EAPI

PICMG EAPI

Revision 1.0
August 8, 2010

Embedded **A**pplication **P**rogramming **I**nterface

Open Modular Computing Specifications

Table of Contents

API Function Prototypes

Revision History

Revision	Date	Action
1.0	August 8, 2010	Revision 1.0

1 Introduction

1.1 Name and Logo Usage

The PCI Industrial Computer Manufacturers Group policies regarding the use of its logos and trademarks are as follows:

Permission to use the PICMG® organization logo is automatically granted to designated members only as stipulated on the most recent Membership Privileges document (available at www.picmg.org) during the period of time for which their membership dues are paid. Nonmembers may not use the PICMG® organization logo.

The PICMG® organization logo must be printed in black or color as shown in the files available for download from the member's side of the Web site. Logos with or without the "Open Modular Computing Specifications" banner can be used. Nothing may be added or deleted from the PICMG® logo.

The use of the COM Express logo is a privilege granted by the PICMG® organization to companies who have purchased the relevant specifications (or acquired them as a member benefit), and that believe their products comply with these specifications. Manufacturers' distributors and sales representatives may use the COM Express logo in promoting products sold under the name of the manufacturer. Use of the logos by either members or non-members implies such compliance. Only PICMG Executive and Associate members may use the PICMG® logo. PICMG® may revoke permission to use logos if they are misused. The COM Express logo can be found on the PICMG web site, www.picmg.org.

The COM Express® logo must be used exactly as shown in the files available for download from the PICMG® Web site. The aspect ratios of the logos must be maintained, but the sizes may be varied. Nothing may be added to or deleted from the COM Express® logo.

The PICMG® name and logo and the COM Express name and logo are registered trademarks of PICMG®. Registered trademarks must be followed by the ® symbol, and the following statement must appear in all published literature and advertising material in which the logo appears:

PICMG, the COM Express name and logo and the PICMG logo are registered trademarks of the PCI Industrial Computers Manufacturers Group.

1.2 Intellectual Property

The Consortium draws attention to the fact that it is claimed that compliance with this specification may involve the use of a patent claim(s) ("IPR"). The Consortium takes no position concerning the evidence, validity or scope of this IPR.

The holder of this IPR has assured the Consortium that it is willing to license or sublicense all such IPR to those licensees (Members and non-Members alike) desiring to implement this specification. The statement of the holder of this IPR to such effect has been filed with the Consortium.

Attention is also drawn to the possibility that some of the elements of this specification ***may*** be the subject of IPR other than those identified below. The Consortium ***shall*** not be responsible for identifying any or all such IPR.

No representation is made as to the availability of any license rights for use of any IPR inherent in this specification for any purpose other than to implement this specification.

This specification conforms to the current PICMG® Intellectual Property Rights Policy and the Policies and Procedures for Specification Development and does not contain any known intellectual property that is not available for licensing under Reasonable and Non-discriminatory terms. In the course of Membership Review the following disclosures were made:

1.2.1 Necessary Claims (referring to mandatory or recommended features)

No disclosures in this category were made during subcommittee review.

1.2.2 Unnecessary Claims (referring to optional features or non-normative elements)

No disclosures in this category were made during subcommittee review.

1.2.3 Third Party Disclosures

(Note that third party IPR submissions do not contain any claim of willingness to license the IPR.)

No disclosures in this category were made during subcommittee review.

Refer to PICMG® IPR Policies and Procedures and the company owner of the patent for terms and conditions of usage.

PICMG® makes no judgment as to the validity of these claims or the licensing terms offered by the claimants.

THIS SPECIFICATION IS BEING OFFERED WITHOUT ANY WARRANTY WHATSOEVER, AND IN PARTICULAR, ANY WARRANTY OF NON-INFRINGEMENT IS EXPRESSLY DISCLAIMED. ANY USE OF THIS SPECIFICATION SHALL BE MADE ENTIRELY AT THE IMPLEMENTER'S OWN RISK, AND NEITHER THE CONSORTIUM, NOR ANY OF ITS MEMBERS OR SUBMITTERS, SHALL HAVE ANY LIABILITY WHATSOEVER TO ANY IMPLEMENTER OR THIRD PARTY FOR ANY DAMAGES OF ANY NATURE WHATSOEVER, DIRECTLY OR INDIRECTLY, ARISING FROM THE USE OF THIS SPECIFICATION.

Compliance with this specification does not absolve manufacturers of COM Express® equipment from the requirements of safety and regulatory agencies (UL, CSA, FCC, IEC, etc.).

PICMG® and the COM Express® logos are trademarks of the PCI Industrial Computer Manufacturers Group.

All other brand or product names may be trademarks or registered trademarks of their respective holder.

1.3 Copyright Notice

PICMG has made every attempt to ensure that the information in this document is accurate yet the information contained within is supplied “as-is”.

1.4 Trademarks

Intel and Pentium are registered trademarks of Intel Corporation. ExpressCard is a registered trademark of Personal Computer Memory Card International Association (PCMCIA). PCI Express is a registered trademark of Peripheral Component Interconnect Special Interest Group (PCI-SIG). COM Express is a registered trademark of PCI Industrial Computer Manufacturers Group (PICMG). I2C is a registered trademark of NXP Semiconductors. CompactFlash is a registered trademark of CompactFlash Association. Winbond is a registered trademark of Winbond Electronics Corp. AVR is a registered trademark of Atmel Corporation. Microsoft®, Windows®, Windows NT®, Windows CE and Windows XP® are registered trademarks of Microsoft Corporation. VxWorks is a registered trademark of WindRiver. All product names and logos are property of their owners.

1.5 Special Word Usage

Mandatory features are indicated by the use of the word "***shall***."

Recommended features are indicated by the use of the word "***should***."

Optional features are indicated by the use of the word "***may***."

1.6 Acronyms / Definitions

Term	Definition
a.out	a.out is a file format used in older versions of Unix-like computer operating systems for executable, object code, and, in later systems, shared libraries. The name stands for assembler output.
AC '97	Audio CODEC (Coder-Decoder)
ACPI	Advanced Configuration Power Interface – standard to implement power saving modes in PC-AT systems
API	Application Programming Interface
Atomic Error Checking	This Is used here to refer to the mechanism of validating all arguments before modifications are carried out. **Examples:** EapiI2CWriteReadRaw: Validation of arguments for Read operation should be carried out before write operation. Furthermore neither write nor read operation should be carried out if an error is detected in the arguments for either. GPIO: Bitmask and Direction registers should be validated before modifying GPIO state.
BBS	BIOS Boot Specification
BIOS	Basic Input Output System – firmware in PC-AT system that is used to initialize system components before handing control over to the operating system.
Boot Counter	A lifetime system count of the number of times the EFI/BIOS's Post is completed. Post completion is defined as being just prior to processor control being passed to the first boot vector. Should no valid boot vectors be present it is also considered a valid boot.
Boot Entry Vector	Valid int19h Interrupt Handler Vector
Boot Vector	Short for Boot Entry Vector
Carrier Board	An application specific circuit board that accepts a COM Express® Module.
COFF	Common Object File Format
COM Express	PICMG Definition for Computer-On-Modules
CPU	Central Processing Unit.
DDC	Display Data Control – VESA (Video Electronics Standards Association) standard to allow identification of the capabilities of a VGA monitor
DisplayPort	DisplayPort is a digital display interface standard put forth by the Video Electronics Standards Association (VESA). It defines a new license free, royalty free, digital audio/video interconnect, intended to be used primarily between a computer and its display monitor.
DVI	Digital Visual Interface - a Digital Display Working Group (DDWG) standard that defines a standard video interface supporting both digital and analog video signals. The digital signals use TMDS.
EAPI	Embedded API
EApiDK	EAPI Development Kit. Open source project with a test bench and sample code to encourage EAPI development. http://eapidk.sourceforge.net/.
EDB	Eccentric Device Behavior

Term	Definition
eDP	Embedded Display Port
EEPROM	Electrically Erasable Programmable Read-Only Memory
EFI	Extensible Firmware Interface(Next Generation BIOS)
EFI BIOS	Used to explicitly distinguish EFI and Legacy BIOS.
EFP	External Flat Panel
ELF	Executable and Linkable Format
Gb	Gigabit
GBE	Gigabit Ethernet
GPI	General Purpose Input
GPIO	General Purpose Input Output
GPO	General Purpose Output
HDA	Intel High Definition Audio (HD Audio) refers to the specification released by Intel in 2004 for delivering high definition audio that is capable of playing back more channels at higher quality than AC97.
HDMI	High Definition Multimedia Interface
I^2C	Inter Integrated Circuit – 2 wire (clock and data) signaling scheme allowing communication between integrated circuits, primarily used to read and load register values.
IDE	Integrated Device Electronics – parallel interface for hard disk drives – also known as PATA
Int19h	Bootstrap Loader Service. This is the interrupt that is called at end of post to boot the OS.
Interrupt Handler Vector	An interrupt Handler vector is the memory address of an interrupt handler, or an index into an array called an interrupt vector table or dispatch table. Interrupt vector tables contain the memory addresses of interrupt handlers. When an interrupt is generated, the processor saves its execution state via a context switch, and begins execution of the interrupt handler at the interrupt vector.
Interrupt Vector	Short for Interrupt Handler Vector
LAN	Local Area Network
Legacy BIOS	Used to explicitly distinguish EFI and Legacy BIOS.
Legacy Device	Relics from the PC-AT computer that are not in use in contemporary PC systems: primarily the ISA bus, UART-based serial ports, parallel printer ports, PS-2 keyboards, and mice. Definitions vary as to what constitutes a legacy device. Some definitions include IDE as a legacy device.
LFP	Local Flat Panel
LPC	Low Pin-Count Interface: a low speed interface used for peripheral circuits such as Super I/O controllers, which typically combine legacy-device support into a single IC.
LS	Least Significant
LSB	Least Significant Byte
LVDS	Low Voltage Differential Signaling – widely used as a physical interface for TFT flat panels. LVDS can be used for many high-speed signaling applications. In this document, it refers only to TFT flat-panel applications.
MS	Most Significant
MSB	Most Significant Byte
NA	Not Available
NC	No Connect
OEM	Original Equipment Manufacturer
OS	Operating System
PATA	Parallel AT Attachment – parallel interface standard for hard-disk drives – also known as IDE, AT Attachment, and as ATA
PC-AT	“Personal Computer – Advanced Technology” – an IBM trademark term used to refer to Intel x86 based personal computers in the 1990s
PCB	Printed Circuit Board
PCI	Peripheral Component Interface
PCI Express PCIE	Peripheral Component Interface Express – next-generation high speed Serialized I/O bus

