Error Codes

If the execution of an APIS function is successful the function returns zero, if not the function returns an APIS error code.

APIS error codes are 16-bit wide and are referred to with a symbolic name: APIS_Exxxxxxxx, where xxxxxxxx is a short description of 8 characters max. The most significant bit of an error code must always be cleared.

The table below gives an overview of the possible errors codes.

Symbolic name	Code	Description
APIS_NOERR	0x0000	no error
APIS_ENOTSUP	0x0001	function not supported
APIS_EPARAM	0x0010	bad parameter
APIS_EPARMOR	0x0011	parameter out of range
APIS_EPERMIT	0x0012	no permission
APIS_EWIDTH	0x0013	invalid data width
APIS_EGOS	0x0014	general operating system error
APIS_EINVREQ	0x0015	invalid request
APIS_ENOMEM	0x0016	no memory available
APIS_EMODERR	0x0017	APIS support module not found
APIS_ENOIRQH	0x0018	no interrupt handler installed
APIS_EINVPATH	0x0019	invalid path ID
APIS_ESIG	0x001a	signal error
APIS_EINVMOD	0x001b	invalid module
APIS_EINVDRV	0x001c	invalid driver
APIS_EOPENDEV	0x001d	error opening device
APIS_ELOCK	0x001e	device is already in use
APIS_EINCDEV	0x001f	incorrect device
APIS_EPCIERR	0x0020	PCI error
APIS_EINVHAND	0x0021	invalid handle
APIS_EINVOFF	0x0022	invalid offset
APIS_EINTRPT	0x0023	interrupt error
APIS_ENIRQIU	0x0028	interrupt in use
APIS_EINVVER	0x0029	invalid APIS version
APIS_ETIMER	0x002a	timeout occurred

Note: This list can be expanded at any time however existing error codes must NOT be removed or changed.

For details refer to *apis.h* in the appendix.

3.3.4. Function References and Macro’s

The platform dependent definition file *platform_apis.h* included by *apis.h* contains references to the APIS functions. If possible a function must be implemented as a macro (results in time-efficient code). It is not allowed to use global variables in a macro definition. The functions listed below are obligated, however it is allowed to add optional functions.

Function	Description
_platform_open()	open a hardware path
_platform_close()	close a hardware path
_platform_read()	perform a single read
_platform_write()	perform a single write
_platform_readblock()	perform a block read
_platform_writeblock()	perform a block write
_platform_readfifo()	perform a fifo read
_platform_writefifo()	perform a fifo write
_platform_irqinstall()	install an interrupt service routine and setup interrupts
_platform_irqremove()	remove interrupt service routine
_platform_criticalcode()	execute user code with (all) interrupts disabled
_platform_waitforirq()	wait for an interrupt
_platform_waitforirqtmd()	timed wait for an interrupt
_platform_delay()	insert a delay
_platform_getversion()	get current APIS version
_platform_checkversion()	check if correct APIS version is used

For a full description of the APIS functions refer to section 3.4.
For details refer to *apis_i4000os9.h* in the appendix.

3.3.5. MULTI-PLATFORM SUPPORT

APIS must support multi-platform usage: An application must be able to access hardware on more than one platform.
The macro APIS_MULTIPLATFORM is provided to use APIS in an environment that contains more than one platform. When only one platform is used, the macro APIS_MULTIPLATFORM must NOT be defined and APIS calls are made with apis_xxx. When more than one platform is used, the macro APIS_MULTIPLATFORM must be defined and calls are made with _platform_xxx.
With multi-platform usage the application software must include both the platform dependent definition files (*platform_apis.h*) and the general definition file *apis.h*.

Example of normal usage:

```
#include <apis.h>

apis_open(...);
```

Example of Multi-Platform usage:

OS-9 system consisting of a i6030 CPU board with 2 M-module sockets and an i4000 M-module carrier board.

```
#define APIS_MULTIPLATFORM
#include <apis.h>
#include <i4000os9_apis.h>
#include <i6030os9_apis.h>

_i4000os9_open(...);
_i6030os9_open(...);
```

The macro APIS_MULTIPLATFORM can be defined, either via a pre-processor definition provided at compile time or via a macro-definition in the application source.
For details on multi-platform facilities refer to *apis_i4000os9.h* in the appendix.

Note: If the macro APIS_MULTIPLATFORM is defined, the previously described macro PLATFORM must not be defined.

3.4. APIS FUNCTIONS DEFINITIONS

This section contains detailed information with respect to the APIS functions that can be used in APIS based applications and that must be provided by the APIS platform support.

For an example of the functions described, refer to *apis_i4000os9.c*.

3.4.1. Open a Hardware Path

Function: apis_open

Description: A physical path will be opened for the requested path ID. The path ID is system dependent and can be a hardware address, port ID or an index number of some kind. The open function returns a handle to the hardware path information structure.The open function has a variable number of arguments of at least two: the hardware path ID and a pointer provided for passing the APIS path handle.The additional open parameters are platform specific but platforms that are somehow related must have the same parameter definitions when possible, for instance platforms consisting of an M-module carrier board must have the same parameters for configuring the M-module access types.

Arguments: APIS_PATH path

The definition of the path ID is platform dependent. A path ID of zero selects the platform default ID.

APIS_HANDLE *handle

Pointer to a hardware path information structure, the information structure is platform dependent and transparent for the application software.

...

Variable argument list. Provided for implementation of platform dependent configuration data.

For platforms consisting of an M-module Carrier Board the following additional arguments are defined:

UINT32 size

The size of the physical memory to be mapped in bytes.

UINT32 mtype

The type of M-module access:

Value	Name	Description
0	A08D16	8-bit addressbus, 16-bit databus
1	A08D32	8-bit addressbus, 32-bit databus
2	A24D16	24-bit addressbus, 16-bit databus
3	A24D32	24-bit addressbus, 32-bit databus

Returns: APIS error code

3.4.2. Close a Hardware Path

Function: apis_close

Description: A previously opened hardware is closed: installed interrupts if any, are removed, allocated memory is freed and links or paths open to Operating System services must be closed.

Arguments: APIS_HANDLE handle

Hardware path handle.

Returns: APIS error code

3.4.3. Single Read

Function: apis_read

Description: Performs a single read operation according to the specified width. The physical path is obtained via the handle and the path information and offset are used to determine the physical address. The data which is read will be passed via the supplied data pointer.

Arguments:

APIS_HANDLE handle
Hardware path handle
APIS_WIDTH width
Access width in number of bytes
UINT32 offset
Memory offset
void *data
pointer to return data

Returns: APIS error code

3.4.4. Single Write

Function: apis_write

Description: Performs a single write operation according to the specified width. The physical path is obtained via the handle and the path information and offset are used to determine the physical address. The supplied data has a variable type and is processed according to the specified width.

Arguments:

APIS_HANDLE handle
Hardware path handle
APIS_WIDTH width
Access width in number of bytes
UINT32 offset
Memory offset
... data
Data of type [width]

Returns: APIS error code

3.4.5. Block Read

Function: apis_readblock

Description: Perform a burst read operation according to the specified width and burst length. The physical path is obtained via the handle and the path information and offset are used to determine the physical address. The data which is read will be passed via the supplied data pointer.

Arguments:
APIS_HANDLE handle
    Hardware path handle
APIS_WIDTH width
    Access width in number of bytes
UINT32 offset
    Memory offset
UINT32 length
    Number of elements to read
void *buffer
    Pointer to data buffer, make sure sufficient memory is available at this location.

Returns: APIS error code

3.4.6. Block Write

Function: apis_writeblock

Description: Perform a burst write operation according to the specified width and burst length. The physical path is obtained via the handle and the path information and offset are used to determine the physical address. The data is obtained from a buffer via the supplied data pointer.

Arguments:
APIS_HANDLE handle
    Hardware path handle
APIS_WIDTH width
    Access width in number of bytes
UINT32 offset
    Memory offset
UINT32 length
    Number of elements to write
void *buffer
    Pointer to data buffer

Returns: APIS error code

3.4.7. FIFO READ

Function: apis_readfifo

Description: Perform a burst read operation according to the specified width and burst length. The physical path is obtained via the handle and the path information and offset are used to determine the physical address. The data which is read will be passed via the supplied data pointer. With the fifo read function the source address will not be incremented.

Arguments: APIS_HANDLE handle
Hardware path handle
APIS_WIDTH width
Access width in number of bytes
UINT32 offset
Memory offset
UINT32 length
Number of elements to read
void *buffer
Pointer to data buffer, make sure sufficient memory is available at this location.

Returns: APIS error code

3.4.8. FIFO WRITE

Function: apis_writefifo

Description: Perform a burst write operation according to the specified width and burst length. The physical path is obtained via the handle and the path information and offset are used to determine the physical address. The data is obtained from a buffer via the supplied data pointer. With the fifo write function the destination address will not be incremented.

Arguments: APIS_HANDLE handle
Hardware path handle
APIS_WIDTH width
Access width in number of bytes
UINT32 offset
Memory offset
UINT32 length
Number of elements to write
void *buffer
Pointer to data buffer

Returns: APIS error code

3.4.9. Install Interrupt Service Routine

Function: apis_irqinstall

Description: An interrupt service routine is installed for the requested path with the requested vector and level. First the interrupt level is checked, if zero then the default level is used.

Next the interrupt vector is evaluated. If the vector is zero the default value is used.

The mode word is platform dependent.

The variable pointer can be used to pass a pointer to the interrupt service routine. This makes it possible to use user variables in the interrupt service routine.

Next the interrupt service routine for this path is installed. The user interrupt service routine receives two arguments, first argument is the handle and the second argument is a pointer to user data.

The user interrupt service routine must return either 0 if the interrupt is handled or -1 if interrupt is not his. If the user interrupt service routine returns 0, a signal must be sent to wake-up a pending apis_waitforirq(). The default vector and level are defined in the platform dependent definition file *platform_apis.h*.

Arguments: APIS_HANDLE handle
Hardware path handle
void *irqh_handler
Pointer to user part of interrupt service routine
int vector
Interrupt vector (0 for default)
int level
Interrupt level (0 for default)
int mode
The mode parameter is provided for configuration of the interrupt controller. The mode is platform dependent. For M-module carrier boards the mode is used to configure the interrupt vector source; This can either be Vector-From-Baseboard (bit #0 cleared) or Vector-From-Module (bit #0 set).
void *var_ptr
The var_ptr is provided to pass user variables to the interrupt service routine.

Returns: APIS error code

3.4.10. Remove Interrupt Service Routine

Function: apis_irqremove

Description: Remove the interrupt service routine for the hardware path indicated by the supplied handle.

Arguments: APIS_HANDLE handle
Hardware path handle

Returns: APIS error code

3.4.11. WAIT FOR INTERRUPT

Function: apis_waitforirq

Description: Suspend current process. When a signal is received that is sent by an APIS interrupt service routine return with APIS_NOERR as result code. If a signal is received not caused by APIS (e.g. keyboard interrupt), the function returns with APIS_ESIG as result code.

Note: Make sure interrupts received before apis_waitforirq is called are not missed.

Arguments: None

Returns: APIS error code

3.4.12. TIMED WAIT FOR INTERRUPT

Function: apis_waitforirqtmd

Description: Suspend current process for a requested time. When a signal is received that is sent by an APIS interrupt service routine return with APIS_NOERR as result code. If a signal is received not caused by APIS (e.g. keyboard interrupt), the function returns with APIS_ESIG as result code. If the requested time has elapsed return with APIS_ETIMER.

Note: Make sure interrupts received before apis_waitforirqtmd is called are not missed.

Arguments: UINT32 time_out
Time-out in milliseconds.

Returns: APIS error code

3.4.13. EXECUTE CRITICAL CODE

Function: apis_criticalcode

Description: A function is executed via the supplied pointer. The function is executed with all interrupts disabled and if possible in kernel mode of the Operating System. This function is provided for execution of critical user code.

Note: The user supplied function must be as short as possible and must not call time consuming routines like printf, getchar etc.

Warning: apis_criticalcode must not be called from within an interrupt service routine or from within a routine called via apis_criticalcode.

Arguments: void *func

Pointer to the user routine

int narg

Number of argurments following in the argument list

...

Variable argument list, the arguments in this list are passed to the user routine.

Returns: APIS error code

3.4.14. DELAY

Function: apis_delay

Description: Suspend the current process for a requested delay. The delay must be provided in milliseconds. The minimum delay time is platform dependent. Although the delay request is passed in milliseconds the resolution and minimum delay might be greater then 1 msec (mostly 10msec).

Arguments: UINT32 dtime

Delay time in milliseconds.

Returns: APIS error code

3.4.15. CHECK VERSION

Checking the APIS version consists of two functions apis_getversion and apis_checkversion.

Function: apis_checkversion

Description: Compare the APIS version defined in the platform dependent header file (e.g. *apis_i4000os9.h*) with the installed APIS version. The APIS version is a string which contains all versions of the components of APIS. For example, the apis_i4000os9 version string looks like this: "LIB:XX_TRAP:XX". The string contains the version of the apis library (vX.X) and the version of the trap handler (vX.X).

Arguments: None

Returns: APIS error code

Function: apis_getversion

Description: Get version of the installed APIS. The version is returned as a string.

Arguments: char *version
Pointer to apis version.

Returns: Nothing

3.5. Endian Issues

Whenever hardware has to be accessed through multiple buses problems may arise with respect to the byte order of multiple byte words. Especially when software must be platform independent. This section covers endian issues as they affect APIS.

3.5.1. Big Endian versus Little Endian Platforms

In general APIS platforms are byte addressable. There are two ways to store words that consists of more than one byte, in memory: Big Endian or Little Endian.
The Big Endian approach uses a byte order in which the most significant byte is stored first (at the lowest address). Little Endian uses a byte order in which the least significant byte is stored first. In other words in Big Endian architectures, the leftmost bytes are most significant, in Little Endian architectures the rightmost bytes are most significant.