Term	Definition
PE	Portable Executable
PE/COFF	Another name for PE.
PEG	PCI Express Graphics
PHY	Ethernet controller physical layer device
PNP	Plug and Play
PNPID	Microsoft Plug-And-Play ID (PNP ID). This ID can be registered at the Microsoft web page (http://www.microsoft.com/whdc/system/pnppwr/pnp/pnpid.mspx) free of charge. The PNP ID Format is XXX where 'A'<=X<='Z'
Post	Power On Self Test, For the purpose of this document this refers to period of time After CPU exits reset up until OS Boot Process Begins. Standard definition is the Period of Time after Boot block if present up until OS Boot Process Begins.
PS2 **PS2 Keyboard** **PS2 Mouse**	"Personal System 2" - an IBM trademark term used to refer to Intel x86 based personal computers in the 1990s. The term survives as a reference to the style of mouse and keyboard interface that were introduced with the PS2 system.
Reset Counter	A lifetime System count of the number of times the system reset goes inactive. Adequate debounce protection is presumed.
ROM	Read Only Memory – a legacy term – often the device referred to as a ROM can actually be written to, in a special mode. Such writable ROMs are sometimes called Flash ROMs. BIOS is stored in ROM or Flash ROM.
RTC	Real Time Clock – battery backed circuit in PC-AT systems that keeps system time and date as well as certain system setup parameters
Run-time	Used to refer to the time period after post and before a CPU reset.
Run-time Services	Used to refer to Services, Software interfaces left active after Post. Examples Are Int13h Disk access interface and PNP.
Running Time Meter	A lifetime system count of the number of whole minutes that the system reset has been inactive. The readout values might differ depending on the on hardware depending internal resolution. I.E. 200 boot cycles each lasting 59 seconds might return a running time meter value of 0 min or 197 min.
S0, S1, S2, S3, S4, S5	System states describing the power and activity level S0 Full power, all devices powered S1 S2 S3 Suspend to RAM System context stored in RAM; RAM is in standby S4 Suspend to Disk System context stored on disk S5 Soft Off Main power rail off, only standby power rail present
SAS	Serial Attached SCSI – high speed serial version of SCSI
SATA	Serial AT Attachment: serial-interface standard for hard disks
SCSI	Small Computer System Interface – an interface standard for high end disk drives and other computer peripherals
SDVO	Serialized Digital Video Output – Intel defined format for digital video output that can be used with Carrier Board conversion ICs to create parallel, TMDS, and LVDS flat-panel formats as well as NTSC and PAL TV outputs. SDVO is a proprietary Intel technology introduced with their 9xx-series of motherboard chipsets. The specification is not public available.
SM Bus	System Management Bus
SPI	Serial Peripheral Interface
Super I/O	An integrated circuit, typically interfaced via the LPC bus that provides legacy PC I/O functions including PS2 keyboard and mouse ports, serial and parallel port(s) and a floppy interface.
TFT	Thin Film Transistor – refers to technology used in active matrix flat-panel displays, in which there is one thin film transistor per display pixel.
TMDS	Transition Minimized Differential Signaling - a digital signaling protocol between the graphics subsystem and display. TMDS is used for the DVI digital signals.
TPM	Trusted Platform Module, chip to enhance the security features of a computer system.
USB	Universal Serial Bus
VGA	Video Graphics Adapter – PC-AT graphics adapter standard defined by IBM.

Term	Definition
Watchdog	A hardware or software count-down timer that must be repeatedly reset or triggered at regular intervals by an operating program to prevent time-out. If the timer is not reset in time (meaning the operating program has stalled or has failed in some way), the timer times-out and alerts the system and/or operator of the failure.
Watchdog Trigger	The process or technique or resting the Watchdog counter, preventing a Watchdog timeout event.
WDT	Watch Dog Timer.

1.7 Applicable Documents and Standards

The following publications are used in conjunction with this standard. When any of the referenced specifications are superseded by an approved revision, that revision ***shall*** apply. All documents ***may*** be obtained from their respective organizations.

- SDVO Specification. Intel confidential. http://www.intel.com
- Advanced Configuration and Power Interface Specification Revision 4.0, June 16, 2009 Copyright © 1996-2003 Compaq Computer Corporation, Intel Corporation, Microsoft Corporation, Phoenix Technologies Ltd., Toshiba Corporation. All rights reserved. http://www.acpi.info/
- ANSI/TIA/EIA-644-A-2001: Electrical Characteristics of Low Voltage Differential Signaling (LVDS) Interface Circuits, January 1, 2001. http://www.ansi.org/
- ANSI INCITS 361-2002: AT Attachment with Packet Interface - 6 (ATA/ATAPI-6), November 1, 2002. http://www.ansi.org/
- ANSI INCITS 376-2003: American National Standard for Information Technology – Serial Attached SCSI (SAS), October 30, 2003. http://www.ansi.org/
- Audio Codec '97 Revision 2.3 Revision 1.0, April 2002 Copyright © 2002 Intel Corporation. All rights reserved. download.intel.com/support/motherboards/desktop/sb/ac97_r23.pdf
- Display Data Channel Command Interface (DDC/CI) Standard (formerly DDC2Bi) Version 1, August 14, 1998 Copyright © 1998 Video Electronics Standards Association. All rights reserved. http://www.vesa.org/summary/sumddcci.htm
- ExpressCard® Standard 2.0, June 2009 Copyright © 2009 PCMCIA. All rights reserved. http://www.expresscard.org/
- HDA - High Definition Audio Specification, Revision 1.0, April 15, 2004
  Copyright © 2002 Intel Corporation. All rights reserved.
  http://www.intel.com/standards/hdaudio/
-
- Intel Low Pin Count (LPC) Interface Specification Revision 1.1, August 2002 Copyright © 2002 Intel Corporation. All rights reserved. http://developer.intel.com/design/chipsets/industry/lpc.htm
- PCI Express Base Specification Revision 2.0, December 20, 2006, Copyright © 2002-2006 PCI Special Interest Group. All rights reserved. http://www.pcisig.com/
-
- PCI Local Bus Specification Revision 3.0, February 3, 2004 Copyright © 1992, 1993, 1995, 1998, and 2004 PCI Special Interest Group. All rights reserved. http://www.pcisig.com/
- PICMG® Policies and Procedures for Specification Development, Revision 2.0, September 14, 2004, PCI Industrial Computer Manufacturers Group (PICMG®), 401 Edgewater Place, Suite 500, Wakefield, MA 01880 USA, Tel: 781.224.1100, Fax: 781.224.1239. http://www.picmg.org/
- PICMG® COM Express Specification, PCI Industrial Computer Manufacturers Group (PICMG®), 401 Edgewater Place, Suite 500, Wakefield, MA 01880 USA, Tel: 781.224.1100, Fax: 781.224.1239. http://www.picmg.org/
- SDIO, Secure Digital Input/Output
  SD Specifications Part E1 SDIO Specification Version 2.00, February 8, 2007
  Copyright 2007 SD Card Association
  http://www.sdcard.org
- Serial ATA: High Speed Serialized AT Attachment Revision 1.0a January 7, 2003 Copyright © 2000-2003, APT Technologies, Inc., Dell Computer Corporation, Intel Corporation, Maxtor Corporation, Seagate Technology LLC. All rights reserved. http://www.sata-io.org/
- Smart Battery Data Specification Revision 1.1, December 11, 1998.
  www.sbs-forum.org

- System Management Bus (SMBus) Specification Version 2.0, August 3, 2000 Copyright © 1994, 1995, 1998, 2000 Duracell, Inc., Energizer Power Systems, Inc., Fujitsu, Ltd., Intel Corporation, Linear Technology Inc., Maxim Integrated Products, Mitsubishi Electric Semiconductor Company, PowerSmart, Inc., Toshiba Battery Co. Ltd., Unitrode Corporation, USAR Systems, Inc. All rights reserved. http://www.smbus.org/
- USB 3.0 Specification, Revision 1.0, November 12, 2008, 2000
  Copyright © 2007-2008 Hewlett-Packard Company, Intel Corporation, , Microsoft Corporation, NEC Corporation, ST-NXP Wireless and Texas Instruments. All rights reserved.
  http://www.usb.org/
- SPI, Serial Peripheral Interface Bus
  http://elm-chan.org/docs/spi_e.html
- Trusted Platform Module (TPM), Trusted Computing Group Specification 1.2 Revision 103, July 9, 2007, http://www.trustedcomputinggroup.org
- DisplayPort Standard Version 1.1 http://www.vesa.org
- High-Definition Multimedia Interface specification version 1.3 http://www.hdmi.org

1.8 Statement of Compliance

Statements of compliance with this specification take the form specified in the PICMG® Policies and Procedures for Specification Development:

"This product provides software support for PICMG® EAPI Revision 1.0."

Products making this simple claim of compliance must provide, at a minimum, all features defined in this specification as being mandatory by the use of the keyword "***shall***" in the body of the specification. Such products ***may*** also provide recommended features associated with the keyword "***should***" and permitted features associated with the keyword "***may***" as well.

Because the specification provides for a number of recommended and permitted features beyond the mandatory minimum set and a wide range of performance capabilities, more complete descriptions of product compliance are encouraged.

2 General

COM Express specifies functions for industrial applications which do not feature a common programming interface. Target is to avoid software modifications when changing COM Express module suppliers. This section describes a proposal for a common API to unify the software control for:

- System information
- Watchdog timer
- I2C Bus
- Flat Panel brightness control
- User storage area
- GPIO

The EAPI definition is open to be used for other embedded form factors too.

2.1 Parameters

Parameters which can return a value are defined as pointers to the data. The other parameters are defined as values. The immediate return value is an error code.

Parameters should be validated using 'Atomic Error Checking'(See chapter 1.5 page 9).

2.2 Keywords

In order to improve the readability this documents features keywords used before variables.

2.2.1 __IN

Parameter Type	Characteristics
Immediate value	Input value that must be specified and is essential
Pointer	Valid pointer to initialized buffer/variable.

2.2.2 __OUT

Parameter Type	Characteristics
Pointer	Valid pointer to destination buffer/variable

2.2.3 __INOPT

Parameter Type	Characteristics
Pointer	Valid pointer to initialized buffer/variable, or NULL Pointer. Note: refer to function specification for specifics.

2.2.4 __OUTOPT

Parameter Type	Characteristics
Pointer	Valid pointer to destination buffer/variable, or NULL Pointer. Note: refer to function specification for specifics.

2.2.5 __INOUT

Parameter Type	Characteristics
Pointer	Valid pointer to initialized buffer/variable. Contents of buffer/variable updated before return.

2.3 Status Codes

All ***EApi**** functions immediately return a status code from a common list of possible errors.

Any function may return any of the defined status codes.

2.3.1 Status Code Description

EAPI_STATUS_NOT_INITIALIZED

Description

The EAPI library is not yet or unsuccessfully initialized. ***EApiLibInitialize*** needs to be called prior to the first access of any other EAPI function.

Actions

Call ***EApiLibInitialize***.

EAPI_STATUS_INITIALIZED

Description

Library is initialized.

Actions

none.

EAPI_STATUS_ALLOC_ERROR

Description

Memory Allocation Error.

Actions

Free memory and try again.

2.3.2 EAPI_STATUS_SW_TIMEOUT

Description

Software time out. This is Normally caused by hardware/software semaphore timeout.

Actions

Retry.

EAPI_STATUS_INVALID_PARAMETER

Description

One or more of the EAPI function call parameters are out of the defined range.

Actions

Verify Function Parameters.

EAPI_STATUS_INVALID_BLOCK_LENGTH

Description

This means that the Block length is too long.

Actions

Use relevant Capabilities information to correct select block lengths.

EAPI_STATUS_INVALID_BLOCK_ALIGNMENT

Description

The Block Alignment is incorrect.

Actions

Use Alignment Capabilities information to correctly align write access.

EAPI_STATUS_INVALID_DIRECTION

Description

The current Direction Argument attempts to set GPIOs to a unsupported directions. I.E. Setting GPI to Output.

Actions

Use pInputs and pOutputs to correctly select input and outputs.

EAPI_STATUS_INVALID_BITMASK

Description

The Bitmask Selects bits/GPIOs which are not supported for the current ID.

Actions

Use pInputs and pOutputs to probe supported bits..

EAPI_STATUS_UNSUPPORTED

Description

This function or ID is not supported at the actual hardware environment.

Actions

none.

EAPI_STATUS_NOT_FOUND

Description

Selected device was not found.

Example

The I2C device address is not Acknowledged, device is not present or inactive.

Actions

none.

EAPI_STATUS_BUSY_COLLISION

Description

The selected device or ID is busy or a data collision was detected.

Example

The addressed I2C bus is busy or there is a bus collision.
The I2C bus is in use. Either CLK or DAT are low.
Arbitration loss or bus Collision, data remains low when writing a 1.

Actions

Retry.

EAPI_STATUS_RUNNING

Description

Watchdog timer already started.

Actions

Call EApiWDogStop, before retrying.

2.3.3 EAPI_STATUS_HW_TIMEOUT

Description

Function call timed out

Example

I2C operation lasted too long.

Actions

Retry.

EAPI_STATUS_READ_ERROR

Description

An error was detected during a read operation.

Example

I2C Read function was not successful

Actions

Retry.

EAPI_STATUS_WRITE_ERROR

Description

An error was detected during a write operation.

Example

I2C Write function was not successful.
No Acknowledge was received after writing any byte after the first address byte.
Can be caused by unsupported device command/index.
10Bit Address Device Not Present
Storage Write Error

Actions

Retry.

EAPI_STATUS_MORE_DATA

Description

The amount of available data exceeds the buffer size.
Storage buffer overflow was prevented. Read count was larger then the defined buffer length.

Actions

Either increase the buffer size or reduce the block length.

EAPI_STATUS_ERROR

Description

Generic error message. No further error details are available.

Actions

none.

EAPI_STATUS_SUCCESS

The value for this status code is defined as 0.

Description

The operation was successful.

Actions

None.

Macros for Status Codes

Name	Description
EAPI_TEST_SUCCESS	Can be used to check if the status is **EAPI_STATUS_SUCCESS**.

3 Initialization Functions

3.1 EApiLibInitialize

```
uint32_t
EAPI_CALLTYPE
EApiLibInitialize(void);
```

FUNC_DEF 1: EApiLibInitialize

3.1.1 Description:

General initialization of the EAPI. Prior to calling any EAPI function the library needs to be initialized by calling this function. The status code for all EAPI function will be **EAPI_STATUS_NOT_INITIALIZED** unless this function is called.

3.1.2 Parameters

None.

3.1.3 Return Status Code

Condition	Return Value
Library initialized	***EAPI_STATUS_INITIALIZED***
...	see 2.3 Status Codes on page 16
Success	***EAPI_STATUS_SUCCESS***

3.2 EApiLibUnInitialize

```
uint32_t
EAPI_CALLTYPE
EApiLibUnInitialize(void);
```

FUNC_DEF 2: EApiLibUnInitialize

3.2.1 Description:

General function to uninitialized the EAPI library. Should be called before program exit.

In a dynamic library environment this function is not expected to replace the native uninitialize routines. It Is expected that in this environments this function has no functionality.

3.2.2 Parameters

None.

3.2.3 Return Status Code

Condition	Return Value
Library Uninitialized	***EAPI_STATUS_NOT_INITIALIZED***
...	see 2.3 Status Codes on page 16
Success	***EAPI_STATUS_SUCCESS***

4 EAPI Information Functions

4.1 EApiBoardGetStringA

```
uint32_t
EAPI_CALLTYPE
EApiBoardGetStringA(
    __IN      uint32_t  Id      , /* Name Id */
    __OUT     char     *pBuffer , /* Destination pBuffer */
    __INOUT   uint32_t *pBufLen   /* pBuffer Length */
    );
```

FUNC_DEF 3: EApiBoardGetStringA

4.1.1 Description:

Text information about the hardware platform.

4.1.2 Parameters

Id

__IN Selects the Get String Sub function Id.

Id	Description	Example
EAPI_ID_BOARD_MANUFACTURER_STR	Board Manufacturer Name	PICMG
EAPI_ID_BOARD_NAME_STR	Board Name	SAMPLE API
EAPI_ID_BOARD_SERIAL_STR	Serial Number	Sample Serial Number
EAPI_ID_BOARD_BIOS_REVISION_STR	Board BIOS Revision	PICMGR2.0
EAPI_ID_BOARD_PLATFORM_TYPE_STR	Platform ID See 'Platform Specification' on page 22	COMExpress

pBuffer

__OUT Pointer to a buffer that receives the value's data. This parameter can be NULL if the data is not required

pBufLen

____INOUT Pointer to a variable that specifies the size, in bytes, of the buffer pointed to by the ***pBuffer*** parameter. When the function returns, this variable contains the size of the data copied to ***pBuffer*** including the terminating null character.

If the buffer specified by **pBuffer** parameter is not large enough to hold the data, the function returns the value **EAPI_STATUS_MORE_DATA** and stores the required buffer size, in bytes, into the variable pointed to by ***pBufLen***.

If **pBuffer** is NULL, and **pBufLen** is non-NULL, the function returns ***EAPI_STATUS_MORE_DATA***, and stores the size of the data, in bytes, in the variable pointed to by ***pBufLen***. This lets an application determine the best way to allocate a buffer for the value's data.

Find more details refer to the platform specific header files. The Platform ID is defined by the specification and is “COMExpress” for COM Express.

4.1.3 Return Status Code

Condition	Return Value
Library Uninitialized	***EAPI_STATUS_NOT_INITIALIZED***
pBufLen==NULL	***EAPI_STATUS_INVALID_PARAMETER***
*pBufLen&&pBuffer==NULL	***EAPI_STATUS_INVALID_PARAMETER***
unknown Id	***EAPI_STATUS_UNSUPPORTED***
Id String Length +1>*pBufLen	***EAPI_STATUS_MORE_DATA***
...	see 2.3 Status Codes on page 16
Success	***EAPI_STATUS_SUCCESS***

4.2 EApiBoardGetValue

```
uint32_t
EAPI_CALLTYPE
EApiBoardGetValue(
    __IN  uint32_t  Id       , /* Value Id */
    __OUT uint32_t *pValue     /* Return Value */
    );
```

FUNC_DEF 4: EApiBoardGetValue

4.2.1 Description

Information about the hardware platform in value format.

4.2.2 Parameters

Id

__IN Selects the Get Value Sub function Id.

Id	Description	Units/Format
EAPI_ID_GET_EAPI_SPEC_VERSION	EAPI Specification Version used to implement API	'Specification Version Number Formatt' see page 77
EAPI_ID_BOARD_BOOT_COUNTER_VAL	'Boot Counter' see page 9	boots[1]
EAPI_ID_BOARD_RUNNING_TIME_METER_VAL	'Running Time Meter' see page 11	minutes[1]
EAPI_ID_BOARD_PNPID_VAL	Board Vendor PNPID see page 11	'Compressed ASCII PNPID' page 77
EAPI_ID_BOARD_PLATFORM_REV_VAL	Platform Specification Version used to create board. See 'Platform ID' on page 21	'Specification Version Number Format' see page 77
EAPI_ID_BOARD_DRIVER_VERSION_VAL	Vendor Specific Driver Version	'General Version number Format'[1] page 78

1 ***may*** be supported

Id	Description	Units/Format
EAPI_ID_BOARD_LIB_VERSION_VAL	Vendor Specific Library Version	'General Version number Format'[1] page 78
EAPI_ID_HWMON_CPU_TEMP	CPU Temperature	0.1 Kelvins[1]
EAPI_ID_HWMON_CHIPSET_TEMP	Chipset Temperature	0.1 Kelvins[1]
EAPI_ID_HWMON_SYSTEM_TEMP	System Temperature	0.1 Kelvins[1]
EAPI_ID_HWMON_VOLTAGE_VCORE	CPU Core Voltage	millivolts[1]
EAPI_ID_HWMON_VOLTAGE_2V5	2.5V Voltage	millivolts[1]
EAPI_ID_HWMON_VOLTAGE_3V3	3.3V Voltage	millivolts[1]
EAPI_ID_HWMON_VOLTAGE_VBAT	Battery Voltage	millivolts[1]
EAPI_ID_HWMON_VOLTAGE_5V	5V Voltage	millivolts[1]
EAPI_ID_HWMON_VOLTAGE_5VSB	5V Standby Voltage	millivolts[1]
EAPI_ID_HWMON_VOLTAGE_12V	12V Voltage	millivolts[1]
EAPI_ID_HWMON_FAN_CPU	CPU Fan	RPM[1]
EAPI_ID_HWMON_FAN_SYSTEM	System Fan	RPM[1]

pValue

__OUT Pointer to a buffer that receives the value's data.

4.2.3 Return Status Code

Condition	Return Value
Library Uninitialized	***EAPI_STATUS_NOT_INITIALIZED***
pValue==NULL	***EAPI_STATUS_INVALID_PARAMETER***
unknown Id	***EAPI_STATUS_UNSUPPORTED***
...	see 2.3 Status Codes on page 16
Success	***EAPI_STATUS_SUCCESS***

5 Backlight Functions

This function sub set facilitates backlight control for Integrated flat panel displays, typically LVDS.