The figure below shows Big Endian byte order of the word 0x12345678:

<table>
<tr><td colspan="3">MSByte</td><td colspan="3"></td><td colspan="3"></td><td colspan="3">LSByte</td></tr>
<tr><td>31</td><td></td><td>24</td><td>23</td><td></td><td>16</td><td>15</td><td></td><td>8</td><td>7</td><td></td><td>0</td></tr>
<tr><td colspan="3">0x12</td><td colspan="3">0x34</td><td colspan="3">0x56</td><td colspan="3">0x78</td></tr>
<tr><td colspan="3">address 0x00</td><td colspan="3">address 0x01</td><td colspan="3">address 0x02</td><td colspan="3">address 0x03</td></tr>
<tr><td colspan="6">Upper Word Lane</td><td colspan="6">Lower Word Lane</td></tr>
</table>

The figure below shows Little Endian byte order of the word 0x12345678:

<table>
<tr><td colspan="3">LSByte</td><td colspan="3"></td><td colspan="3"></td><td colspan="3">MSByte</td></tr>
<tr><td>31</td><td></td><td>24</td><td>23</td><td></td><td>16</td><td>15</td><td></td><td>8</td><td>7</td><td></td><td>0</td></tr>
<tr><td colspan="3">0x78</td><td colspan="3">0x56</td><td colspan="3">0x34</td><td colspan="3">0x12</td></tr>
<tr><td colspan="3">address 0x00</td><td colspan="3">address 0x01</td><td colspan="3">address 0x02</td><td colspan="3">address 0x03</td></tr>
<tr><td colspan="6">Lower Word Lane</td><td colspan="6">Upper Word Lane</td></tr>
</table>

Motorola 68K and RISC based machines are Big Endian, Intel i386 and DEC Alpha based machines are Little Endian.
APIS platform support can handle multiple-byte words in either Big Endian or Little Endian representation.
APIS platforms consisting of an M-module carrier board must be Big Endian.

3.5.2. M-module Carrier Boards

This section applies to APIS platforms consisting of an M-module carrier board. APIS for M-module carrier boards must handle multiple-byte words as Big Endian.
Whether the APIS routines should perform byte-swapping to compensate the byte order is a difficult question with no universal answer. The question is complicated by the fact that in some systems data is passed through multiple bus architectures.
What APIS for M-module carriers has to ensure is that data transferred to the M-module in a sequence of words of the same size arrives with Big Endian byte order regardless of the endianess of the machine. Data transferred from the M-module to the host processor memory should arrive with

the byte order of the machine (either Little or Big Endian). **Interpretation of binary items embedded within a data stream should be handled by the application software.**
Problems may arise on a Little Endian system whenever individual bytes are addressed on a 16-bit word or 32-bit word location or if 16-bit words are addressed on a 32-bit word location. Therefore endian conversion must be implemented on APIS platforms that will be used in Little Endian machines.

Below an overview of the possible scenarios with bus-width and access combinations according to the M-module specification can be found, including examples with respect to endian conversion implemented in software.
The described endian conversions must be applied in the access routines of APIS support used in Little Endian machines. The following routines must contain endian conversion when necessary: apis_read, apis_write, apis_readblock, apis_writeblock, apis_readfifo and apis_writefifo.

8-bit word transfer over a 8-bit data bus
Because the bit width of the transfer complies with the bus width no endian conversion is required.

8-bit word transfer over a 16-bit data bus
Since M-modules are Big Endian the most significant byte on a 16-bit word location must be accessed on address offset 0x00 and the least significant byte on address offset 0x01. This means that Little Endian machines need endian conversion of the target address (MSByte at address 0x01 and LSByte at address 0x00).
E.g. `target_address = cpu_address ^ 1;`

8-bit word transfer over a 32-bit data bus
Since M-module are Big Endian the most significant byte on a 16-bit word location must be accessed on address offset 0x00 and the least significant byte on address offset 0x03. This means that Little Endian machines need endian conversion of the target address (MSByte at address 0x03 and LSByte at address 0x00).
E.g. `target_address = cpu_address ^ 3;`

16-bit word transfer over a 8-bit data bus
The 16-bit access must be divided into two 8-bit accesses, mostly this is handled by the CPU or by the bus-hardware. The most significant byte must be transferred to or from address offset 0x00 and the least significant byte must be transferred to or form address offset 0x01 therefore if the machine is Little Endian, the bytes of the 16-bit word need to be swapped either in hardware or in the APIS data transfer functions.
E.g. `target_data = cpu_data << 8 | cpu_data >> 8;`

16-bit word transfer over a 16-bit data bus
Because the bit width of a transfer complies with the bus width no endian conversion is required.

16-bit word transfer over a 32-bit data bus
Since M-modules are Big Endian the most significant 16-bit word on a 32-bit word location must be accessed on address offset 0x00 and the least significant 16-bit word on address offset 0x02. This means that Little Endian machines need endian conversion of the target address (MSword at address 0x02 and LSWord at address 0x00).
E.g. `target_address = cpu_address ^ 2;`

32-bit word transfer over a 8-bit data bus
The 32-bit access must be divided into four 8-bit accesses, this is mostly handled by the CPU or by the bus-hardware. The most significant byte must be transferred to or from address offset 0x00 and the least significant byte must be transferred to or from address offset 0x03 therefore if the machine is Little Endian, the bytes need to be swapped either in hardware or in the APIS data transfer functions.
E.g. `target_data = cpu_data << 24 | (cpu_data << 8 & 0x00ff0000) |`

```
                  (cpu_data >> 8 & 0x0000ff00) | cpu_data >> 24;
```

32-bit word transfer over a 16-bit data bus
The 32-bit access must be divided into two 16-bit accesses, this is mostly handled by the CPU or by the bus-hardware. The most significant 16-bit word must be transferred to or from address offset 0x00 and the least significant 16-bit word must be transferred to or form address offset 0x02 therefore if the machine is Little Endian, the 16-bit words of the 32-bit long word need to be swapped either in hardware or in the APIS data transfer functions.
E.g. `target_data = cpu_data << 16 | cpu_data >> 16;`

32-bit word transfer over a 32-bit data bus
Because the bit width of a transfer complies with the bus width no endian conversion is required.

Conclusion

The byte order of the supplied data is system dependent. An APIS platform is defined as a combination of hardware and an operating system. If APIS support for a Big Endian platform is provided for integration on a Little Endian machine (e.g. i2000/WIN-95) endian conversion must be implemented. If the APIS support for a Big Endian platform is provided for integration on a Big Endian machine (e.g. i4000/OS-9) no endian conversion is needed. Finally if the APIS support for a Big Endian platform is provided for integration on either a Big Endian or a Little Endian machine (e.g. i4000/VxWorks) the endian conversion must be selectable via a pre-processor Macro.

Caution: Although endian conversion is implemented on Little Endian machines problems may arise when data, transferred between APIS and the application is interpreted with another size than the size used in the transfer. For example: if a variable declared as a **long** (32-bit) is transferred in a block- or fifo-transfer operation using 16-bit accesses then the words in the long access end-up being swapped.
To avoid Endian conflicts in generic application software, handle data in block- and fifo-transfer routines conforming the access width of the transfer operation.

3.6. SOFTWARE DISTRIBUTION

This section contains information about the distribution of APIS software by AcQuisition Technology. All platforms supported by AcQuisition Technology will be gathered on a single APIS distribution. However it is possible that specific platform support modules are distributed separately (e.g. Drivers, Kernel Extensions etc.).
The basic software distribution consists of the following directory structure:

```
+---PROJECT                      APIS base distribution
   +---APIS
   |  +---DOC                    APIS documentation
   |  +---SOFTWARE
   |     |  readme.txt           Distribution overview
   |     |
   |     +---COMMON              Common sources and modules
   |     |  +---DEFS             Common definition files
   |     |  |     apis.h
   |     |  |     .....
   |     |  +---OS9TRAP          OS-9 Trap handler
   |     |  |  +---CMDS
   |     |  |        apistrap
   |     |  +---.....
   |     |
   |     +---I4000OS9            i4000/OS-9 Support
   |     |     relnotes.txt      Release notes/version info
   |     |     apis_i4000os9.h
   |     |     apis_i4000os9.c
   |     |
   |     +---I2000DOS            i2000/MsDos Support
   |     |     relnotes.txt
   |     |     apis_i2000os9.h
   |     |     apis_i2000os9.c
   |     |
   |     +–--PLATFORM_X          Platform x support
   |     |     relnotes.txt
   |     |     .....
   |     |
   |     +–--PLATFORM_Y          Platform y support
   |           relnotes.txt
   |           .....
   |
   +–--M3xx
      +---DOC                    M-module documentation
      +---SOFTWARE               M-module software
```

The subdirectory DEFS in the directory COMMON contains *apis.h* and can contain any other definition files that are applicable for more than one platform. The directory OS9TRAP contains the trap-handler that is used by OS-9 platforms. The directories I2000DOS, I6030OS9, PLATFORM_X and PLATFORM_Y contain the unique platform dependent APIS modules and a textfile named *relnotes.txt* that contains the release notes of the APIS and its components, information about generation of application-code based on the APIS modules etc. For an example refer to the appendix.

Note: The APIS directory structure is distributed as a zip-file.

For a description of the platform name used in files and directories refer to section 3.3.1.

This page contains no essential data.

4. APIS PLATFORM SUPPORT EXAMPLE

This chapter contains an example of the APIS implementation for the i4000 M-module carrier board in an OS-9 environment. The name of the platform is ‘i4000os9'. In the appendix the source code listings of the APIS for i4000/OS-9 can be found.

4.1. APPLICATION PROGRAMMING INTERFACE

The application programming interface of the i4000os9 APIS consists of the general definition file *apis.h* and the platform dependent definition file *apis_i4000os9.h*.
For more information on the APIS application programming interface in general refer to section 3.3.
The file *apis_i4000os9.h* contains references to and macro definitions of the supported APIS functions. Furthermore the file contains configurable macros:

Macro Name	Default	Description
APIS_MULTIPLATFORM	undefined	Must be defined for using the APIS in a mult-platform environment.
i4000os9_DEFAULT_BASE	0xff800000	Base address used when the requested path ID is zero.
i4000os9_DEFAULT_VECTOR	100	Interrupt vector used when the requested vector is zero
i4000os9_DEFAULT_LEVEL	2	Interrupt level used when the requested interrupt level is zero

The table below gives an overview of the available APIS functions:

APIS Function	
platform_open	platform_irqinstall
platform_close	platform_irqremove
platform_read	platform_waitforirq
platform_write	platform_waitforirqtmd
platform_readblock	platform_criticalcode
platform_writeblock	platform_delay
platform_readfifo	platform_checkversion
platform_writefifo	platform_getversion

4.2. APIS PLATFORM SUPPORT MODULES

APIS for the i4000/OS-9 consists of two platform support modules:

! *i4000os9_apis.c* and *i4000os9_apis.h* containing the APIS function implementation
! *apistrap*, OS-9 trap-handler for handling interrupts and execution of critical code.

The APIS support library is available in ANSI-C source code: *apis_i4000os9.c.* The source code must be compiled and linked to the application. The trap-handler *apistrap* is provided as an OS-9 module and must be resident in memory or must be loadable via the execution path , details with respect to the *apistrap* trap-handler are beyond the scope of this document.

4.3. TYPE DEFINITIONS AND STRUCTURES

Below you can find a table containing variable types available to APIS based applications and APIS platform implementations.

Name	Type	Description
INT8	char	8-bit signed data
UINT8	unsigned char	8-bit unsigned data
INT16	short	16-bit signed data
UINT16	unsigned short	16-bit unsigned data
INT32	long	32-bit signed data
UINT32	unsigned long	32-bit unsigned data
PHA8	volatile unsigned char *	8-bit physical access
PHA16	volatile unsigned short *	16-bit physical access
PHA32	volatile unsigned long *	32-bit physical access
APIS_PATH	unsigned long	APIS physical path ID
APIS_HANDLE	void *	APIS physical path handle
APIS_WIDTH	int	APIS access size in bytes

For details on definitions refer to the ANSI-C definitions files *apis.h* and *i4000os9_apis.h*.

4.4. Function Reference

In this section the function provided to the application provided by the i4000/OS-9 APIS are described:

_i4000os9_open()

Open hardware path

Syntax: `int _i4000os9_open (APIS_PATH path, APIS_HANDLE *handle, ...);`

Description: A physical path will be opened for the requested hardware address, memory size (msize) and M-module access type (mtype). The supplied path ID must be the physical M-module address. If the path ID is zero the default M-module base address is used. The M-module access parameters (size and mtype) have no function (on the i4000) but must be provided for compatibility reasons. The requested size and mtype are checked against the following i4000 limitations: size, 0x100 max. mtype, only A08D16 supported. This routine will request access permission to the physical memory. Next, memory for the path information is allocated and the path information structure is initialized. If the link count is zero the process links to the *apistrap* trap-handler. The trap-handler runs in system state and will be used for installation of interrupt routines and for execution of critical user code (with interrupts masked). Finally the pointer to the path info structure is passed to the caller via the handle and the link count is incremented.

Arguments: APIS_PATH path
Path ID, must be either a valid M-module address or zero for the default M-module address
APIS_HANDLE *handle
Pointer to hardware-path information structure
... Variable argument list:
UINT32 size
Memory size (0 <= size <= 0x100)
UINT32 mtype
M-module access type (mtype == A08D16)

Returns: APIS error code

_i4000os9_close()

Close hardware path

Syntax: `int _i4000os9_close(APIS_HANDLE handle);`

Description: Decrement link count, disable interrupt, remove interrupt handler, free allocated memory, and return 0 if the link count reaches zero, unlink the trap-handler. When the link count is already zero the function just returns.

Arguments: APIS_HANDLE handle
Hardware path handle

Returns: APIS error code

_i4000os9_read() Perform a single read

Syntax:

```
int _i4000os9_read (APIS_HANDLE handle, APIS_WIDTH width,
UINT32 offset,  void *data);
```

Description: Perform a single read operation according to the specified width. The M-module base address is obtained via the handle and the base address and offset are used to determine the physical address. The data which is read will be passed via the supplied data pointer.

Arguments:

APIS_HANDLE handle
Hardware path handle
APIS_WIDTH width
Access width in number of bytes
UINT32 offset
Memory offset
void *data
Pointer to return data

Returns: APIS error code

_i4000os9_write() Perform a single write

Syntax:

```
int _i4000os9_write (APIS_HANDLE handle, APIS_WIDTH width,
UINT32 offset, ...);
```

Description: Perform a single write operation according to the specified width. The M-module base address is obtained via the handle and the base address and offset are used to determine the physical address. The supplied data has a variable type and is processed according to the specified width.

Arguments:

APIS_HANDLE handle
Hardware path handle
APIS_WIDTH width
Access width in number of bytes
UINT32 offset
Memory offset
... data
Data of type [width]

Returns: APIS error code

_i4000os9_readblock() Perform a read burst

Syntax: `int _i4000os9_readblock (APIS_HANDLE handle, APIS_WIDTH width, UINT32 offset, UINT32 length, void *buffer);`

Description: Perform a burst read operation according to the specified width and burst length. The M-module base address is obtained via the handle and the base address and offset are used to determine the physical address. The data which is read will be passed via the supplied data pointer.

Arguments: APIS_HANDLE handle
Hardware path handle
APIS_WIDTH width
Access size: HW8, HW16, HW32
UINT32 offset
Memory offset
UINT32 length
Number of elements to read
void *buffer
Pointer to data buffer

Returns: APIS error code

_i4000os9_writeblock() Perform a write burst

Syntax: `int _i4000os9_writeblock (APIS_HANDLE handle, APIS_WIDTH width, UINT32 offset, UINT32 length, void *buffer);`

Description: Perform a burst read operation according to the specified width and burst length. The M-module base address is obtained via the handle and the base address and offset are used to determine the physical address. The data is obtained from a buffer via the supplied buffer pointer.

Arguments: APIS_HANDLE handle
Hardware path handle
APIS_WIDTH width
Access size: HW8, HW16, HW32
UINT32 offset
Memory offset
UINT32 length
Number of elements to write
void *buffer
Pointer to data buffer

Returns: APIS error code

_i4000os9_readfifo() Perform a fifo read

Syntax:

```
int _i4000os9_readfifo (APIS_HANDLE handle, APIS_WIDTH width,
UINT32 offset, UINT32 length, void *buffer);
```

Description: Perform a burst read operation according to the specified width and burst length. The M-module base address is obtained via the handle and the base address and offset are used to determine the physical address. The data which is read will be passed via the supplied data pointer. With the fifo read function the source address will not be incremented.

Arguments:

APIS_HANDLE handle
Hardware path handle
APIS_WIDTH width
Access size: HW8, HW16, HW32
UINT32 offset
Memory offset
UINT32 length
Number of elements to read
void *buffer
Pointer to data buffer

Returns: APIS error code

_i4000os9_writefifo() Perform a fifo write

Syntax:

```
int _i4000os9_writefifo (APIS_HANDLE handle, APIS_WIDTH width,
UINT32 offset, UINT32 length, void *buffer);
```

Description: Perform a burst read operation according to the specified width and burst length. The M-module base address is obtained via the handle and the base address and offset are used to determine the physical address. The data is obtained from a buffer via the supplied buffer pointer. With the fifo write function the destination address will not be incremented.

Arguments:

APIS_HANDLE handle
Hardware path handle
APIS_WIDTH width
Access size: HW8, HW16, HW32
UINT32 offset
Memory offset
UINT32 length
Number of elements to write
void *buffer
Pointer to data buffer

Returns: APIS error code

_i4000os9_irqinstall() Install interrupt service routine

Syntax:

```
int _i4000os9_irqinstall (APIS_HANDLE handle, void
*irq_handler, int vector, int level, int mode, void *var_ptr);
```

Description: An interrupt service routine is installed for the requested path with the requested vector and level. First the interrupt level is checked, if zero then the default level is used. Next the interrupt vector is evaluated, if the vector is zero the default value is used.

The mode word is platform dependent, for M-module carriers it is used for configuration of the base-board. If bit #0 of the mode word is set the i4000 will be configured for Vector-From-Module. If bit #0 of the mode word is cleared, the i4000 will be configured for Vector-From-Baseboard and the vector will be stored in the i4000 vector register.

The variable pointer can be used to pass a variable to the interrupt service routine. This makes it possible to use user variables in the interrupt service routine.

The first time this function is called the current process ID is obtained from OS-9 and a signal intercept routine is installed. Next the interrupt service routine for this path is installed with the address of its handle as port variable (via a call to apistrap). Finally the interrupt vector and pointer to the user part of the interrupt service routine are stored in the path info structure, the irq_installed flag in the structure is set and the interrupt is enabled. The user interrupt service routine receives the handle via (d0) and a pointer to user data via (d1) and must return either 0 if interrupt is handled or -1 if interrupt is not his. If the user interrupt service routine returned 0, a signal must be sent to wake up a pending apis_waitforirq() if any.

Arguments:

APIS_HANDLE handle
  Hardware path handle
void *irq_handler
  Pointer to user part of interrupt service routine
int vector
  Interrupt vector (0 for default)
int level
  Interrupt level (0 for default)
int mode
  The mode is platform dependent, for M-modules this can either be Vector-From-Baseboard (bit #0 cleared) or Vector-From-Module (bit #0 set)
void *var_ptr
  The var_ptr parameter is provided to pass user variables to the interrupt service routine.

Returns: APIS error code

_i4000os9_irqremove() Remove interrupt service routine

Syntax: `int _i4000os9_irqremove (APIS_HANDLE handle);`

Description: Disable interrupts on i4000 for the slot indicated by the handle and remove the interrupt service routine.

Arguments: APIS_HANDLE handle
Hardware path handle

Returns: APIS error code

_i4000os9_waitforirq() Wait for Interrupt

Syntax: `int _i4000os9_waitforirq (void);`

Description: First the routine checks if an APIS signal has been received, if so the signal counter is decremented and APIS_NOERR is returned, if not the proccess goes to sleep. When the sleep is interrupted the routine checks if an APIS signal has been received, if so the signal counter is deremented and APIS_NOERR is returned else APIS_ESIG is returned. To prevent signals being missed, decrementing the signal counter is done with signals masked.

Arguments: None

Returns: APIS error code

_i4000os9_waitforirqtmd Timed wait for interrupt

Syntax: `int _i4000os9_waitforirqtmd (UINT32);`

Description: First the routine checks if an APIS signal has been received, if so the signal counter is decremented and APIS_NOERR is returned, if not the proccess goes to sleep. When the sleep is interrupted the routine checks if an APIS signal has been received, if so the signal counter is decremented and APIS_NOERR is returned else APIS_ESIG is returned. To prevent signals being missed, decrementing the signal counter is done with signals masked. When the sleep is not interrupted and a timeout occurs return with APIS_ETIMER.

Arguments: UINT32
Timeout in milliseconds

Returns: APIS error code

_i4000os9_delay

Insert a delay

Syntax: `int _i4000os9_delay(UINT32 dtime);`

Description: Suspend process for a requested delay The delay is provided in msecs but the minimum delay time is one system tick (mostly 10msecs).

Arguments: UINT32 dtime

delay time in [msec]

Returns: APIS error code

_i4000os9_criticalcode()

Execute Critical Code

Syntax: `int _i4000os9_criticalcode(void *func, int narg, ...);`

Description: The supplied function with a maximum of 16 parameters is executed in system state with all interrupts disabled. This routine is provided for execution of critical user code. The application is running in user state, the critical code will be executed in system state via a call to the *apistrap* trap-handler.

Note: The user supplied function must be as short as possible and must not call time consuming routines like printf, getchar etc.

Warning: i4000os9_criticalcode must not be called from within an interrupt service routine or from within a routine called via i4000os9_criticalcode.

Arguments: void *func

Pointer to the user routine

int narg

Number of argurments following in the argument list

...

Variable argument list, the arguments in this list are passed to the user routine.

Returns: APIS error code

_i4000os9_checkversion

Check APIS version

Syntax: `int _i4000os9_checkversion (void);`

Description: Compare the APIS version defined in *apis_i4000os9.h* with the installed APIS version. The apis_i4000os9 version string looks like this: "LIB:XX_TRAP:XX". The string contains the version of the APIS library (vX.X) and the version of the trap handler (vX.X). If the versions of the APIS library and trap handler defined in *apis_i4000os9.h* are the same or lower as the installed library and trap handler return APIS_ENOERR else return APIS_EINVVER.

Arguments: None

Returns: APIS error code

_i4000os9_getversion

Get APIS version

Syntax: `int _i4000os9_getversion (char *version);`

Description: Get the version of the installed library and trap handler.

Arguments: char * version
Pointer to APIS version

Returns: APIS error code

5. Annex

5.1. Bibliography

i4000 Quad M-module carrier for VMEbus
Hardware manual R3.0
AcQuisition Technology bv, P.O. Box 627, 5340AP OSS, The Netherlands.

5.2. Document History

! Version 0.0

First release

! Version 1.0

Stated that the listing of *apis.h* is merely an example, since *apis.h* is likely to be updated whenever an platform support package is added to APIS.

! Version 2.0

New APIS development tree.

Function apis_irqinstall() changed, added an extra parameter to pass an user variable to the interrupt service routine.

! Version 2.1

APIS version check functions added

Timed wait for interrupt function added

5.3. APIS for i4000/OS-9 Demo Implementation

The next sections contain the source code listings of the APIS support package for the i4000/OS-9 platform.

5.3.1. Release Notes

The file *relnotes.h* contains the release notes of i4000os9 APIS.

```
APIS for i4000/OS9 Version 2.2

Release Notes

##########################################################################
Copyright 1999, 2000, 2001 by AcQuisition Technology B.V. (c)
All Rights Reserved
Reproduced Under License

This source code is the proprietary confidential property of
AcQuisition Technology B.V., and is provided to the licensee
for documentation and educational purposes only.  Reproduction,
publication, or any form of distribution to any party other than
the licensee is strictly prohibited.

##########################################################################
# Release 1.0

This document describes the software distribution of APIS for the
i4000 M-module carrier board in an OS-9 environment: i4000os9.
```

```
The overall version of APIS for i4000os9 corresponds to the version
of the release notes.

##########################################################################
# Release 2.0

made compatible for new development tree.
i4000os9_irqinstall has an additional parameter to pass a variable
to user interrupt service routine.

##########################################################################
# Release 2.1

irq_installed flag is now cleared in routine _i4000os9_irqremove()
APIS version check functions added

##########################################################################
# Release 2.2

Timed wait for interrupt routine added.

##########################################################################
# Revision History

Rev  date       comments                                          by
---  --------   ---------------------------------------------     ---
0.0  31-05-99   Proposal                                          sp
1.0  20-07-99   First Release                                     sp
2.0  10-01-00   Made compatible to new development tree
                i4000os9_irqinstall has an additional parameter
                to pass a variable to user ISR                    jg
2.1  15-05-00   irq_installed flag is now cleared in routine
                _i4000os9_irqremove().
                APIS version check added                          jg
2.2  12-04-01   _i4000os9_waitforirqtmd() added. This is a timed
                wait for irq routine.                             jg

##########################################################################
# Documentation

APIS Programmer's Manual R2.1

##########################################################################
# Dependencies

Filename:                    Version  Description
---------------------------------------------------------------------------
COMMON\DEFS\apis.h           1.8      General definition file
COMMON\OS9TRAP\CMDS\apistrap ed. 10   APIS trap-handler for OS-9
I4000OS9\relnotes.txt        2.2      This file
I4000OS9\apis_i4000os9.c     1.3      Platform dependent functions
I4000OS9\apis_i4000os9.h     1.3      Platform dependent definitions

##########################################################################
# Code Generation

APIS for the i4000os9 consists the following parts:

1. Application Programming Interface
-COMMON\DEFS\apis.h                 general definitions, ANSI-C source code
-I4000OS9\apis_i4000os9.h           platform dependent definitions (ANSI-C)
```

2. APIS platform dependent modules

-I4000OS9\apis_i4000os9.c	APIS function implementation, ANSI-C source
-COMMON\OS9TRAP\CMDS\apistrap	OS-9 trap-handler provided as shared object

The application source code must include apis.h and must be compiled with the pre-processor definition: PLATFORM=i4000os9.

The APIS support library is available in ANSI-C source code: apis_i4000os9.c. The source code must be compiled and linked to the application.

The trap-handler apistrap is provided as an OS-9 module and must be resident in memory or must be loadable via the execution path.