5.1 Common Parameters for Backlight Functions

5.1.1 Backlight Ids

Selects the flat panel display.

Id	Description
EAPI_ID_BACKLIGHT_1	Backlight Local Flat Panel 1
EAPI_ID_BACKLIGHT_2	Backlight Local Flat Panel 2
EAPI_ID_BACKLIGHT_3	Backlight Local Flat Panel 3

The IDs for the backlights are filled according to this fill order:

- Internal PWM
- EAPI_COM0_ID_I2C_LVDS_1 I2C Device
- SDVOB PWM (via SDVO to LVDS converter)
- SDVOB I2C Device (via SDVO to LVDS converter)
- SDVOC PWM (via SDVO to LVDS converter)
- SDVOC I2C Device (via SDVO to LVDS converter)
- DDI1 I2C Device eDP
- DDI2 I2C Device eDP
- DDI3 I2C Device eDP

The EAPI expects that the backlight control hardware for the SDVO ports is implemented according to the Intel SDVO specification.

5.1.2 Backlight Enable Values

Name	Description
EAPI_BACKLIGHT_SET_ON	Requests/Signifies that the Backlight be Enabled
EAPI_BACKLIGHT_SET_OFF	Requests/Signifies that the Backlight be Disabled

5.1.3 Backlight Brightness Value Range

Name	Description
EAPI_BACKLIGHT_SET_DIMMEST	Represents the lower range bound for the backlight brightness
EAPI_BACKLIGHT_SET_BRIGHTEST	Represents the upper range bound for the backlight brightness

5.2 EApiVgaGetBacklightEnable

```
uint32_t
EAPI_CALLTYPE
EApiVgaGetBacklightEnable(
    __IN  uint32_t  Id      , /* Backlight Id */
    __OUT uint32_t *pEnable   /* Backlight Enable */
    );
```

FUNC_DEF 5: EApiVgaGetBacklightEnable

5.2.1 Description

Returns current Backlight Enable state for specified Flat Panel.

5.2.2 Parameters

Id

__IN See '5.1.1 Backlight Ids' on page 24

pEnable

__OUT Pointer to a buffer that receives the the current backlight enable state.

See 5.1.2 Backlight Enable Values on page 24.

5.2.3 Return Status Code

Condition	Return Value
Library Uninitialized	***EAPI_STATUS_NOT_INITIALIZED***
pEnable==NULL	***EAPI_STATUS_INVALID_PARAMETER***
unknown Id	***EAPI_STATUS_UNSUPPORTED***
…	see 2.3 Status Codes on page 16
Success	***EAPI_STATUS_SUCCESS***

5.3 EApiVgaSetBacklightEnable

```
uint32_t
EAPI_CALLTYPE
EApiVgaSetBacklightEnable(
    __IN  uint32_t  Id      , /* Backlight Id */
    __IN  uint32_t  Enable    /* Backlight Enable */
    );
```

FUNC_DEF 6: EApiVgaSetBacklightEnable

5.3.1 Description

Enables the backlight of the selected flat panel display.

5.3.2 Parameters

Id

__IN See '5.1.1 Backlight Ids' on page 24

Enable

__IN Backlight Enable options. See 5.1.2 Backlight Enable Values on page 24.

5.3.3 Return Status Code

Condition	Return Value
Library Uninitialized	***EAPI_STATUS_NOT_INITIALIZED***
Enable!=EAPI_BACKLIGHT_SET_ON && Enable!=EAPI_BACKLIGHT_SET_OFF	***EAPI_STATUS_INVALID_PARAMETER***
unknown Id	***EAPI_STATUS_UNSUPPORTED***
…	see 2.3 Status Codes on page 16
Success	***EAPI_STATUS_SUCCESS***

5.4 EApiVgaGetBacklightBrightness

```
uint32_t
EAPI_CALLTYPE
EApiVgaGetBacklightBrightness(
    __IN  uint32_t  Id      , /* Backlight Id */
    __OUT uint32_t *pBright   /* Backlight Brightness */
    );
```

FUNC_DEF 7: EApiVgaGetBacklightBrightness

5.4.1 Description

Reads the current brightness of the selected flat panel display.

5.4.2 Parameters

Id

__IN See '5.1.1 Backlight Ids' on page 24

pBright

__OUT Pointer to a buffer that receives the the current backlight brightness level.

See 5.1.3 Backlight Brightness Value Range on page 24.

5.4.3 Return Status Code

Condition	Return Value
Library Uninitialized	***EAPI_STATUS_NOT_INITIALIZED***
pBright==NULL	***EAPI_STATUS_INVALID_PARAMETER***
unknown Id	***EAPI_STATUS_UNSUPPORTED***
...	see 2.3 Status Codes on page 16
Success	***EAPI_STATUS_SUCCESS***

5.5 EApiVgaSetBacklightBrightness

```
uint32_t
EAPI_CALLTYPE
EApiVgaSetBacklightBrightness(
    __IN  uint32_t  Id      , /* Backlight Id */
    __IN  uint32_t  Bright    /* Backlight Brightness */
    );
```

FUNC_DEF 8: EApiVgaSetBacklightBrightness

5.5.1 Description

Sets the brightness of the selected flat panel display.

5.5.2 Parameters

Id

__IN See '5.1.1 Backlight Ids' on page 24

Bright

__IN Backlight Brightness level. See 5.1.3 Backlight Brightness Value Range on page 24.

5.5.3 Return Status Code

Condition	Return Value
Library Uninitialized	***EAPI_STATUS_NOT_INITIALIZED***
Bright>EAPI_BACKLIGHT_SET_BRIGHTEST	***EAPI_STATUS_INVALID_PARAMETER***
unknown Id	***EAPI_STATUS_UNSUPPORTED***
...	see 2.3 Status Codes on page 16
Success	***EAPI_STATUS_SUCCESS***

6 Storage Functions

The EAPI defines one user storage area with a minimal size of 32 Byte.

6.1 Common Parameters

6.1.1 Storage Ids

The EAPI only defines one user storage area. Additional vendor specific IDs are possible (see section 11 'Vendor Specific IDs' on page 76)

Id	Description
EAPI_ID_STORAGE_STD	Standard Storage Area >=32Bytes for read/write access

6.2 EApiStorageCap

```
uint32_t
EAPI_CALLTYPE
EApiStorageCap(
    __IN  uint32_t  Id             , /* Storage Area Id */
    __OUT uint32_t  *pStorageSize  , /* Total */
    __OUT uint32_t  *pBlockLength    /* Write Block Length
                                      * & Alignment
                                      */

    );
```

FUNC_DEF 9: EApiStorageCap

6.2.1 Description

Get the capabilities of the selected storage area.

6.2.2 Parameters

Id

__IN See '6.1.1 Storage Ids' on page 28

pStorageSize

__OUTOPT Pointer to a buffer that receives storage area size. This parameter can be NULL if the data is not required.

pBlockLength

__OUTOPT Pointer to a buffer that receives the the storage areas alignment/Block size. This parameter can be NULL if the data is not required. The value must be used to calculate write block alignment and size.

6.2.3 Return Status Codes

Condition	Return Value
Library Uninitialized	***EAPI_STATUS_NOT_INITIALIZED***
((pStorageSize==NULL)&& (pBlockLength==NULL))	***EAPI_STATUS_INVALID_PARAMETER***
Unsupported Id	***EAPI_STATUS_UNSUPPORTED***
...	see 2.3 Status Codes on page 16
Success	***EAPI_STATUS_SUCCESS***

6.3 EApiStorageAreaRead

```
uint32_t
EAPI_CALLTYPE
EApiStorageAreaRead(
    __IN  uint32_t  Id      , /* Storage Area Id */
    __IN  uint32_t  Offset  , /* Byte Offset */
    __OUT     void *pBuffer , /* Pointer to Data pBuffer */
    __IN  uint32_t  BufLen  , /* Data pBuffer Size in
                               * bytes
                               */
    __IN  uint32_t  ByteCnt   /* Number of bytes to read */
    );
```

FUNC_DEF 10: EApiStorageAreaRead

6.3.1 Description

Writes data to the selected user data area.