Code generation has been verified with the following tools:

Microware Ultra C Compiler. Version 2.1
Copyright 1998 Microware

5.3.2. APPLICATION PROGRAMMING INTERFACE

This section contains the listing of *apis.h* which contains the application programming interface for APIS. The listing is merely an example since *apis.h* is likely to be updated whenever an APIS platform support package is added.

```
/*
 *  File:     apis.h
 *  Revision: 1.8
 *  Date:     27/04/01
 *  Author:   SP
 * ----------------------------------------------------------------------
 *  General Definitions for APIS
 *
 *  This header file contains the general definitions for APIS,
 *  AcQuisition Technology's Platform Interface Software.
 *  This file is generic and must be included by the APIS based
 *  application and by the APIS support software for a platform.
 *
 *  Although this file contains general APIS definitions, it is
 *  allowed to add specific definitions that are somehow related
 *  between a range of platforms, e.g. M-module carrier boards.
 *
 *  When implementing APIS for another platform it is allowed to
 *  add general definitions however it is NOT ALLOWED to alter
 *  any of the current definitions !
 * ----------------------------------------------------------------------
 * Copyright 1999, 2000, 2001 by AcQuisition Technology B.V. (c)
 * All Rights Reserved
 * Reproduced Under License
 *
 * This source code is the proprietary confidential property of
 * AcQuisition Technology B.V., and is provided to the licensee
 * for documentation and educational purposes only.  Reproduction,
 * publication, or any form of distribution to any party other than
 * the licensee is strictly prohibited.
 * ----------------------------------------------------------------------
 * Edition History
 *
 * #    date        Comments                                        by
 * ---  --------    ----------------------------------------------  ----
 * 0.0  31-05-99    Initial version.
 *                  Supported platform(s): i4000/OS-9               sp
 * 0.1  28-06-99    Definitions HW8, HW16 and HW32 removed.
 *                  APIS error list changed.
 *                  APIS_MULTIPLATFORM and PLATFORM defined
 *                  simultaneously is not allowed.
 *                  i2000dos APIS added.                            sp
 * 1.0  20-07-99    First release
 *                  type BOOL removed and TRUE and FALSE defined    sp
 * 1.1  27-07-99    i2000win support added                          jg
 * 1.2  02-09-99    i2000lnx support added                          jg
 * 1.3  03-11-99    SBC060Aos9 support added                        MHAs
 * 1.4  30-11-99    i3000win + i3000lnx386 support added
 *                  i6030OS9 support added
 *                  i2000lnx changed to i2000lnx386                 jg
 * 1.5  07-01-00    Made compatible to new development tree         jg
 * 1.6  02-05-00    Version error added                             jg
 * 1.7  05-03-01    i4000sss support added                          jg
 * 1.8  27-04-01    Error APIS_ETIMER added for timed wait for irq
 *                  routine
 *                  powernecsecos support added
 *                  Platform name changed SBC060Aos9->sbc060aos9    jg
 */
#ifndef INCapisH
#define INCapisH

/* General Definitions */
#ifndef TRUE
#define TRUE            1           /* Boolean TRUE */
#endif
#ifndef FALSE
#define FALSE           0           /* Boolean FALSE */
#endif


/* Currently supported platforms */
#define i4000os9        1           /* i4000/OS-9 */
#define i2000dos        99          /* i2000/MsDOS */
#define i2000win        10          /* i2000/Windows 95/98/NT */
#define i2000lnx386     20          /* i2000/Linux i386*/
#define sbc060aos9      30          /* SBC060A/OS-9 */
#define i3000win        40          /* i3000/Windows 95/98/NT */
#define i3000lnx386     50          /* i3000/Linux i386 */
#define i6030os9        60          /* i6030/OS-9 */
#define i4000sss        70          /* i4000/Solaris 8/SPARC/Solflower */
#define powernecsecos   80          /* PowerNECS/eCos */

/* type definitions
 */
typedef char            INT8;       /* 8-bit signed data */
typedef unsigned char   UINT8;      /* 8-bit unsigned data */
typedef short           INT16;      /* 16-bit signed data */
typedef unsigned short  UINT16;     /* 16-bit unsigned data */
typedef long            INT32;      /* 32-bit signed data */
typedef unsigned long   UINT32;     /* 32-bit unsigned data */

typedef volatile UINT8  * PHA8;     /* 8-bit physical access */
typedef volatile UINT16 * PHA16;    /* 16-bit physical access */
typedef volatile UINT32 * PHA32;    /* 32-bit physical access */

typedef unsigned long   APIS_PATH;      /* APIS physical path ID */
typedef void *          APIS_HANDLE;    /* APIS physical path handle */
typedef int             APIS_WIDTH;     /* APIS access size in bytes */

/* APIS error codes
 *
 * APIS error codes are 16-bit wide and are referred to with
 * a symbolic name: APIS_Exxxxxxxx, where xxxxxxxx is a short
 * description of 8 characters max.
 */
#define APIS_NOERR      0x0000      /* no error */
#define APIS_ENOTSUP    0x0001      /* not supported function */
#define APIS_EPARAM     0x0010      /* bad parameter */
#define APIS_EPARMOR    0x0011      /* parameter out of range */
#define APIS_EPERMIT    0x0012      /* no permission */
#define APIS_EWIDTH     0x0013      /* invalid data width */
#define APIS_EGOS       0x0014      /* general operating system error */
#define APIS_EINVREQ    0x0015      /* invalid request */
#define APIS_ENOMEM     0x0016      /* no memory available */
```

```
#define APIS_EMODERR    0x0017      /* APIS support module not found  */
#define APIS_ENOIRQH    0x0018      /* no interrupt handler installed  */
#define APIS_EINVPATH   0x0019      /* invalid path ID */
#define APIS_ESIG       0x001a      /* signal error */
#define APIS_EINVMOD    0x001b      /* invalid module */
#define APIS_EINVDRV    0x001c      /* invalid driver */
#define APIS_EOPENDEV   0x001d      /* error opening device */
#define APIS_ELOCK      0x001e      /* device is already in use */
#define APIS_EINCDEV    0x001f      /* incorrect device */
#define APIS_EPCIERR    0x0020      /* PCI error */
#define APIS_EINVHND    0x0021      /* invalid handle */
#define APIS_EINVOFF    0x0022      /* invalid offset */
#define APIS_EINTRPT    0x0023      /* interrupt error */
#define APIS_ENIRQIU    0x0028      /* interrupt in use  */
#define APIS_EINVVER    0x0029      /* invalid versions */
#define APIS_ETIMER     0x002a      /* time out occured */

/* APIS M-module carrier specific definitions
 *
 * Access types
 */
#define A08D16      0               /* 8-bit address bus, 16-bit data bus
*/
#define A08D32      1               /* 8-bit address bus, 32-bit data bus
*/
#define A24D16      2               /* 24-bit address bus, 16-bit data bus
*/
#define A24D32      3               /* 24-bit address bus, 32-bit data bus
*/
/*
 * Interrupt Mode
 */
#define VECTOR_FROM_BRD 0x0000      /* vector from carrier board */
#define VECTOR_FROM_MOD 0x0001      /* vector from module */

/*
 * Macro PLATFORM
 *
 * The macro PLATFORM must be defined, either via a pre-processor
 * definition provided at compile time or via a macro-definition
 * in the application source.
 *
 * Macro APIS_MULTIPLATFORM
 *
 * The macro APIS_MULTIPLATFORM is provided to use APIS in
 * an environment that contains more than one platform, e.g.:
 * an OS-9 system consisting of a i6030 CPU board with 2 M-module
 * sockets and an i4000 M-module carrier board.
 * if Multi-platform operation is used the Macro PLATFORM must not
 * be defined
 */
#ifdef APIS_MULTIPLATFORM
    #ifdef PLATFORM
        #error Macro PLATFORM must not be defined with Multi-platform
operation
    #endif
#else
    #ifndef PLATFORM
        #error Macro PLATFORM must be defined
    #else
        #if (PLATFORM == i4000os9)
```

```
            #include "../../I4000OS9/apis_i4000os9.h"
        #elif (PLATFORM == i2000dos)
            #include "../../../APIS/SOFTWARE/I2000DOS/apis_i2000dos.h"
        #elif (PLATFORM == i2000win)
            #include "../../../APIS/SOFTWARE/I2000WIN/apis_i2000win.h"
        #elif (PLATFORM == i2000lnx386)
            #include "../../I2000LNX386/apis_i2000lnx386.h"
        #elif (PLATFORM == i3000win)
            #include "../../../APIS/SOFTWARE/I3000WIN/apis_i3000win.h"
        #elif (PLATFORM == i3000lnx386)
            #include "../../I3000LNX386/apis_i3000lnx386.h"
        #elif (PLATFORM == sbc060aos9)
            #include "../../SBC060AOS9/apis_sbc060aos9.h"
        #elif (PLATFORM == i6030os9)
            #include "../../I6030OS9/apis_i6030os9.h"
        #elif (PLATFORM == i4000sss)
            #include "../../I4000SSS/apis_i4000sss.h"
        #elif (PLATFORM == powernecsecos)
            #include "../../POWERNECSECOS/apis_powernecsecos.h"
        #else
            #error Invalid PLATFORM Macro
        #endif
    #endif
#endif

#endif /* INCapisH */
```