6.3.2 Parameters

Id

__IN See '6.1.1 Storage Ids' on page 28

Offset

__IN Storage area start address offset in bytes.

pBuffer

__OUT Pointer to a buffer that receives the read data.

BufLen

__IN Size, in bytes, of the buffer pointed to by the **pBuffer** parameter

ByteCnt

__IN Size, in bytes, of the information read to the buffer pointed to by the **pBuffer** parameter

6.3.3 Return Status Codes

Condition	Return Value
Library Uninitialized	***EAPI_STATUS_NOT_INITIALIZED***
pBuffer==NULL	***EAPI_STATUS_INVALID_PARAMETER***
ByteCnt==0	***EAPI_STATUS_INVALID_PARAMETER***
BufLen==0	***EAPI_STATUS_INVALID_PARAMETER***
unknown Id	***EAPI_STATUS_UNSUPPORTED***
Offset+ByteCnt>pStorageSize	***EAPI_STATUS_INVALID_BLOCK_LENGTH***
Read Error	***EAPI_STATUS_READ_ERROR***
ByteCnt>BufLen	***EAPI_STATUS_MORE_DATA***
...	see 2.3 Status Codes on page 16
Success	***EAPI_STATUS_SUCCESS***

6.4 EApiStorageAreaWrite

```
uint32_t
EAPI_CALLTYPE
EApiStorageAreaWrite(
    __IN  uint32_t  Id      , /* Storage Area Id */
    __IN  uint32_t  Offset  , /* Byte Offset */
    __IN      void *pBuffer , /* Pointer to Data pBuffer */
    __IN  uint32_t  ByteCnt   /* Number of bytes to write*/
    );
```

FUNC_DEF 11: EApiStorageAreaWrite

6.4.1 Description

Writes data to the selected user data area.

6.4.2 Parameters

Id

__IN See '6.1.1 Storage Ids' on page 28

Offset

__IN Storage area start address offset in bytes. This value must be a multiple of ***pBlockLength.**

pBuffer

__IN Pointer to a buffer containing the data to be stored.

ByteCnt

__IN Size, in bytes, of the information pointed to by the **pBuffer** parameter

6.4.3 Return Status Codes

Condition	Return Value
Library Uninitialized	***EAPI_STATUS_NOT_INITIALIZED***
pBuffer==NULL	***EAPI_STATUS_INVALID_PARAMETER***
ByteCnt==0	***EAPI_STATUS_INVALID_PARAMETER***
unknown Id	***EAPI_STATUS_UNSUPPORTED***
Offset%pBlockLength	***EAPI_STATUS_INVALID_BLOCK_ALIGNMENT***
ByteCnt%pBlockLength	***EAPI_STATUS_INVALID_BLOCK_ALIGNMENT***
Offset+ByteCnt>pStorageSize	***EAPI_STATUS_INVALID_BLOCK_LENGTH***
Write Error	***EAPI_STATUS_WRITE_ERROR***
...	see 2.3 Status Codes on page 16
Success	***EAPI_STATUS_SUCCESS***

7 Functions for the I2C Bus

Set of function to access the I2C bus.

7.1 Common Parameters

7.1.1 I2C Bus Ids

The EAPI specification currently defines three I2C buses for COM Express. Additional vendor or platform specific IDs are possible (see section 11 'Vendor Specific IDs' on page 76)

Id	Description
EAPI_ID_I2C_EXTERNAL	Baseboard I2C Interface
EAPI_ID_I2C_LVDS_1	LVDS/EDP 1 Interface
EAPI_ID_I2C_LVDS_2	LVDS/EDP 2 Interface

7.1.2 LVDS Ids Fill order.

Only required if multiple Local Flat panels present.

- Internal LFP
- SDVOB
- SDVOC
- DDI1
- DDI2
- DDI3

7.2 Address Format for the I2C Bus

The I2C specification defines a 7 bit and a 10 bit address format. Both formats can be used for the High level functions EapiI2CReadRegister and EapiI2CWriteRegister. Only 7 bit addresses are supported for the low level function EapiI2CWriteRead. This is because 10 Bit addresses are realized in the I2C Specification as an extended write read transfer and are not addressable as 7 Bit devices.

7 Bit and 10 Bit I2C Address format

```
/*
 *  L = Set to 0
 *  H = Set to 1
 *  X = Don't Care
 *  0-F Address Bit
 *
 *  Bits 31-16 are Reserved and should be set to 0
 *
 *  Bit Offset       F E D C B A 9 8 7 6 5 4 3 2 1 0
 *  7  Bit Address   L L L L L L L L 6 5 4 3 2 1 0 X
 *  10 Bit Address   H H H H L 9 8 X 7 6 5 4 3 2 1 0
 *
 *  Examples where Don't Care bits set to 0
 *              Encoded Encoded
 *  Address      7Bit    10Bit
 *    0x01       0x02   0xF001
 *    0x58       0xA0   0xF058
 *    0x59       0xA2   0xF059
 *    0x77       0xEE   0xF077
 *    0x3FF             0xF6FF
 *
 */
```

Macros for address decoding and encoding

Name	Description
EAPI_I2C_ENC_7BIT_ADDR()	Encodes 7Bit I2C Address
EAPI_I2C_ENC_10BIT_ADDR()	Encodes 10Bit I2C Address

Name	Description
EAPI_I2C_DEC_7BIT_ADDR()	Decodes 7Bit I2C Address
EAPI_I2C_DEC_10BIT_ADDR()	Decodes 10Bit I2C Address

Name	Description
EAPI_I2C_IS_7BIT_ADDR()	Checks if 7Bit I2C Address
EAPI_I2C_IS_10BIT_ADDR()	Checks if 10Bit I2C Address

7.3 I2CTransfer Types

Transfer Type 1:

Address Format : 7Bit
Command Type : None
Data Direction : Write
Start<Addr Byte 1 LSB><W>Ack<Data Byte 1>Ack Stop

Transfer Type 2:

Address Format : 7Bit
Command Type : None
Data Direction : Read
Start<Addr Byte 1 LSB><R>Ack<Data Byte 1>Nak Stop

Transfer Type 3:

Address Format : 7Bit
Command Type : Standard
Data Direction : Write
Start<Addr Byte LSB><W>Ack<CMD Byte >Ack<Data Byte 1>Ack Stop

Transfer Type 4:

Address Format : 7Bit
Command Type : Standard
Data Direction : Read
Start<Addr Byte LSB><W>Ack<CMD Byte>Ack

Start<Addr Byte LSB><R>Ack<Data Byte 1>Nak Stop

Transfer Type 5:

Address Format : 7Bit
Command Type : Extended
Data Direction : Write
Start<Addr Byte LSB><W>Ack<CMD Byte MSB>Ack<CMD Byte LSB>Ack<Data Byte 1>Ack Stop

Transfer Type 6:

Address Format : 7Bit
Command Type : Extended
Data Direction : Read
Start<Addr Byte LSB><W>Ack<CMD Byte MSB>Ack<CMD Byte LSB>Ack
Start<Addr Byte LSB><R>Ack<Data Byte 1>Nak Stop

Transfer Type 7:

Address Format : 10Bit
Command Type : None
Data Direction : Write
Start<Addr Byte MSB><W>Ack<Addr Byte LSB>Ack<Data Byte 1>Ack Stop

Transfer Type 8:

Address Format : 10Bit
Command Type : None
Data Direction : Read
Start<Addr Byte MSB><W>Ack<Addr Byte LSB>Ack
Start<Addr Byte MSB><R>Ack<Data Byte 1>Nak Stop

Transfer Type 9:

Address Format : 10Bit
Command Type : Standard
Data Direction : Write
Start<Addr Byte MSB><W>Ack<Addr Byte LSB>Ack<CMD Byte >Ack<Data Byte 1>Ack Stop

Transfer Type 10:

Address Format : 10Bit
Command Type : Standard
Data Direction : Read
Start<Addr Byte MSB><W>Ack<Addr Byte LSB>Ack<CMD Byte >Ack
Start<Addr Byte MSB><R>Ack<Data Byte 1>Nak Stop

Transfer Type 11:

Address Format : 10Bit
Command Type : Extended
Data Direction : Write
Start<Addr Byte MSB><W>Ack<Addr Byte LSB>Ack<CMD Byte MSB>Ack<CMD Byte LSB>Ack<Data Byte 1>Ack Stop

Transfer Type 12:

Address Format : 10Bit
Command Type : Extended
Data Direction : Read
Start<Addr Byte MSB><W>Ack<Addr Byte LSB>Ack<CMD Byte MSB>Ack<CMD Byte LSB>Ack
Start<Addr Byte MSB><R>Ack<Data Byte 1>Nak Stop

Macros for Command/Index Encoding/Decoding

Name	Description
EAPI_I2C_STD_CMD	Signifies Transfer using Standard Command
EAPI_I2C_EXT_CMD	Signifies Transfer using Extended Command
EAPI_I2C_NO_CMD	Signifies Transfer using No Command
EAPI_I2C_CMD_TYPE_MASK	Used to mask Command Request Bits

Name	Description
EAPI_I2C_ENC_STD_CMD()	Encodes Request for Standard Command
EAPI_I2C_ENC_EXT_CMD()	Encodes Request for Extended Command

Name	Description
EAPI_I2C_IS_STD_CMD()	Tests if Transfer using Standard Command is requested
EAPI_I2C_IS_EXT_CMD()	Tests if Transfer using Extended Command is requested
EAPI_I2C_IS_NO_CMD()	Tests if Transfer using No Command is requested

7.4 I2C Device Probe Types

*

Probe Type 1:

Address Format : 7Bit
Start<Addr Byte LSB><W>Ack Stop

Probe Type 2:

Address Format : 10Bit
Start<Addr Byte MSB><W>Ack<Addr Byte LSB>Ack Stop

7.5 EapiI2CGetBusCap

```
uint32_t
EAPI_CALLTYPE
EApiI2CGetBusCap(
    __IN  uint32_t  Id          , /* I2C Bus Id */
    __OUT uint32_t *pMaxBlkLen    /* Max Block Length
                                   * Supported on this
                                   * interface
                                   */

    );
```

FUNC_DEF 12: EApiI2CGetBusCap

7.5.1 Description

Returns the capabilities of the selected I2C bus.

7.5.2 Parameters

Id

__IN See '7.1.1 I2C Bus Ids' on page 32

pMaxBlkLen

__OUT size in bytes, Pointer to a buffer that receives the maximum transfer block length for the given interface. Please note care must be taken when using in combination with ***EapiI2CWriteTransfer*** as the maximum data payload length is then ***pMaxBlkLen***-(write overhead). So for example a 10 Bit Addressed device with Extended command has a write overhead of 3 bytes. Address Byte 2 and 2 byte command.