5.3.3. Platform Support Modules

This section contains the source code listings of *apis_i4000os9.h* and *apis_i4000os9.c* .

```
/*
 *  File:     apis_i4000os9.h
 *  Revision: 1.3
 *  Date:     12/04/01
 *  Author:   SP
 * ----------------------------------------------------------------------
 *  Definitions for APIS for i4000/OS-9
 *
 *  This file contains i4000/OS-9 specific definitions for APIS, and must
 *  be included by the apis_i4000os9.c and by apis.h (provided that the
 *  macro PLATFORM is set to i4000os9).
 * ----------------------------------------------------------------------
 * Copyright 1999, 2000, 2001 by AcQuisition Technology B.V. (c)
 * All Rights Reserved
 * Reproduced Under License
 *
 * This source code is the proprietary confidential property of
 * AcQuisition Technology B.V., and is provided to the licensee
 * for documentation and educational purposes only.  Reproduction,
 * publication, or any form of distribution to any party other than
 * the licensee is strictly prohibited.
 * ----------------------------------------------------------------------
 * Edition History
 *
 * #    date        Comments                                        by
 * ---  --------    ----------------------------------------------  ---
 * 0.0  31-05-99    Initial version.                                sp
 * 0.1  28-06-99    Swap routines removed.
 *                  Macro BIGE removed.                             sp
 * 1.0  20-07-99    First release
 *                  Declaration of apis_criticalcode changed        sp
 * 1.1  10-01-00    Declaration of i4000os9_irqinstall() changed    jg
 * 1.2  15-05-00    APIS version check added                        jg
 * 1.3  12-04-01    _i4000os9_waitforirqtmd() added. This is a
 *                  timed wait for irq routine.                     jg
 *
 */
#ifndef INCapis_i4000os9H
#define INCapis_i4000os9H

#define APIS_VERSION_i4000os9 "LIB:13_TRAP:10"

/* The macro APIS_MULTIPLATFORM is provided to use APIS in
 * an environment that contains more than one platform, e.g.:
 * an OS-9 system consisting of a i6030 CPU board with 2 M-module
 * sockets and an i4000 M-module carrier board.
 *
 * When only one platform is used, the macro APIS_MULTIPLATFORM
 * must NOT be defined and APIS calls are made with apis_xxx.
 * e.g.     apis_open(...);
 *
 * When more than one platform is used, the macro APIS_MULTIPLATFORM
 * must be defined and calls are made with _platform_xxx.
 * e.g.     _i4000os9_open(...);
 *          _i6030os9_open(...);
 *
 * The macro APIS_MULTIPLATFORM can be defined, either via a
 * pre-processor definition provided at compile time or via
 * a macro-definition in the application source.
 * The macro PLATFORM must not be defined when APIS_MULTIPLATFORM
 * is defined.
 *
 */
#ifndef APIS_MULTIPLATFORM
#define apis_open           _i4000os9_open          /* open path */
#define apis_close          _i4000os9_close         /* close path */

#define apis_read           _i4000os9_read          /* single read */
#define apis_write          _i4000os9_write         /* single write */

#define apis_readblock      _i4000os9_readblock     /* burst read */
#define apis_writeblock     _i4000os9_writeblock    /* burst write */
#define apis_readfifo       _i4000os9_readfifo      /* fifo read */
#define apis_writefifo      _i4000os9_writefifo     /* fifo write */

#define apis_irqinstall     _i4000os9_irqinstall    /* inst irqhandler */
#define apis_irqremove      _i4000os9_irqremove     /* remove irq */
#define apis_waitforirq     _i4000os9_waitforirq    /* wait for irq */
#define apis_waitforirqtmd  _i4000os9_waitforirqtmd /* timed wait for irq
*/
#define apis_criticalcode   _i4000os9_criticalcode  /* ex. critical code */

#define apis_delay          _i4000os9_delay         /* delay function */

#define apis_getversion     _i4000os9_getversion    /* get APIS version */
#define apis_checkversion   _i4000os9_checkversion  /* check APIS version
*/
#define APIS_VERSION        APIS_VERSION_i4000os9   /* APIS support library
                                                       versions */

#endif

/* Default variables
 *
 * An APIS implementation must contain at least the following
 * default parameters:
 *
 *   1. Physical location, for instance a memory address or a port ID.
 *       The default value is used when the requested path in apis_open()
 *       is zero.
 *   2. Interrupt vector
 *       The default value is used when the requested vector in
 *       apis_irqinstall() is zero.
 *   3. Interrupt level.
 *       The default value is used when the requested level in
 *       apis_irqinstall() is zero.
 */
#define i4000os9_DEFAULT_BASE   0xff800000  /* default base address */
#define i4000os9_VECTOR         100         /* default interrupt vector */
#define i4000os9_LEVEL          2           /* default interrupt level */

/* Function References and/or Function Macros
 *
 * Below the supported APIS calls are referenced or defined
 * as a macro. If possible a function must be implemented as
 * a macro (results in time-efficient code). It is not allowed
 * to use global variables in a macro definition.
 *
 * The functions listed below are obligated, however it is allowed
 * to add optional functions.
 *
 *  _platform_open()            _platform_irqinstall()
 *  _platform_close()           _platform_irqremove()
 *  _platform_read()            _platform_criticalcode()
 *  _platform_write()           _platform_waitforirq()
 *  _platform_readblock()       _platform_waitforirqtmd()
 *  _platform_writeblock()      _platform_delay()
 *  _platform_readfifo()        _platform_getversion()
 *  _platform_writefifo()       _platform_checkversion()
 *
 *
 */
extern int _i4000os9_open (APIS_PATH, APIS_HANDLE *, ...);
extern int _i4000os9_close (APIS_HANDLE);

extern int _i4000os9_readblock (APIS_HANDLE, APIS_WIDTH, UINT32,
                                                        UINT32, void *);
extern int _i4000os9_writeblock (APIS_HANDLE, APIS_WIDTH, UINT32,
                                                        UINT32, void *);
extern int _i4000os9_readfifo (APIS_HANDLE, APIS_WIDTH, UINT32,
                                                        UINT32, void *);
extern int _i4000os9_writefifo (APIS_HANDLE, APIS_WIDTH, UINT32,
                                                        UINT32, void *);
extern int _i4000os9_irqinstall (APIS_HANDLE, void *, int, int, int,
                                                                void *);
extern int _i4000os9_irqremove (APIS_HANDLE);
extern int _i4000os9_criticalcode (void *, int, ...);
extern int _i4000os9_waitforirq (void);
extern int _i4000os9_waitforirqtmd (UINT32);
extern int _i4000os9_delay (UINT32);

extern void _i4000os9_getversion(char *);

int _i4000os9_cmpversion(char *);

#define _i4000os9_checkversion()\
_i4000os9_cmpversion(APIS_VERSION_i4000os9)


/*
 * Macro:       _i4000os9_read
 *
 * Description: Perform a single read operation according to
 *              the specified width.
 *              The M-module base address is obtained via the handle
 *              and the base address and offset are used
 *              to determine the physical address. The data
 *              which is read will be passed via the supplied
 *              data pointer.
 *
 * Parameters:  APIS_HANDLE handle
 *                  hardware path handle
 *              APIS_WIDTH width
 *                  access width: [# of bytes]
 *              UINT32 offset
 *                  memory offset
 *              void *data
 *                   pointer to return data
 *
 * Returns:     APIS error code
 *
 */
#define _i4000os9_read(handle, width, offset, dptr)\
(width==1?(*(char *)((UINT32)dptr)=*(PHA8)((*(PHA32)handle+offset))),0:\
(width==2?(*(short *)((UINT32)dptr)=*(PHA16)((*(PHA32)handle+offset))),0:\
(width==4?(*(long *)((UINT32)dptr)=*(PHA32)((*(PHA32)handle+offset))),0:\
APIS_EWIDTH)))

/*
 * Macro:       _i4000os9_write
 *
 * Description: Perform a single write operation according to
 *              the specified width.
 *              The M-module base address is obtained via the handle
 *              and the base address and offset are used
 *              to determine the physical address.
 *              The supplied data has a variable type and is
 *              processed according to the specified width.
 *
 * Parameters:  APIS_HANDLE handle
 *                  hardware path handle
 *              APIS_WIDTH width
 *                  access width [# of bytes]
 *              UINT32 offset
 *                  memory offset
 *              ... data
 *                  data of type [width]
 *
 * Returns:     APIS error code
 *
 */
#define _i4000os9_write(handle, width, offset, data)\
(width==1?(*(PHA8)((*(PHA32)handle+offset))=(UINT8)data),0:\
(width==2?(*(PHA16)((*(PHA32)handle+offset))=(UINT16)data),0:\
(width==4?(*(PHA32)((*(PHA32)handle+offset))=(UINT32)data),0:\
APIS_EWIDTH)))

#endif /* INCapis_i4000os9H */
```

```
/*
 *  File:     apis_i4000os9.c
 *  Revision: 1.3
 *  Date:     12/04/01
 *  Author:   SP
 * ----------------------------------------------------------------------
 *  APIS for i4000/OS-9
 *
 *  This file contains the APIS entries for the platform i4000/OS-9.
 *  This file along with apis.h and apis_i4000os9.h can be used by
 *  an APIS based OS-9 application for accessing M-modules on an i4000
 *  M-module carrier.
 *  For interrupt handling interrupts the APIS/OS-9 traphandler:
 *  "apistrap" is used.
 *
 * ----------------------------------------------------------------------
 * Copyright 1999, 2000, 2001 by AcQuisition Technology B.V. (c)
 * All Rights Reserved
 * Reproduced Under License
 *
 * This source code is the proprietary confidential property of
 * AcQuisition Technology B.V., and is provided to the licensee
 * for documentation and educational purposes only.  Reproduction,
 * publication, or any form of distribution to any party other than
 * the licensee is strictly prohibited.
 * ----------------------------------------------------------------------
 * Edition History
 *
 * #    date        Comments                                        by
 * ---  ----------  ----------------------------------------------  -----
 * 0.0  31-05-1999  initial version.                                sp
 * 0.1  28-06-1999  APIS_EPARMOR return code added.
 *                  Description of swap macros removed.             sp
 *                  _os_permit call divided into separate calls
 *                  for M-module and i4000 control register
 * 1.0  20-07-1999  First release.
 *                  _i4000os9_criticalcode has an additional
 *                  parameter to indicate the number of arguments   sp
 * 1.1  10-01-2000  _i4000os9_irqinstall() has an additional
 *                  parameter to pass a variable to the interrupt
 *                  service routine                                 jg
 * 1.2  15-05-2000  irq_installed flag is now cleared in routine
 *                  _i4000os9_irqremove()                           MHAs
 *                  APIS version check added                        jg
 * 1.3  12-04-2001  _i4000os9_waitforirqtmd() added. This is a
 *                  timed wait for irq routine.                     jg
 *
 */
#include "../COMMON/DEFS/apis.h"
#include <stdlib.h>
#include <types.h>
#include <sysglob.h>
#include <cglob.h>
#include <const.h>
#include <signal.h>
#include <process.h>
#include <machine/reg.h>
#include <stdarg.h>

/* local type definitions
 */
```

```
typedef struct {
    UINT32 base;                /* M-module base address */
    volatile void *user_irqh;   /* pointer to user interrupt handler */
    volatile void *vptr;        /* pointer to user variable */
    int irq_installed;          /* interrupt installed flag */
    int irq_vector;             /* interrupt vector */
} PATH_INFO;

/* externals
 */
extern void *_glob_data;        /* OS-9 global data */

/* globals
 */
volatile process_id proc_id;    /* process ID */
static volatile int sigcnt;     /* signal counter */
static UINT32 link_count;       /* number of open paths */
static int trapnr;              /* user trap nr. */
static UINT16 traped;           /* Edition of trap handler */

/* forward declarations
 */
int _i4000os9_irqremove (APIS_HANDLE);  /* remove interrupt handler */
static void sighand(int);               /* signal handler */
void irqh (void);                       /* interrupt service routine */

/* local definitions
 */
#define TRAPNAME    "apistrap"          /* name of APIS traphandler */
#define IRQSIG      400                 /* interrupt received signal */
#define INSTIRQ     1                   /* install interrupt */
#define USRFUNC     2                   /* execute user function */
#define LIB_VERSION 13                  /* APIS library version */

/* APIS functions
 */

/*
 * Function:    _i4000os9_open
 *
 * Description: A physical path will be opened for the requested
 *              hardware address, memory size (msize) and
 *              M-module access type (mtype).
 *              The supplied path ID must be the physical M-module
 *              address. If the path ID is zero the default
 *              M-module base address is used.
 *              The M-module access parameters (size and mtype) have
 *              no function (on the i4000) but must be provided for
 *              compatibility reasons.
 *              The requested size and mtype are checked against
 *              the following i4000 limitations:
 *                  size, 0x100 max.
 *                  mtype, only A08D16 supported.
 *
 *              This routine will request access permission to the
 *              the physical memory.
 *              Next, memory for the path information is allocated
 *              and the path information structure is initialized.
 *              If the link count is zero the process links
 *              to the apistrap trap-handler. The trap-handler
 *              runs in system state and will be used for installation
 *              of interrupt routines and for execution critical
 *              user code (with interrupts masked).
 *              Finally the pointer to the path info structure is
 *              passed to the caller via the handle and the
 *              link count is incremented.
 *
 * Parameters:  APIS_PATH path
 *                  path ID, must be either a valid M-module address
 *                  or zero for the default M-module address
 *              APIS_HANDLE *handle
 *                  pointer to hardware-path information structure
 *
 *              ...
 *                  variable argument list
 *                  the number of parameters and their
 *                  type are variable. The type must
 *                  NOT!!! be of the following types:
 *                  - register storage class
 *                  - function type
 *                  - an array type
 *                  - type that is not compatible after
 *                      applying the default parameter
 *                      promotions (e.g. char, short
 *                      and float).
 *
 *              The next parameters are variable function
 *              parameters:
 *
 *              UINT32 size
 *                  memory size (0 <= size <= 0x100)
 *              UINT32 mtype
 *                  M-module access type (mtype == A08D16)
 *
 * Returns:     APIS error code
 *
 */
int _i4000os9_open (APIS_PATH path, APIS_HANDLE *handle, ...)
{
    PATH_INFO *pi;                  /* pointer to path information */
    va_list argp;                   /* variable argument pointer */
    void *mem_ptr;                  /* pointer to physical memory */
    int size = 256;                 /* size of physical memory window */
    int mtype = A08D16;             /* M-module access type */
    u_int16 address;                /* storage for lower part of base */

    /* The passed path is a M-module base address. If the
     * the requested path is zero, the default base address
     * for i4000 M-module carrier boards is used
     */
     mem_ptr = path == 0 ? (void *)i4000os9_DEFAULT_BASE : (void *)path;

    /* Verify requested M-module base address,
     * the base address must be one of:
     * 0x_____000 slot 0
     * 0x_____200 slot 1
     * 0x_____400 slot 2
     * 0x_____600 slot 3
     */
    address = (UINT16)mem_ptr&0x7ff;
    if (address != 0x0000 && address != 0x0200 && address != 0x0400
        && address != 0x0600)
        return APIS_EINVPATH;

    va_start(argp, handle);         /* initialize parameter pointer */
    size = va_arg (argp, UINT32);   /* get memory window size */
    mtype = va_arg (argp, UINT32);  /* get M-module type */
    va_end(argp);                   /* end variable arguments */