7.5.3 Return Status Codes

Condition	Return Value
Library Uninitialized	***EAPI_STATUS_NOT_INITIALIZED***
pMaxBlkLen==NULL	***EAPI_STATUS_INVALID_PARAMETER***
unknown Id	***EAPI_STATUS_UNSUPPORTED***
...	see 2.3 Status Codes on page 16
Success	***EAPI_STATUS_SUCCESS***

7.6 EapiI2CWriteReadRaw

```
uint32_t
EAPI_CALLTYPE
EApiI2CWriteReadRaw(
    __IN     uint32_t  Id        , /* I2C Bus Id */
    __IN     uint8_t   Addr      , /* Encoded 7Bit I2C
                                    * Device Address
                                    */
    __INOPT  void     *pWBuffer , /* Write Data pBuffer */
    __IN     uint32_t  WriteBCnt, /* Number of Bytes to
                                    * write
                                    */
    __OUTOPT void     *pRBuffer , /* Read Data pBuffer */
    __IN     uint32_t  RBufLen  , /* Data pBuffer Length */
    __IN     uint32_t  ReadBCnt   /* Number of Bytes to
                                    * Read
                                    */

    );
```

FUNC_DEF 13: EApiI2CWriteReadRaw

7.6.1 Description

Universal function for read and write operations to the I2C bus.

7.6.2 Parameters

Id

__IN See '7.1.1 I2C Bus Ids' on page 32

Addr

__IN First Byte of I2C Device Address.

pWBuffer

__INOPT Pointer to a buffer containing the data to be transferred. This parameter can be NULL if the data is not required.

WriteBCnt

__IN Size, in bytes, of the information pointed to by the **pWBuffer** parameter plus 1 If **pWBuffer** is NULL this must be zero or one.

pRBuffer

__OUTOPT Pointer to a buffer that receives the read data. This parameter can be NULL if the data is not required.

RBufLen

__IN Size, in bytes, of the buffer pointed to by the **pRBuffer** parameter.

If the buffer specified by **pRBuffer** parameter is not large enough to hold the data, the function returns the value **EAPI_STATUS_MORE_DATA**

If **pRBuffer** is NULL this must be zero.

ReadBCnt

__IN Size, in bytes, to be read to **pRBuffer** plus 1 If **pRBuffer** is NULL this must be zero or one.

7.6.3 Return Status Codes

Condition	Return Value
Library Uninitialized	***EAPI_STATUS_NOT_INITIALIZED***
(WriteBCnt>1)&&(pWBuffer==NULL)	***EAPI_STATUS_INVALID_PARAMETER***
(ReadBCnt>1)&&(pRBuffer==NULL)	***EAPI_STATUS_INVALID_PARAMETER***
(ReadBCnt>1)&&(RBufLen==0)	***EAPI_STATUS_INVALID_PARAMETER***
((WriteBCnt==0)&&(ReadBCnt==0))	***EAPI_STATUS_INVALID_PARAMETER***
unknown Id	***EAPI_STATUS_UNSUPPORTED***
WriteBCnt>(pMaxBlkLen+1)	***EAPI_STATUS_INVALID_BLOCK_LENGTH***
ReadBCnt>(pMaxBlkLen+1)	***EAPI_STATUS_INVALID_BLOCK_LENGTH***
Bus Busy SDA/SDC low	***EAPI_STATUS_BUSY_COLLISION***
Arbitration Error/Collision Error On Write 1 write cycle SDA Remains low	***EAPI_STATUS_BUSY_COLLISION***
Timeout due to clock stretching	***EAPI_STATUS_HW_TIMEOUT***
start<AddrByte1><W>Nak	***EAPI_STATUS_NOT_FOUND***
start<AddrByte1><R>Nak	***EAPI_STATUS_NOT_FOUND***
start<AddrByte1><W>Ack<WriteByte1>Nak	***EAPI_STATUS_WRITE_ERROR***
ReadBCnt>(RBufLen+1)	***EAPI_STATUS_MORE_DATA***
...	see 2.3 Status Codes on page 16
Success	***EAPI_STATUS_SUCCESS***

7.7 EapiI2CReadTransfer

```
uint32_t
EAPI_CALLTYPE
EApiI2CReadTransfer(
    __IN  uint32_t  Id      , /* I2C Bus Id */
    __IN  uint32_t  Addr    , /* Encoded 7/10Bit I2C
                               * Device Address
                               */
    __IN  uint32_t  Cmd     , /* I2C Command/Offset */
    __OUT     void *pBuffer , /* Transfer Data pBuffer */
    __IN  uint32_t  BufLen  , /* Data pBuffer Length */
    __IN  uint32_t  ByteCnt   /* Byte Count to read */
    );
```

FUNC_DEF 14: EApiI2CReadTransfer

7.7.1 Description

Reads from a specific register in the selected I2C device.

Reads from I2C device at the I2C address ***Addr*** the amount of ***ByteCnt*** bytes to the buffer ***pBuffer*** while using the device specific command ***Cmd***. Depending on the addressed I2C device ***Cmd*** can be a specific command or a byte offset.

7.7.2 Parameters

Id

__IN See '7.1.1 I2C Bus Ids' on page 32

Addr

__IN Encoded 7/10 Bit I2C Device Address.

Cmd

__IN Encoded I2C Device Command / Index.

pBuffer

__OUT Pointer to a buffer that receives the read data. This parameter can be NULL if the data is not required.

BufLen

__IN Size, in bytes, of the buffer pointed to by the ***pBuffer*** parameter.

If the buffer specified by ***pBuffer*** parameter is not large enough to hold the data, the function returns the value ***EAPI_STATUS_MORE_DATA***

ByteCnt

__IN Size, in bytes, of data to be read

7.7.3 Return Status Codes

Condition	Return Value
Library Uninitialized	***EAPI_STATUS_NOT_INITIALIZED***
pBuffer==NULL	***EAPI_STATUS_INVALID_PARAMETER***
ByteCnt==0	***EAPI_STATUS_INVALID_PARAMETER***
BufLen==0	***EAPI_STATUS_INVALID_PARAMETER***
Unknown Id	***EAPI_STATUS_UNSUPPORTED***
ByteCnt>pMaxBlkLen	***EAPI_STATUS_INVALID_BLOCK_LENGTH***
Bus Busy SDA/SDC low	***EAPI_STATUS_BUSY_COLLISION***
Arbitration Error/Collision Error On Write 1 write cycle SDA Remains low	***EAPI_STATUS_BUSY_COLLISION***
Timeout due to clock stretching	***EAPI_STATUS_HW_TIMEOUT***
start<AddrByte1><W>Nak	***EAPI_STATUS_NOT_FOUND***
start<AddrByte1><R>Nak	***EAPI_STATUS_NOT_FOUND***
start<Addr Byte 2><W>Ack<Addr Byte 1>Nak	***EAPI_STATUS_WRITE_ERROR Or EAPI_STATUS_NOT_FOUND***
start<Addr Byte 1><W>Ack<Cmd Byte 1>Nak	***EAPI_STATUS_WRITE_ERROR***
start<Addr Byte 1><W>Ack<Data Byte 1>Nak	***EAPI_STATUS_WRITE_ERROR***
ByteCnt>BufLen	***EAPI_STATUS_MORE_DATA***
...	see 2.3 Status Codes on page 16
Success	***EAPI_STATUS_SUCCESS***

7.8 EapiI2CWriteTransfer

```
uint32_t
EAPI_CALLTYPE
EApiI2CWriteTransfer(
    __IN  uint32_t  Id      , /* I2C Bus Id */
    __IN  uint32_t  Addr    , /* Encoded 7/10Bit I2C
                               * Device Address
                               */
    __IN  uint32_t  Cmd     , /* I2C Command/Offset */
    __IN      void *pBuffer , /* Transfer Data pBuffer */
    __IN  uint32_t  ByteCnt   /* Byte Count to write */
    );
```

FUNC_DEF 15: EApiI2CWriteTransfer

7.8.1 Description

Write to a specific register in the selected I2C device.

Writes to an I2C device at the I2C address ***Addr*** the amount of ***ByteCnt*** bytes from the buffer ****pBuffer*** while using the device specific command ***Cmd***. Depending on the addressed I2C device ***Cmd*** can be a specific command or a byte offset.

7.8.2 Parameters

Id

__IN See '7.1.1 I2C Bus Ids' on page 32

Addr

__IN Encoded 7/10 Bit I2C Device Address.

Cmd

__IN Encoded I2C Device Command / Index.

pBuffer

__IN Pointer to a buffer containing the data to be transferred.

ByteCnt

__IN Size, in bytes, of the information pointed to by the **pBuffer** parameter

7.8.3 Return Status Codes

Condition	Return Value
Library Uninitialized	***EAPI_STATUS_NOT_INITIALIZED***
pBuffer==NULL	***EAPI_STATUS_INVALID_PARAMETER***
ByteCnt=0	***EAPI_STATUS_INVALID_PARAMETER***
unknown Id	***EAPI_STATUS_UNSUPPORTED***
ByteCnt+(overhead)>pMaxBlkLen	***EAPI_STATUS_INVALID_BLOCK_LENGTH***
Bus Busy SDA/SDC low	***EAPI_STATUS_BUSY_COLLISION***
Arbitration Error/Collision Error On Write 1 write cycle SDA Remains low	***EAPI_STATUS_BUSY_COLLISION***
Timeout due to clock stretching	***EAPI_STATUS_HW_TIMEOUT***
start<AddrByte1><W>Nak	***EAPI_STATUS_NOT_FOUND***
start<AddrByte1><R>Nak	***EAPI_STATUS_NOT_FOUND***
start<Addr Byte 2><W>Ack<Addr Byte 1>Nak	***EAPI_STATUS_WRITE_ERROR Or EAPI_STATUS_NOT_FOUND***
start<Addr Byte 1><W>Ack<Cmd Byte 1>Nak	***EAPI_STATUS_WRITE_ERROR***
start<Addr Byte 1><W>Ack<Data Byte 1>Nak	***EAPI_STATUS_WRITE_ERROR***
...	see 2.3 Status Codes on page 16
Success	***EAPI_STATUS_SUCCESS***

7.9 EApiI2CProbeDevice

```
uint32_t
EAPI_CALLTYPE
EApiI2CProbeDevice(
    __IN  uint32_t  Id   , /* I2C Bus Id */
    __IN  uint32_t  Addr   /* Encoded 7/10Bit
                           * I2C Device Address
                           */
    );
```

FUNC_DEF 16: EApiI2CProbeDevice

7.9.1 Description

Probes I2C address to test I2C Device present.