    /* The requested memory size and M-module access type
     * are verified.
     * On the i4000 only A08D16 is supported with a maximum
     * memory size of 256.
     */
    if (size > 256)
        return APIS_EPARMOR;        /* parameter out of range */
    if (mtype != A08D16)
        return APIS_EPARAM;         /* bad parameter */

    /* Request read/write access to the M-module.
     */
    if (_os_permit(mem_ptr, size, 3, 0) != 0)
        return APIS_EPERMIT;        /* No permission */

    /* Request read/write access to the i4000 control registers
     * corresponding to the requested M-module slot.
     */
    if (_os_permit((void *)((UINT32)mem_ptr+0x100UL), 4, 3, 0) != 0)
        return APIS_EPERMIT;        /* No permission */


    /* Allocate memory (filled with zeroes) for path information
     * and pass pointer to the data block via handle
     */
    if ((pi = (PATH_INFO *)calloc(1, sizeof(PATH_INFO))) == 0)
        return APIS_ENOMEM;

    pi->base = (UINT32)mem_ptr;     /* store module base address */

    /* Link to trap handler (only once)
     * the traphandler is provided for installing the interrupt
     * service routine and for setting the interrupt mask
     *
     * A user traphandler requires a trapnumber. Trapnumbers
     * 1 to 12 are available for user traps, the following
     * code will use the first available trapnumber.
     * If no trapnumber is available or if the trap if the
     * traphandler is not in memory an error code is returned.
     */
    if (link_count == 0) {
        sigcnt = 0;
        for (trapnr = 1; trapnr < 13; trapnr++)
            if (tlink(trapnr, TRAPNAME, &traped) == 0)
                break;
        if (trapnr == 13) {
            free((void*)pi);
            return APIS_EMODERR;
        }
    }

    *handle = (APIS_HANDLE)pi;      /* pass handle */
    link_count++;                   /* increment link count */
    return APIS_NOERR;
}

/*
 * Function:    _i4000os9_close
 *
 * Description: Decrement link count, disable interrupt,
 *              remove interrupt handler, free allocated memory,
 *              and return APIS_NOERR
 *              if the link count reaches zero, unlink the
 *              traphandler.
 *              if the link count is already zero the function
 *              just returns.
 *
 * Parameters:  APIS_HANDLE handle
 *                  hardware path handle
 *
 * Returns:     APIS_NOERR
 *
 */
int _i4000os9_close(APIS_HANDLE handle)
{
    if (link_count == 0)            /* if no paths open */
        return APIS_NOERR;                   /* then just return */

    link_count--;                   /* decrement link count */
    _i4000os9_irqremove (handle);   /* remove interrupt handle */
    free((void *)handle);           /* free handle */

    /*return APIS_NOERR;*/

    if (link_count == 0)
        tlink(trapnr,0);            /* unlink trap handler */

    return APIS_NOERR;
}


/*
 * Macro:       _i4000os9_read
 *
 *  Defined in i4000os9.h !!!
 *
 * Description: Perform a single read operation according to
 *              the specified width.
 *              The M-module base address is obtained via the handle
 *              and the base address and offset are used
 *              to determine the physical address. The data
 *              which is read will be passed via the supplied
 *              data pointer.
 *
 * Parameters:  APIS_HANDLE handle
 *                  hardware path handle
 *              APIS_WIDTH width
 *                  access width in number of bytes
 *              UINT32 offset
 *                  memory offset
 *              void *data
 *                  pointer to return data
 *
 * Returns:     APIS error code
 *
 */

/*
 * Macro:       _i4000os9_write
 *
 *  Defined in i4000os9.h !!!
 *
 * Description: Perform a single write operation according to
 *              the specified width.
 *              The M-module base address is obtained via the handle
 *              and the base address and offset are used
 *              to determine the physical address.
 *              The supplied data has a variable type and is
 *              processed according to the specified width.
 *
 * Parameters:  APIS_HANDLE handle
 *                  hardware path handle
 *              APIS_WIDTH width
 *                  access width in number of bytes
 *              UINT32 offset
 *                  memory offset
 *              ... data
 *                  data of type [width]
 *
 * Returns:     APIS error code
 *
 */

/*
 * Function:    _i4000os9_readblock
 *
 * Description: Perform a burst read operation according to
 *              the specified width and burst length.
 *              The M-module base address is obtained via the handle
 *              and the base address and offset are used
 *              to determine the physical address. The data
 *              which is read will be passed via the supplied
 *              data pointer.
 *
 * Parameters:  APIS_HANDLE handle
 *                  hardware path handle
 *              APIS_WIDTH width
 *                  access size in number of bytes
 *              UINT32 offset
 *                  memory offset
 *              UINT32 length
 *                  number of elements to read
 *              void *buffer
 *                  pointer to data buffer
 *
 * Returns:     APIS error code
 *
 */
int _i4000os9_readblock (APIS_HANDLE handle, APIS_WIDTH width,
                    UINT32 offset, UINT32 length, void *buffer)
{
    PATH_INFO *pi = (PATH_INFO *)handle;    /* initialize pi pointer */

    UINT32 i;                   /* general index */
    UINT8 *src8, *dst8;         /* byte-copy pointers */
    UINT16 *src16, *dst16;      /* word-copy pointers */
    UINT32 *src32, *dst32;      /* long-word-copy pointers */

    switch (width)
    {
        case 1:
            src8 = (UINT8 *)(pi->base + offset);
            dst8 = (UINT8 *)buffer;
            for (i = 0; i < length; i++)
                *dst8++ = *(PHA8)src8++;
            break;
        case 2:
            src16 = (UINT16 *)(pi->base + offset);
            dst16 = (UINT16 *)buffer;
            for (i = 0; i < length; i++)
                *dst16++ = *(PHA16)src16++;
            break;
        case 4:
            src32 = (UINT32 *)(pi->base + offset);
            dst32 = (UINT32 *)buffer;
            for (i = 0; i < length; i++)
                *dst32++ = *(PHA32)src32++;
            break;
        default:
            return APIS_EWIDTH;
    }
    return APIS_NOERR;
}


/*
 * Function:    _i4000os9_writeblock
 *
 * Description: Perform a burst write operation according to
 *              the specified width and burst length.
 *              The M-module base address is obtained via the handle
 *              and the base address and offset are used
 *              to determine the physical address. The data
 *              is obtained from a buffer via the supplied
 *              buffer pointer.
 *
 * Parameters:  APIS_HANDLE handle
 *                  hardware path handle
 *              APIS_WIDTH width
 *                  access size in number of bytes
 *              UINT32 offset
 *                  memory offset
 *              UINT32 length
 *                  number of elements to write
 *              void *buffer
 *                  pointer to data buffer;
 *
 * Returns:     APIS error code
 *
 */
int _i4000os9_writeblock (APIS_HANDLE handle, APIS_WIDTH width,
                    UINT32 offset, UINT32 length, void *buffer)
{
    PATH_INFO *pi = (PATH_INFO *)handle;    /* initialize pi pointer */

    UINT32 i;                   /* general index */
    UINT8 *src8, *dst8;         /* byte-copy pointers */
    UINT16 *src16, *dst16;      /* word-copy pointers */
    UINT32 *src32, *dst32;      /* long-word-copy pointers */

    switch (width)
    {
        case 1:
            src8 = (UINT8 *)buffer;
            dst8 = (UINT8 *)(pi->base + offset);
            for (i = 0; i < length; i++)
                *(PHA8)dst8++ = *src8++;
            break;
        case 2:
            src16 = (UINT16 *)buffer;
            dst16 = (UINT16 *)(pi->base + offset);
            for (i = 0; i < length; i++)
                *(PHA16)dst16++ = *src16++;
            break;
        case 4:
            src32 = (UINT32 *)buffer;
            dst32 = (UINT32 *)(pi->base + offset);
            for (i = 0; i < length; i++)
                *(PHA32)dst32++ = *src32++;
            break;
        default:
            return APIS_EWIDTH;
    }
    return APIS_NOERR;
}



/*
 * Function:    _i4000os9_readfifo
 *
 * Description: Perform a fifo read operation according to
 *              the specified width and burst length.
 *              The M-module base address is obtained via the handle
 *              and the base address and offset are used
 *              to determine the physical address. The data
 *              which is read will be passed via the supplied
 *              data pointer.
 *              With the fifo read function the source address
 *              will not be incremented.
 *
 * Parameters:  APIS_HANDLE handle
 *                  hardware path handle
 *              APIS_WIDTH width
 *                  access size in number of bytes
 *              UINT32 offset
 *                  memory offset
 *              UINT32 length
 *                  number of elements to read
 *              void *buffer
 *                  pointer to data buffer;
 *
 * Returns:     APIS error code
 *
 */
int _i4000os9_readfifo (APIS_HANDLE handle, APIS_WIDTH width,
                    UINT32 offset, UINT32 length, void *buffer)
```

```
{
    PATH_INFO *pi = (PATH_INFO *)handle;    /* initialize pi pointer */

    UINT32 i;               /* general index */
    UINT8 *dst8;            /* byte-copy pointers */
    UINT16 *dst16;          /* word-copy pointers */
    UINT32 *dst32;          /* long-word-copy pointers */

    switch (width)
    {
        case 1:
            dst8 = (UINT8 *)buffer;
            for (i = 0; i < length; i++)
                *dst8++ = *(PHA8)(pi->base + offset);
            break;
        case 2:
            dst16 = (UINT16 *)buffer;
            for (i = 0; i < length; i++)
                *dst16++ = *(PHA16)(pi->base + offset);
            break;
        case 4:
            dst32 = (UINT32 *)buffer;
            for (i = 0; i < length; i++)
                *dst32++ = *(PHA32)(pi->base + offset);
            break;
        default:
            return APIS_EWIDTH;
    }
    return APIS_NOERR;
}


/*
 * Function:    _i4000os9_writefifo
 *
 * Description: Perform a fifo write operation according to
 *              the specified width and burst length.
 *              The M-module base address is obtained via the handle
 *              and the base address and offset are used
 *              to determine the physical address. The data
 *              is obtained from a buffer via the supplied
 *              buffer pointer.
 *              With the fifo write function the destination address
 *              will not be incremented.
 *
 * Parameters:  APIS_HANDLE handle
 *                  hardware path handle
 *              APIS_WIDTH width
 *                  access size in number of bytes
 *              UINT32 offset
 *                  memory offset
 *              UINT32 length
 *                  number of elements to write
 *              void *buffer
 *                  pointer to data buffer;
 *
 * Returns:     APIS error code
 *
 */
int _i4000os9_writefifo (APIS_HANDLE handle, APIS_WIDTH width,
                    UINT32 offset, UINT32 length, void *buffer)
```

```
{
    PATH_INFO *pi = (PATH_INFO *)handle;    /* initialize pi pointer */

    UINT32 i;                   /* general index */
    UINT8 *src8;                /* byte-copy pointers */
    UINT16 *src16;              /* word-copy pointers */
    UINT32 *src32;              /* long-word-copy pointers */

    switch (width)
    {
        case 1:
            src8 = (UINT8 *)buffer;
            for (i = 0; i < length; i++)
                *(PHA8)(pi->base + offset) = *src8++;
            break;
        case 2:
            src16 = (UINT16 *)buffer;
            for (i = 0; i < length; i++)
                *(PHA16)(pi->base + offset) = *src16++;
            break;
        case 4:
            src32 = (UINT32 *)buffer;
            for (i = 0; i < length; i++)
                *(PHA32)(pi->base + offset) = *src32++;
            break;
        default:
            return APIS_EWIDTH;
    }
    return APIS_NOERR;
}


/*
 * Function:    _i4000os9_irqinstall
 *
 * Description: A interrupt service routine is installed for the
 *              requested path with the requested vector and level.
 *
 *              First the interrupt level is checked, if zero then
 *              the default level is used. Next the interrupt vector
 *              is evaluated, if the vector is zero the default
 *              value is used.
 *              The mode word is platform dependent, for M-module
 *              carriers it is used for configuration of the base-board.
 *              If bit #0 of the mode word is set the i4000 will
 *              be configured for Vector-From-Fodule. If bit #0
 *              of the mode word is cleared, the i4000 will be configured
 *              for Vector-From-Baseboard and the vector will be
 *              stored in the i4000 vector register.
 *
 *              The first time this function is called the current
 *              process ID is obtained from OS-9 and a signal intercept
 *              routine is installed.
 *              Next the interrupt service routine for this path
 *              is installed with the address of its handle as
 *              port variable (via a call to apistrap).
 *
 *              Finally the interrupt vector and pointer to the
 *              user part of the interrupt service routine are stored
 *              in the path info structure, the irq_installed
 *              flag in the structure is set and the interrupt is enabled.
 *
 *              The user interrupt service routine receives the handle
 *              via (d0) and must return either 0 if interrupt is handled
 *              or -1 if interrupt is not his.
 *
 * Parameters:  APIS_HANDLE handle
 *                  hardware path handle
 *              void *irqh_handler
 *                  pointer to user part of interrupt service routine
 *              int vector
 *                  interrupt vector (0 for default)
 *              int level
 *                  interrupt level  (0 for default)
 *              int mode
 *                  the mode is platform dependent, for M-modules this
 *                  can either be Vector-From-Baseboard (bit #0 cleared)
 *                  or Vector-From-Module (bit #0 set)
 *              void *vptr
 *                  pointer to a variable which can be passed to user IRQH
 *
 * Returns:     APIS error code
 */
int _i4000os9_irqinstall (APIS_HANDLE handle, void *irq_handler,
    int vector, int level, int mode, void *vptr)
{
    static int first_call = TRUE;           /* first call flag */
    PATH_INFO *pi=(PATH_INFO *)handle;      /* initialize pi pointer */
    u_int16 nil;                            /* nil variable */
    UINT8 cbyte;                            /* i4000 config byte */

    if (pi->irq_installed == TRUE)          /* if already installed */
        return APIS_EINVREQ;                /* return error code */

    /* assign interrupt level
     * if requested level is zero then use default
     * level
     */
    level = level == 0 ? i4000os9_LEVEL : level&7;

    if (vector == 0)                        /* if requested vector 0 */
        vector = i4000os9_VECTOR;           /* then use default vector */

    if ((mode&1) == 1)                      /* if bit #0 set configure */
        cbyte = 0x30+level;                 /* for Vector-From-Module */
    else                                    /* if not the i4000 */
    {
        *(PHA8)(pi->base+0x103) = vector;   /* program vector, the i4000 */
        cbyte = 0x10+level;                 /* must deliver the vector */
    }

    if (first_call == TRUE)
    {
        first_call = FALSE;

        /* Get and save ID of current process
         *  The process ID will be used by the interrupt service routine
         *  for sending a signal to this process
         */
        if (_os9_id((process_id *)&proc_id, &nil, &nil, &nil) != 0)
            return APIS_EGOS;
        /* Setup signal intercept trap
        */
        if (_os_intercept(sighand, _glob_data) != 0)
            return APIS_EGOS;

    }

    /*
     * Install interrupt service routine via
     * traphandler
     */