7.9.2 Parameters

Id

__IN See '7.1.1 I2C Bus Ids' on page 32

Addr

__IN Encoded 7/10 Bit I2C Device Address.

7.9.3 Return Status Codes

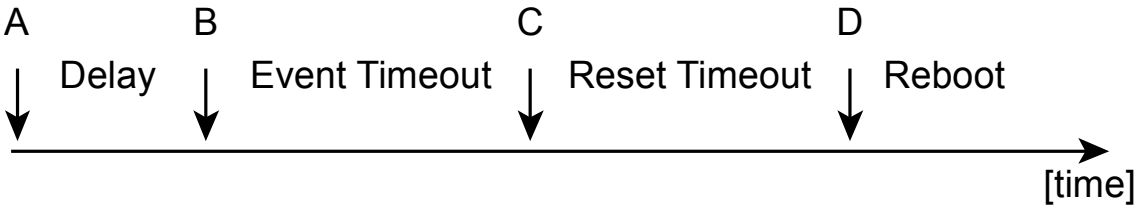
Condition	Return Value
Library Uninitialized	***EAPI_STATUS_NOT_INITIALIZED***
Bus Busy SDA/SDC low	***EAPI_STATUS_BUSY_COLLISION***
Arbitration Error/Collision Error **On Write 1 write cycle** **SDA Remains low**	***EAPI_STATUS_BUSY_COLLISION***
Timeout due to clock stretching	***EAPI_STATUS_HW_TIMEOUT***
7bit Address	
start<AddrByte1><W>Nak	***EAPI_STATUS_NOT_FOUND***
10bit Address	
start<AddrByte1><W>Nak	***EAPI_STATUS_NOT_FOUND***
start<Addr Byte 2><W>Ack<Addr Byte 1>Nak	***EAPI_STATUS_NOT_FOUND***
...	***see 2.3 Status Codes on page 16***
Success	***EAPI_STATUS_SUCCESS***

8 WATCHDOG

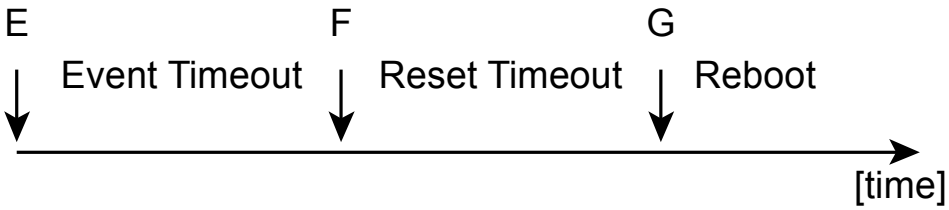
After the watchdog timer has been set by the ***EApiWDogStart*** function it must be triggered by ***EApiWDogTrigger*** within **(*delay*+*EventTimeout)*** milliseconds as set with the ***EApiWDogStart*** function, following the initial trigger every subsequent trigger must occur within **(*EventTimeout)*** milliseconds. Should ***EApiWDogTrigger*** not be called within the relevant time limit a system reset will occur.

The EAPI watchdog timer ***may*** support two stages. If the watchdog is not triggered within the event timeout a NMI, IRQ, or hardware output will be generated. Then the reset timeout becomes active. If the watchdog timer is not triggered within the reset timeout a reset will be generated.

Initial Timing



Timing after EApiWDogTrigger



Stage A

Watchdog is started.

Stage B

Initial Delay Period is exhausted.

Stage C/F

Event is triggered, NMI, IRQ, or PIN is Triggered.
To Allow for possible Software Recovery.

Stage D/G

System is reset.

Stage E

Watchdog is Triggered.

EApiWDogTrigger/EApiWDogStop Must be called before Stage C/F to prevent event from being generated.

EApiWDogTrigger/EApiWDogStop Must be called before Stage D/G to prevent The system from being reset.

8.1 EApiWDogGetCap

```
uint32_t
EAPI_CALLTYPE
EApiWDogGetCap(
    __OUTOPT  uint32_t *pMaxDelay ,
                        /* Max. supported delay in msec */
    __OUTOPT  uint32_t *pMaxEventTimeout ,
                        /* Max. supported event timeout
                           in  msec, 0 == Unsupported  */
    __OUTOPT  uint32_t *pMaxResetTimeout
                        /* Max. supported reset timeout
                           in msec  */
    );
```

FUNC_DEF 17: EApiWDogStart

8.1.1 Description

Get the capabilities of the watchdog timer.

8.1.2 Parameters

pMaxDelay

__OUTOPT Pointer to a buffer that receives maximum supported initial delay time of the watchdog timer in miliseconds.

pMaxEventTimeout

__OUTOPT Pointer to a buffer that receives maximum supported event timeout of the watchdog timer in miliseconds.

pMaxResetTimeout

__OUTOPT Pointer to a buffer that receives maximum supported event timeout of the watchdog timer in miliseconds.

8.1.3 Return Status Codes

Condition	Return Value
Library Uninitialized	***EAPI_STATUS_NOT_INITIALIZED***
Unsupported	***EAPI_STATUS_UNSUPPORTED***
pMaxDelay==NULL&& pMaxResetTimeout==NULL&& pMaxEventTimeout==NULL	***EAPI_STATUS_INVALID_PARAMETER***
...	see 2.3 Status Codes on page 16
Success	***EAPI_STATUS_SUCCESS***

8.2 EApiWDogStart

```
uint32_t
EAPI_CALLTYPE
EApiWDogStart(
    __IN  uint32_t delay   ,    /* Delay in msec */
    __IN  uint32_t EventTimeout /* Timeout in msec  */
    __IN  uint32_t ResetTimeout /* Reset in msec  */
    );
```

FUNC_DEF 18: EApiWDogStart

8.2.1 Description

Start the watchdog timer and set the parameters. To adjust the parameters, the watchdog must be stopped via ***EApiWDogStop*** and then ***EApiWDogStart*** must be called again with the new values.

If the hardware implementation of the watchdog timer does not allow to set exactly the selected timing, the EAPI ***shall*** select the next possible longer timing.

8.2.2 Parameters

delay

__IN Initial delay for the watchdog timer in milliseconds.

The first trigger must happen within (***delay*** + **EventTimeout)** milliseconds, of calling ***EApiWDogStart***.

EventTimeout

__IN Watchdog timeout interval in milliseconds to trigger an event.

ResetTimeout

__IN Watchdog timeout interval in milliseconds to trigger a reset.

8.2.3 Return Status Codes

Condition	Return Value
Library Uninitialized	***EAPI_STATUS_NOT_INITIALIZED***
Unsupported	***EAPI_STATUS_UNSUPPORTED***
Delay>pMaxDelay	**EAPI_STATUS_INVALID_PARAMETER**
EventTimeout>pMaxEventTimeout	**EAPI_STATUS_INVALID_PARAMETER**
ResetTimeout>pMaxResetTimeout	**EAPI_STATUS_INVALID_PARAMETER**
Watchdog already started	***EAPI_STATUS_RUNNING***
...	see 2.3 Status Codes on page 16
Success	***EAPI_STATUS_SUCCESS***

8.3 EApiWDogTrigger

```
uint32_t
EAPI_CALLTYPE
EApiWDogTrigger(void);
```

FUNC_DEF 19: EApiWDogTrigger

8.3.1 Description

Trigger the watchdog timer.

8.3.2 Parameters

None.

8.3.3 Return Status Codes

Condition	Return Value
Library Uninitialized	***EAPI_STATUS_NOT_INITIALIZED***
Unsupported	***EAPI_STATUS_UNSUPPORTED***
Not Started	***EAPI_STATUS_ERROR***
...	see 2.3 Status Codes on page 16
Success	***EAPI_STATUS_SUCCESS***

8.4 EApiWDogStop

```
uint32_t
EAPI_CALLTYPE
EApiWDogStop(void);
```

FUNC_DEF 20: EApiWDogStop

8.4.1 Description

Stops the operation of the watchdog timer.

8.4.2 Parameters

None.

8.4.3 Return Status Codes

Condition	Return Value
Library Uninitialized	***EAPI_STATUS_NOT_INITIALIZED***
Unsupported	***EAPI_STATUS_UNSUPPORTED***
…	see 2.3 Status Codes on page 16
Success	***EAPI_STATUS_SUCCESS***

9 GPIO Functions

COM Express specifies pins for general purpose I/Os. The EAPI provides a set of functions to control these hardware GPIO pins.

9.1 Common Parameters

9.1.1 GPIO Ids

The EAPI defines 4 general purpose inputs and 4 general purpose outputs. The possible configuration as GPIO is also supported. Additional vendor or platform specific IDs are possible (see section 11 'Vendor Specific IDs' on page 76)

The EAPI library defines two different mechanisms to address GPIO ports.

Single port addressing

Example for GPIO single port addressing EApiGPIOGetLevel

```
                        EAPI_ID_GPIO_GPIO03   EAPI_ID_GPIO_GPIO02
  Hardware GPIO Levels  0 0 0 0 0 0 1 1       0 0 0 0 0 0 1 1
  Bitmask                 0 0 0 1                 0 0 0 1
  Resulting pLevel        0 0 0 0                 0 0 0 0

                        EAPI_ID_GPIO_GPIO01   EAPI_ID_GPIO_GPIO00
  Hardware GPIO Levels  0 0 0 0 0 0 1 1       0 0 0 0 0 0 1 1
  Bitmask                 0 0 0 1                 0 0 0 1
  Resulting pLevel        0 0 0 1                 0 0 0 1
```

Example for GPIO single port addressing EApiGPIOSetLevel

```
                          EAPI_ID_GPIO_GPIO07       EAPI_ID_GPIO_GPIO05
  Initial H/W GPIO Level        0 0 1 1 1 1 1 1       0 0 1 1 1 1 1 1
  Bitmask                   0 0 0 1                 0 0 0 1
  Level                     0 0 0 1                 0 0 0 0
  Resulting H/W GPIO Level      1 0 1 1 1 1 1 1       0 0 0 1 1 1 1 1
```

Bank addressing

Example for GPIO bit mask addressing EApiGPIOGetLevel

```
Sample Logic:
  pLevel=HardwareLevel&Bitmask

Truth tables:
 - Parallel addressing (Bitmask)
                        EAPI_ID_GPIO_BANK00
  Hardware GPIO Level   0 0 0 0 0 0 1 1
  Bitmask               0 0 0 0 0 1 0 1
  Resulting pLevel      0 0 0 0 0 0 0 1
```

Example for GPIO bit mask addressing EApiGPIOSetLevel

```
Sample Logic:
  HardwareLevel=(HardwareLevel&~Bitmask)|(Bitmask&Level)

Truth table:
 - Parallel addressing (Bitmask)
                          EAPI_ID_GPIO_BANK00
  Initial H/W GPIO Level   0 1 0 1 0 0 0 0       0 1 0 1 0 0 0 0
  Bitmask                  0 0 1 1 0 0 0 0       0 0 1 1 0 0 0 0
  Level                    0 0 0 0 0 0 0 0       1 1 1 1 0 0 0 0
  Resulting H/W GPIO Level 0 1 0 0 0 0 0 0       0 1 1 1 0 0 0 0
```

Single GPIO addressing

Each GPIO pin can be addressed individually. Please refer to the platform specific header file (COM Express: EApiCOM0.h) for the detailed pin assignment.