    if (tcall(trapnr, INSTIRQ, vector, 1 /* priority */, irqh,
                                     _glob_data, (void *)handle) != 0)
        return APIS_EGOS;

    /* Save user interrupt handler
     * This routine will be jumped to by the irqh()
     */
    pi->user_irqh = (volatile void *)irq_handler;

    pi->irq_vector = vector&0xff;    /* save vector */
    pi->irq_installed = TRUE;        /* interrupt handling ready */
    pi->vptr = vptr;                 /* save pointer to user IRQ variable
*/

    *(PHA8)(pi->base+0x101) = cbyte; /* enable interrupt */
    return APIS_NOERR;
}

/*
 * Function:    _i4000os9_irqremove
 *
 * Description: Disable interrupt on i4000 for the slot indicated
 *              by the handle and remove the interrupt service routine.
 *
 * Parameters:  APIS_HANDLE handle
 *                  hardware path handle
 *
 * Returns:     APIS error code
 */
int _i4000os9_irqremove (APIS_HANDLE handle)
{
    PATH_INFO *pi = (PATH_INFO *)handle;    /* initialize pi pointer */

    if (pi->irq_installed == FALSE)         /* if no interrupt installed */
        return APIS_NOERR;                           /* then just return */

    *(PHA8)(pi->base+0x101) = 0;            /* disable i4000 interrupt */

    /* remove interrupt service routine
     */
    if (tcall(trapnr, INSTIRQ, pi->irq_vector, 0, 0,
                                        _glob_data, (void *)handle) != 0)
        return APIS_EGOS;

    pi->irq_installed = FALSE;              /* clear irq installed flag */
    return APIS_NOERR;
}


/*
```

```
 * Function:    _i4000os9_waitforirq
 *
 * Description: First check if an APIS signal has been received, if so
 *              then decrement the signal counter and return APIS_NOERR.
 *              if not goto sleep. When the sleep is interrupted
 *              check if an APIS signal has been received, if so
 *              decrement the signal counter and return APIS_NOERR
 *              if not return APIS_ESIG.
 *              To prevent signals being missed, decrementing the signal
 *              counter must be done with signals masked.
 *
 * Parameters:  none
 *
 * Returns:     APIS error code
 */
int _i4000os9_waitforirq (void)
{
    u_int32 infinite = 0;   /* variable for _os9_sleep */

    _os_sigmask(1);         /* mask signals */
    if (sigcnt != 0) {      /* signal already received ? */
        sigcnt--;           /* decrement signal counter */
        _os_sigmask(0);     /* unmask signal */
        return APIS_NOERR;  /* return */
    }
    else {
        /* unmask signals and goto sleep
         * if a singal is received that is not ours (sigcnt == 0)
         * then return APIS_ESIG else return zero
         */
        _os9_sleep (&infinite);
    }

    _os_sigmask(1);         /* mask signals */
    if (sigcnt != 0) {      /* signal already received ? */
        sigcnt--;           /* decrement signal counter */
        _os_sigmask(0);     /* unmask signal */
        return APIS_NOERR;  /* return */
    }

    _os_sigmask(0);         /* unmask signal */
    return APIS_ESIG;
}

/*
 * Function:    _i4000os9_waitforirqtmd
 *
 * Description: Timed wait for irq routine.
 *              First check if an APIS signal has been received, if so
 *              then decrement the signal counter and return APIS_NOERR.
 *              if not goto sleep. When the sleep is interrupted
 *              check if an APIS signal has been received, if so
 *              decrement the signal counter and return APIS_NOERR
 *              if an APIS signal is received return APIS_ESIG. When the
 *              timer is expired without interruption return APIS_ETIMER.
 *              To prevent signals being missed, decrementing the signal
 *              counter must be done with signals masked.
 *
 * Parameters:  UINT32 time_out
 *                  Time out in msecs.
 *
 * Returns:     APIS error code
 */
int _i4000os9_waitforirqtmd (UINT32 time_out)
{
    static u_int16 tcksec = 0;  /* system ticks per second */
    u_int32 ticks_to_sleep;     /* number of ticks to sleep */

    _os_sigmask(1);         /* mask signals */
    if (sigcnt != 0) {      /* signal already received ? */
        sigcnt--;           /* decrement signal counter */
        _os_sigmask(0);     /* unmask signal */
        return APIS_NOERR;  /* return */
    }
    else {
        /* get number of ticks per second
         * from system globals (only the first time)
         */
        if (tcksec == 0)
            if (_os_getsys(0x28, 2, (glob_buff *)&tcksec) != 0)
                return APIS_EGOS;

        /* ticks to sleep := requested delay * ticks/sec (minimum of 1)
         */
        ticks_to_sleep = (time_out * tcksec)/1000+0.5;
        if (ticks_to_sleep != 0)
            _os9_sleep (&ticks_to_sleep);
    }

    _os_sigmask(1);         /* mask signals */
    if (ticks_to_sleep == 0) {
        _os_sigmask(0);
        return APIS_ETIMER;
    }
    if (sigcnt != 0) {      /* signal already received ? */
        sigcnt--;           /* decrement signal counter */
        _os_sigmask(0);     /* unmask signal */
        return APIS_NOERR;  /* return */
    }

    _os_sigmask(0);         /* unmask signal */
    return APIS_ESIG;
}

/*
 * Function:    _i4000os9_criticalcode
 *
 * Description: The supplied function with a variable number
 *              of parameters is executed in system state
 *              with all interrupts disabled.
 *              This routine is provided for execution
 *              of critical user code
 *
 *
 * Parameters:  void *func
 *                  pointer to user routine
 *              int narg
 *                  number of arguments to be passed to the
 *                  user function (0 <= narg < 16)
 *              ...
 *                  variable argument list
 *                  the number of parameters and their
 *                  type are variable. The type must
 *                  NOT!!! be of the following types:
 *                  - register storage class
 *                  - function type
 *                  - an array type
 *                  - type that is not compatible after
 *                      applying the default parameter
 *                      promotions (e.g. char, short
 *                      and float).
 *
 *
 * Returns:     APIS error code
 */
int _i4000os9_criticalcode (void *func, int narg, ...)
{
    int i;                          /* index */
    va_list argp;                   /* argument pointer */
    struct {                        /* function argument */
        int arg[16];                /* function argument list */
    } arglist;

    if (narg > 16 || narg < 0)      /* maximum of 16 user parameters */
        return APIS_EPARAM;

    va_start(argp, narg);          /* initialize parameter pointer */
    for (i = 0; i < narg; i++)
        arglist.arg[i] =
            (int)va_arg(argp, int); /* get next parameter */

    va_end(argp);                   /* end variable arguments */

    /* execute subroutine in user trap handler
     */
    if (tcall(trapnr, USRFUNC, func, arglist) == -1)
        return APIS_EINVMOD;

    return APIS_NOERR;
}


/*
 * Function:    _i4000os9_delay
 *
 * Description: Suspend process for a requested delay
 *              The delay is provided in msecs but
 *              the minimum delay time is one
 *              system tick (mostly 10msecs).
 *
 * Parameters:  UINT32 dtime
 *                  delay time in [msec]
 *
 * Returns:     APIS error code
 */
int _i4000os9_delay (UINT32 dtime)
{
    static u_int16 tcksec = 0;  /* system ticks per second */
    u_int32 ticks_to_sleep;     /* number of ticks to sleep */

    /* get number of ticks per second
     * from system globals (only the first time)
     */
    if (tcksec == 0)
        if (_os_getsys(0x28, 2, (glob_buff *)&tcksec) != 0)
            return APIS_EGOS;

    /* ticks to sleep := requested delay * ticks/sec (minimum of 1)
     */
    ticks_to_sleep = (dtime * tcksec)/1000+0.5;

    while (ticks_to_sleep != 0)         /* goto sleep */
        _os9_sleep(&ticks_to_sleep);    /* for the requested delay */

    return APIS_NOERR;
}

/*
 * Function:     _i4000os9_getversion
 *
 * Description:  Get version of source and trap handler
 *
 * Parameter:    char *verstr
 *                  Pointer to version string
 *
 * Returns:      Nothing
 *
 */
void _i4000os9_getversion(char *verstr)
{
    sprintf(verstr,"LIB:%d_TRAP:%d", LIB_VERSION, traped);
}

/*
 * Function:     _i4000os9_cmpversion
 *
 * Description:  Compare version of apis_i4000os9.h with current versions
 *               of source and trap handler.
 *
 * Parameter:    char *apis_version
 *                  Define from apis_i4000os9.h
 *
 * Returns:      APIS_NOERR, versions are correct
 *               APIS_EINVVER, invalid version
 *
 */
int _i4000os9_cmpversion(char *apis_version)
{
    char verstr[40];
    int libver, trapver;
    int libcur, trapcur;

    _i4000os9_getversion(verstr);
    sscanf(apis_version, "LIB:%d_TRAP:%d", &libver, &trapver);
    sscanf(verstr, "LIB:%d_TRAP:%d", &libcur, &trapcur);

    if ((libcur >= libver) && (trapcur >= trapver))
        return APIS_NOERR;
    else
        return APIS_EINVVER;
}
```

```
/* Local Functions */

/*
 * sighand  -   signal handler
 *
 * Description: This routine is called when a signal is intercepted.
 *              If the signal is generated by our interrupt service
 *              sigcnt is incremented.
 *
 * Inputs:      int sig
 *                  signal number
 *
 * Returns:     n/a
 */
static void sighand(int sig)
{
    if (sig == IRQSIG)                      /* if signal caused by APIS */
        sigcnt = 1;                         /* set signal count to one */
    _os_rte();                              /* exit */
}

/*
 * irqh  -   main interrupt handler for OS-9
 *
 * Description: This interrupt handler is provided for use with
 *              OS-9 without an OS9-driver.
 *              The main interrupt handler will call a user defined
 *              interrupt service routine (user_irqh()). A pointer
 *              to the user interrupt service routine is taken from
 *              the path's info structure. A pointer to the info
 *              structure belonging to the path that is responsable
 *              for the interrupt is taken from the CPU-register (a3)
 *              and passed to the user interrupt service routine
 *              via (d0).
 *
 *              The user interrupt service routine must check
 *              if the interrupt is valid, if not the routine must
 *              return a non-zero value and the interrupt service
 *              routine is terminated with the carry set.
 *              If the interrupt was valid a signal (IRQSIG)
 *              is sent to the process with the ID as programmed in
 *              'proc_id' and the routine is terminated with the carry
 *              cleared.
 *
 * Inputs:      none
 *
 * Returns:     n/a
 */
_asm("**********                                                        ");
_asm("* irqh: main interrupt handler                                    ");
_asm("*                                                                 ");
_asm("* Passed:  (a2) = Static Storage addr                             ");
_asm("*          (a3) = handle                                          ");
_asm("*          (a6) = system global data ptr                          ");
_asm("*                                                                 ");
_asm("* Returns:  (cc) = carry set if false interrupt, else clear       ");
_asm("*                                                                 ");
_asm("* Destroys:  May only destroy D0, D1, A0, A2, A3 and A6.  Any     ");
_asm("*            other registers used MUST be preserved.              ");
_asm("*                                                                 ");
_asm("UIRQH     equ     4               ; PATH_INFO offset to uirqh     ");
_asm("VPTR      equ     8               ; PATH_INFO offset to vptr      ");
_asm("IRQSIG    equ     400             ; IRQSIG                        ");
_asm("irqh:     move.l  UIRQH(a3),a0    ; get pointer to user irqh      ");
_asm("          move.l  a2,a6           ; set-up global storage for C   ");
_asm("          move.l  a3,d0           ; pass handle as parameter      ");
_asm("          move.l  VPTR(a3),d1     ; pass user variable            ");
_asm("          jsr     (a0)            ; jump to user irqh             ");
_asm("          tst.l   d0              ; if result != 0                ");
_asm("          bne     not_ours        ; then interrupt not ours       ");
_asm("          move.l  proc_id(a6),d0  ; else send signal #IRQSIG      ");
_asm("          move.w  #IRQSIG,d1      ; to the process with id        ");
_asm("          OS9     F$Send          ; proc_id                       ");
_asm("          moveq   #0,d1           ; clear error code              ");
_asm("          rts                     ; return with not error         ");
_asm("not_ours  ori.b   #Carry,ccr      ; interrupt not ours: set carry ");
_asm("          rts                     ; and return                    ");

/*
 * tlink  -   trap link
 *
 * Description: Link to traphandler
 *
 * Parameters:  int trapnum
 *                  trap number
 *              char *trapname
 *                  pointer to a string containing the
 *                  name of the traphandler
 *              UINT16 *edition
 *                  pointer to edition of trap handler
 *
 * Returns:     0 or -1
 *
 * tlink(int trapnum, char* trapname, UINT16 *edition)
 */
_asm("tlink:    link    a5,#0               ; link a5                   ");
_asm("          movem.l a0-a2,-(a7)         ; save registers            ");
_asm("          movea.l d1,a0               ; get pointer to trapname   ");
_asm("          moveq   #0,d1               ; no memory overide         ");
_asm("          OS9     F$TLink             ; link trap handler         ");
_asm("          bcc.s   tlink99             ; if carry not set return   ");
_asm("          move.l  d1,errno(a6)        ; else set global errno     ");
_asm("          moveq   #-1,d0              ; and return -1             ");
_asm("tlink99   movea.l 8(a5),a0            ; get param3                ");
_asm("          move.w  $16(a2),(a0)        ; copy edition to param3    ");
_asm("          movem.l (a7)+,a0-a2         ; restore registers         ");
_asm("          unlk    a5                  ; unlink a5                 ");
_asm("          rts                         ; return                    ");

/*
 * tcall  -     trap call
 *
 * Description: Call traphandler
 *
 * Parameters:  int trapnum
 *                  trap number
 *              short func
 *                  function of traphandler
 *              int p1,p2,p3,p4,p5
 *                  trap function parameters
 *
 * Returns:     0 or -1
 *
 * tcall(int trapnum, short func, p1, p2, p2, p3, p4, p5)
 */
_asm("TRAP      equu     $4e40               ; TRAP instruction format   ");
_asm("RTS       equ     $4e75               ; RTS instruction format    ");
_asm("          vsect                       ; storage for trap call sub-");
_asm("trapinst  ds.w    2                   ; routine:                  ");
_asm("rtsinst   ds.w    1                   ;   TRAP #vector            ");
_asm("          ends                        ;   RTS                     ");
_asm("tcall:    link    a5,#0               ; link a5                   ");
_asm("          tst.l   d0                  ; if trapnr == 0            ");
_asm("          beq.s   paramerr            ; then error                ");
_asm("          cmp.l   #15,d0              ; if trapnr > 15            ");
_asm("          bhi.s   paramerr            ; then error                ");
_asm("          add.w   #TRAP,d0            ; else compose TRAP instr.  ");
_asm("          movem.w d0-d1,trapinst(a6)  ; patch TRAP instruction    ");
_asm("          move.w  #RTS,rtsinst(a6)    ; patch RTS instruction     ");
_asm("          moveq.l #0,d0               ; clear instruction and     ");
_asm("          OS9     F$CCtl              ; data cache                ");
_asm("          jsr     trapinst(a6)        ; call trap instruction     ");
_asm("          bcc.s   tcall99             ; if carry not set return   ");
_asm("          move.l  d1,errno(a6)        ; else set global errno     ");
_asm("          bra.s   tcallerr            ; and exit with with -1     ");
_asm("paramerr  move.l  #E$Param,errno(a6)  ; set parameter error       ");
_asm("tcallerr  moveq   #-1,d0              ; exit with -1              ");
_asm("tcall99   unlk    a5                  ; unlink a5                 ");
_asm("          rts                         ; return                    ");
```

5.4. Example of APIS Based Application Software

This section shows a demo application which is based on APIS. The ANSI-C source file that contains the program's main entry is listed, source files that do not add to the clarity of APIS example are omitted.
The demo application is programmed around the M321. The M321 is a stepper-motor controller M-module from AcQ.

```
/*
 *  file:     m321apis.c
 *  revision: 1.1
 *  date:     12/01/00
 *  author:   SP
 * ----------------------------------------------------------------------
 *
 * M321/APIS demo
 *
 * ----------------------------------------------------------------------
 * Copyright 1999 by AcQuisition Technology B.V. (c)
 * All Rights Reserved
 * Reproduced Under License
 *
 * This source code is the proprietary confidential property of
 * AcQuisition Technology B.V., and is provided to the licensee
 * for documentation and educational purposes only. Reproduction,
 * publication, or any form of distribution to any party other than
 * the licensee is strictly prohibited.
 * ----------------------------------------------------------------------
 * Edition History
 *
 * #    date        Comments                                        by
 * --_  --------    ----------------------------------------------  ---
 * 0.0  03-06-99    derived from m321irq.c                          sp
 * 0.1  28-06-99    swap routines removed                           sp
 *                  default interrupt vector and level used
 * 1.0  20-07-99    First release
 *                  call to apis_criticalcode changed               sp
 * 1.1  12-01-00    call to apis_irqinstall changed                 jg
 *
 */
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include "../../../APIS/SOFTWARE/COMMON/DEFS/apis.h"


sd
/* M321 Definitions
 *  Definitions irrelevant for this demo are omitted
 */

/* Host register definitions (offsets from module's base address)
 */
#define PAGEREG     0x80    /* dual ported memory page register */
#define CTRLREG     0x82    /* control register */
#define CMDPAGE     4       /* command page */

/* Hardware register bit definitions
 */
```

```
#define HIRQ        0x0001  /* host interrupt request bit */
#define RESET       0x0004  /* local reset bit */
#define POLL        0x0008  /* poll bit, set by local cpu cleared by host
*/

/* M321 command interface offsets
 */
#define C_CMD       0x0000  /* command */
#define C_RES       0x0002  /* execution result */
#define C_PRF       0x0004  /* parameter field */
#define C_IRQ       0x0044  /* interrupt request status */

/* M321 command definitions
 */
#define VERSION     0x0001  /* returns version information */
#define MSKI        0x0020  /* mask/un-mask interrupts */
#define IACK        0x0030  /* acknowledge interrupts */
#define PROF_REL_A  0x0400  /* relative trapezoidal movement motor A */
#define SET_BPR_A   0x0800  /* set relative breakpoint motor A */

/* M321 command parameter identifiers
 */
#define FWVER       0       /* firmware version */
#define IMASK       0       /* interrupt mask */
#define RISE_H      0       /* rise frequency high */
#define RISE_L      1       /* rise frequency low */
#define DRIVE_H     2       /* drive frequency high */
#define DRIVE_L     3       /* drive frequency low */
#define ACC_H       4       /* acceleration high */
#define ACC_L       5       /* acceleration low */
#define POS_H       6       /* position high */
#define POS_L       7       /* position low */
#define BRKPT_H     0       /* breakpoint high */
#define BRKPT_L     1       /* breakpoint low */

/*  Interrupt status bits
 */
#define IRQ_BRKPT_A 0x0002  /* breakpoint motor A */
#define IRQ_TRAJ_A  0x0008  /* trajectory complete motor A */

/* Externals
 */
extern const unsigned char ms21firm[];  /* firmware image */
extern const unsigned ms21firm_sz;      /* firmware size in bytes */

/* Globals
 */
volatile UINT16 m321istat;      /* storage for m321 IRQ status */
volatile int term = 0;          /* program termination flag */

/* Forward declarations
 */
int m321_boot(APIS_HANDLE , UINT8*, unsigned);      /* boot firmware */
int m321_poll(APIS_HANDLE);                         /* poll module */
int m321_cmd(APIS_HANDLE, UINT16, void*, int ,int); /* send command */
int m321_irqh(APIS_HANDLE, void *);                 /* m321 IRQ handler */
void usage(char *);                                 /* display usage */
void setstat(UINT16 *, int);                        /* set status */

/*
 * Function:    main
 *
 * Description: M321/APIS Demo program entry
 *
 * Parameters:  int argc
 *                  number of program arguments
 *              char *argv[]
 *                  pointer to program argument list
 *
 * Returns:     0
 */
int main (int argc, char *argv[])
{
    APIS_HANDLE handle;     /* APIS Handle */
    UINT16 *args;           /* M321 argument list */
    UINT32 pathid = 0;      /* M321 base address */
    int i, j;               /* General idici */
    int result;             /* Command result */
    int repeat = 0;         /* Repeat flag */

    printf("\nM321/APIS Demo\n");
    printf("by AcQuisition Technology B.V. 1999\n\n");

    for (i = 1; i < argc; i++) {
        if (argv[i][0] == '-') {
            for (j = 1; argv[i][j]; j++) {
                switch (tolower(argv[i][j])) {
                    case 'b':
                        if (argv[i][++j] == '=')
                            j++;
                        sscanf(argv[i]+j, "%lx", &pathid);
                        while(argv[i][j])
                            j++;
                        j--;
                        break;
                    case 'r':
                        repeat = 1;
                        break;
                    default:
                        usage(argv[0]);
                        exit(0);
                }
            }
        }
    }

    /*
     * Allocate memory for the M321 command argument list
     */
    if ((args = (UINT16 *)malloc(32)) == 0) {
        printf("Not enough memory\n");
        exit(0);
    }

    /*
     * Open a hardware path to the M321
     */
    if ((result = apis_open(pathid, &handle, 0, 0)) != 0) {
        printf("Could not open path: 0x%04x\n", result);
        exit(0);
    }
```

```
/*
 * boot the M321
 */
printf("Booting..."); fflush(0);
if (m321_boot(handle, (UINT8 *)&ms21firm[0], ms21firm_sz) != 0) {
    printf("failed\n\n");
    apis_close(handle);
    exit(0);
}
printf("done\n\n");

/*
 * Get firmware version
 */
if ((result = m321_cmd(handle, VERSION, args, 0, 1)) != 0) {
    printf("VERSION command failed: 0x%04x\n", result);
    apis_close(handle);
    exit(0);
}
printf("Running firmware version: %d\n", args[FWVER]);


/*
 * Set up interrupts with default interrupt vector, level
 * and mode.
 */
if ((result = apis_irqinstall(handle, (void *)m321_irqh, 0,0,0,NULL))
    != 0) {
    printf("Irqinstall error %d\n", result);
    apis_close(handle);
    exit (0);
}

/*
 * Unmask interrupts
 */
args[IMASK] = (IRQ_TRAJ_A|IRQ_BRKPT_A);
if ((result = m321_cmd(handle, MSKI, args, 1, 0)) != 0) {
    printf("MSKI command failed: 0x%04x\n", result);
    apis_irqremove(handle);
    apis_close(handle);
    exit(0);
}

/*
 * Clear interrupt status bits in m321istat
 * This is done via the apis_criticalcode function.
 * The routine setstat is executed with all interrupts disabled.
 */
apis_criticalcode((void*)setstat, 2, &m321istat,
                                    (IRQ_TRAJ_A|IRQ_BRKPT_A));

do {
    printf("Set Relative Breakpoint to +1000\n");
    args[BRKPT_H] = 0;
    args[BRKPT_L] = 0x3e8;
    if ((result = m321_cmd(handle, SET_BPR_A, args, 2, 0)) != 0) {
        printf("SET_BPR_A command failed: 0x%04x\n", result);
        term = 1;
    }
```

```
        printf("Moving +5000 steps\n\n");
        args[RISE_H] = 0;           /* rise frequency = 300 Hz */
        args[RISE_L] = 0x12c;
        args[DRIVE_H] = 0;          /* drive frequency = 1000 Hz */
        args[DRIVE_L] = 0x3e8;
        args[ACC_H] = 0;            /* acceleration = 10000 Hz/sec */
        args[ACC_L] = 0x2710;
        args[POS_H] = 0;            /* end position = 5000 steps */
        args[POS_L] = 0x1388;
        if ((result = m321_cmd(handle, PROF_REL_A, args, 8, 0)) != 0) {
            printf("PROF_REL_A command failed: 0x%04x\n", result);
            term = 1;
        }


        do {
            if (apis_waitforirq() != 0)
                term = 1;

            /* Verify interrupt flags
             * Unmasked interrupt sources:
             *      IRQ_BRKPT_A
             *      IRQ_TRAJ_A
             */
            if (m321istat & IRQ_BRKPT_A) {
                printf("Breakpoint detected\n\n");
                apis_criticalcode((void*)setstat, 2,
                                            &m321istat, IRQ_BRKPT_A);
            }

        } while (!(m321istat & IRQ_TRAJ_A) && !term);

        if (m321istat & IRQ_TRAJ_A) {
            printf("Traject A completed\n\n");
            apis_criticalcode((void*)setstat, 2,
                                            &m321istat, IRQ_TRAJ_A);
        }
    } while (repeat && !term);

    /*
     * Mask all interrupts
     */
    args[IMASK] = 0;
    if ((result = m321_cmd(handle, MSKI, args, 1, 0)) != 0) {
        printf("MSKI command failed: 0x%04x\n", result);
    }
    apis_irqremove(handle);     /* remove the interrupt handle */
    apis_close(handle);         /* and close path */
    return 0;

}

/*
 * Function:    m321_poll
 *
 * Description: Wait until an action is handled.  An action can be
 *              a command being executed or an indication that
 *              the module is ready after a reset.
 *              The routine reads the control register of the M321
 *              and checks the polling bit (if set then action the
 *              action is done or the module is ready).
 *              A timeout mechanism of 10 seconds is implemented.
 *
 *
 * Parameters:  APIS_HANDLE handle
 *                  hardware path handle
 *
 * Returns:     0 or -1 if timeout occured
 */
int m321_poll (APIS_HANDLE handle)
{
    UINT16 data;
    int i;

    for (i = 0; i < 1000; i++)
    {
        apis_read(handle, sizeof(UINT16), CTRLREG, &data);
        if (data & POLL)
            break;
        apis_delay(10);
    }
    if (data & POLL)
        return 0;
    else
        return -1;
}

/*
 * Funciton:    m321_boot
 *
 *
 * Description: Put module in reset, download boot image to shared RAM
 *              and remove reset. Wait until the polling bit in the
 *              control register is set by the local CPU, which indicates
 *              that the firmware is up and running.
 *
 *              CAUTION:    for this function the jumper configuration
 *                          must be set to booting from RAM.
 *
 * Parameters:  APIS_HANDLE handle
 *                  hardware path handle
 *              UINT8 *pBootImage
 *                  pointer to the M321 boot code image
 *              unsigned size
 *                  size of the M321 boot code image
 *
 * Returns:     OKE, -1 if there is no response from the firmware
 *              or INVDEV if the device number is invalid
 *
 */
int m321_boot (APIS_HANDLE handle, UINT8 *pBootImage, unsigned size)
{
    unsigned i, j;
    UINT8 *ptr;
    UINT16 data;

    /* Put module in reset
     */
    apis_write(handle, sizeof(UINT16), CTRLREG, (UINT16)(RESET));

    ptr = pBootImage;       /* get pointer to boot image */
    size = (size+128)/128;  /* convert size in bytes to size in pages */
    /* Download firmware image to dual ported memory
     */
    for (i = 0; i < size; i++)
    {
        apis_write(handle, sizeof(UINT16), PAGEREG, i);
        for (j = 0; j < 64; j++)
        {
            data = *ptr++ << 8 ;
            data |= *ptr++;
            apis_write(handle, sizeof(UINT16), (2*j), data);
        }
    }

    /* Release module reset
     */
    apis_write(handle, sizeof(UINT16), CTRLREG, (UINT16)0);
    return (m321_poll(handle));
}

/*
 * Function:    m321_cmd
 *
 *
 * Description: Copy command parameters to the parameter field in
 *              shared ram of the module. Execute command and wait
 *              for the command to be executed.
 *              Copy the result parameters and return with the
 *              result code obtained from the firmware.
 *
 * Parameters:  APIS_HANDLE handle
 *                  hardware path handle
 *              UINT16 command
 *                  firmware command
 *              void *pArgList
 *                  pointer to the command parameter list
 *              int ni
 *                  number of input command parameters
 *              int no
 *                  number of output command parameters
 *
 *
 * Returns:     the result code or -1 if the module does not respond
 *
 */
int m321_cmd (APIS_HANDLE handle, UINT16 command, void *pArgList,
                int ni, int no)
{
    UINT16 result;

    apis_write(handle, sizeof(UINT16), PAGEREG, CMDPAGE);
    apis_writeblock(handle, sizeof(UINT16), C_PRF, ni, pArgList);

    apis_write(handle, sizeof(UINT16), CTRLREG, POLL);
    apis_write(handle, sizeof(UINT16), C_CMD, command);

    if (m321_poll(handle) != 0)
        return -1;              /* timeout */

    apis_readblock(handle, sizeof(UINT16), C_PRF, no, pArgList);

    apis_read(handle, sizeof(UINT16), C_RES, &result);
    return ((int)result);       /* return result code */
}


/*
 * Function:    m321_irqh
 *
 * Description: The interrupt handler will clear the interrupt of
 *              the m321. The interrupt status is saved in 'm321istat'
 *              and pending interrupts are cleared with the IACK
 *              command.
 *
 * Parameters:  none
 *
 * Returns:     0 or -1 if not ours
 */
int m321_irqh (APIS_HANDLE handle, void *vptr)
{
    UINT16 data;
    UINT16 istat;

    apis_read(handle, sizeof(UINT16), CTRLREG, &data);

    if (data & HIRQ) {          /* is interrupt ours ? */

        apis_write(handle, sizeof(UINT16), CTRLREG, HIRQ);
        apis_read(handle, sizeof(UINT16), C_IRQ, (void *)&istat);

        do
            apis_read(handle, sizeof(UINT16), C_CMD, &data);
        while (data);
        apis_write(handle, sizeof(UINT16), C_PRF, istat);
        apis_write(handle, sizeof(UINT16), C_CMD, IACK);

        m321istat |= istat;     /* set global interrupt status */

        do
            apis_read(handle, sizeof(UINT16), C_CMD, &data);
        while (data);
        return 0;
    }
    return -1;
}

/*
 * Function:    setstat
 *
 * Description: Clear bits in the passed status word, this routine
 *              is provided for execution via the critical code
 *              function of APIS, to ensure the integrity of
 *              the supplied status word.
 *
 * Parameters:  UINT16 *status
 *                  pointer to status word
 *              int mask
 *                  bits to clear
 *
 * Returns:     nothing
 */
void setstat(UINT16 *status, int mask)
{
    *status &= ~mask;
}


/*
 * Function:     usage
 *
 * Description: The program usage is displayed and the program
 *               is terminated
 *
 * Parameters:   pointer to the program name
 *
 * Returns:      nothing
 */
void usage(char *pname)
{
    printf("Syntax: %s [<opts>]\n", pname);
    printf("Options:\n");
    printf("  -b=<base>     module base address in hex\n");
    printf("  -r            repeat\n\n");
}
```