Individual GPIO Ids	Description
EAPI_GPIO_ID0	'GPIO 0' Bit mapped to Bit 0

Individual GPIO Ids	Description
EAPI_GPIO_ID1	'GPIO 1' Bit mapped to Bit 0
EAPI_GPIO_ID2	'GPIO 2' Bit mapped to Bit 0
EAPI_GPIO_ID3	'GPIO 3' Bit mapped to Bit 0
EAPI_GPIO_ID4	'GPIO 4' Bit mapped to Bit 0
EAPI_GPIO_ID5	'GPIO 5' Bit mapped to Bit 0
EAPI_GPIO_ID6	'GPIO 6' Bit mapped to Bit 0
EAPI_GPIO_ID7	'GPIO 7' Bit mapped to Bit 0

Parallel GPIO addressing

A group of selected GPIO pins can be addressed simultaneously.

Multiple GPIO Ids	Description
EAPI_ID_GPIO_BANK00	GPIO 0-31 Bit mapped to Bit 0-31

9.1.2 Bit-mask Bit States

Name	Description
EAPI_GPIO_BITMASK_SELECT	Used to specify that the Specific GPIO is selected
EAPI_GPIO_BITMASK_NOSELECT	Used to specify that the Specific GPIO is not selected, and should be ignored.

9.1.3 Level Bit States

Name	Description
EAPI_GPIO_LOW	Used to specify/signify that the Specific GPIO is low(not asserted).
EAPI_GPIO_HIGH	Used to specify/signify that the Specific GPIO is high(asserted).

9.1.4 Direction Bit States

Name	Description
EAPI_GPIO_INPUT	Used to specify/signify that the Specific GPIO is in input mode.
EAPI_GPIO_OUTPUT	Used to specify/signify that the Specific GPIO is in output mode.

9.2 EApiGPIOGetDirectionCaps

```
uint32_t
EAPI_CALLTYPE
EApiGPIOGetDirectionCaps(
    __IN     uint32_t Id         , /* GPIO Id */
    __OUTOPT uint32_t *pInputs   , /* Supported GPIO Input
                                    * Bit Mask
                                    */
    __OUTOPT uint32_t *pOutputs    /* Supported GPIO Output
                                    * Bit Mask
                                    */
    );
```

FUNC_DEF 21: EApiGPIOGetDirectionCaps

9.2.1 Description

Reads the capabilities of the current GPIO implementation from the selected GPIO interface.

The ports where both input and output bit masks are 1 are GPIOs. The direction of this ports can be configured by ***EApiGPIOSetDirection***

9.2.2 Parameters

Id

__IN See '9.1.1 GPIO Ids' on page 47

pInputs

__OUTOPT Pointer to a buffer that receives the bit mask of the supported inputs.

pOutputs

__OUTOPT Pointer to a buffer that receives the bit mask of the supported inputs.

9.2.3 Return Status Codes

Condition	Return Value
Library Uninitialized	***EAPI_STATUS_NOT_INITIALIZED***
((pInputs==NULL)&&(pOutputs==NULL))	***EAPI_STATUS_INVALID_PARAMETER***
Unsupported ID	***EAPI_STATUS_UNSUPPORTED***
Not Started	***EAPI_STATUS_ERROR***
...	see 2.3 Status Codes on page 16
Success	***EAPI_STATUS_SUCCESS***

9.3 EApiGPIOGetDirection

```
uint32_t
EAPI_CALLTYPE
EApiGPIOGetDirection(
    __IN  uint32_t Id           , /* GPIO Id */
    __IN  uint32_t Bitmask      , /* Bit mask of Affected
                                   * Bits
                                   */
    __OUT uint32_t *pDirection    /* Current Direction */
    );
```

FUNC_DEF 22: EApiGPIOGetDirection

9.3.1 Description

Reads the current configuration of the selected GPIO ports.

9.3.2 Parameters

Id

__IN See '9.1.1 GPIO Ids' on page 47

Bitmask

__IN Bit mask.

Value	Description
EAPI_GPIO_BITMASK_NOSELECT	Do not use the selected GPIO port for this operation.
EAPI_GPIO_BITMASK_SELECT	Use the selected GPIO port for this operation.

See also 'Bitmask' on page 52.

pDirection

__OUT Pointer to a buffer that receives the direction of the selected GPIO ports.

Value	Description
EAPI_GPIO_INPUT	Selected GPIO port is an input.
EAPI_GPIO_OUTPUT	Selected GPIO port is an output.

9.3.3 Return Status Codes

Condition	Return Value
Library Uninitialized	***EAPI_STATUS_NOT_INITIALIZED***
Bitmask==0	***EAPI_STATUS_INVALID_PARAMETER***
Unsupported ID	***EAPI_STATUS_UNSUPPORTED***
(Bitmask&~(pInputs|pOutputs))	***EAPI_STATUS_INVALID_BITMASK***
Success	***EAPI_STATUS_SUCCESS***

See also '2.3 Status Codes' on page 16.

9.4 EApiGPIOSetDirection

```
uint32_t
EAPI_CALLTYPE
EApiGPIOSetDirection(
    __IN  uint32_t Id           , /* GPIO Id */
    __IN  uint32_t Bitmask      , /* Bit mask of Affected
                                   * Bits
                                   */
    __IN  uint32_t Direction      /* Direction */
    );
```

FUNC_DEF 23: EApiGPIOSetDirection

9.4.1 Description

Sets the configuration for the selected GPIO ports.

9.4.2 Parameters

Id

__IN See '9.1.1 GPIO Ids' on page 47

Bitmask

__IN Bit mask. Only the ports with the level ***EAPI_GPIO_HIGH*** are processed.

Value	Description
EAPI_GPIO_BITMASK_NOSELECT	Do not use the selected GPIO port for this operation.
EAPI_GPIO_BITMASK_SELECT	Use the selected GPIO port for this operation.

See also 'Bitmask' on page 53.

Direction

__IN Sets the direction of the selected GPIO ports. Bits with the value **EAPI_GPIO_INPUT** are inputs, bits with ***EAPI_GPIO_OUTPUT*** are outputs.

Value	Description
EAPI_GPIO_INPUT	Selected GPIO port is an input.
EAPI_GPIO_OUTPUT	Selected GPIO port is an output.

9.4.3 Return Status Codes

Condition	Return Value
Library Uninitialized	***EAPI_STATUS_NOT_INITIALIZED***
Bitmask==0	***EAPI_STATUS_INVALID_PARAMETER***
Unsupported ID	***EAPI_STATUS_UNSUPPORTED***
(Bitmask&~(pInputs|pOutputs))	***EAPI_STATUS_INVALID_BITMASK***
(Bitmask&Direction)&~pInputs	***EAPI_STATUS_INVALID_DIRECTION***
(Bitmask&~Direction)&~pOutputs	***EAPI_STATUS_INVALID_DIRECTION***
Not Started	***EAPI_STATUS_ERROR***
...	see 2.3 Status Codes on page 16
Success	***EAPI_STATUS_SUCCESS***

9.5 EApiGPIOGetLevel

```
uint32_t
EAPI_CALLTYPE
EApiGPIOGetLevel(
    __IN  uint32_t Id           , /* GPIO Id */
    __IN  uint32_t Bitmask      , /* Bit mask of Affected
                                   * Bits
                                   */
    __OUT uint32_t *pLevel        /* Current Level */
    );
```

FUNC_DEF 24: EApiGPIOGetLevel

9.5.1 Description

Read the from GPIO ports.

9.5.2 Parameters

Id

__IN See '9.1.1 GPIO Ids' on page 47

Bitmask

__IN Bit mask. Only selected bits are returned. Unselected bits return ***EAPI_GPIO_LOW.***

Value	Description
EAPI_GPIO_BITMASK_NOSELECT	Do not use the selected GPIO port for this operation.
EAPI_GPIO_BITMASK_SELECT	Use the selected GPIO port for this operation.

See examples 'Single port addressing' and 'Bank addressing' on page 47.

pLevel

__OUT Pointer to a buffer that receives the GPIO level. Results can be read on a bit level.

Value	Description
EAPI_GPIO_LOW	Selected GPIO port is low.
EAPI_GPIO_HIGH	Selected GPIO port is high.

9.5.3 Return Status Codes

Condition	Return Value
Library Uninitialized	***EAPI_STATUS_NOT_INITIALIZED***
Bitmask==0	***EAPI_STATUS_INVALID_PARAMETER***
Unsupported ID	***EAPI_STATUS_UNSUPPORTED***
(Bitmask&~(pInputs|pOutputs))	***EAPI_STATUS_INVALID_BITMASK***
...	***see 2.3 Status Codes on page 16***
Success	***EAPI_STATUS_SUCCESS***

9.6 EApiGPIOSetLevel

```
uint32_t
EAPI_CALLTYPE
EApiGPIOSetLevel(
    __IN  uint32_t Id           , /* GPIO Id */
    __IN  uint32_t Bitmask      , /* Bit mask of Affected
                                   * Bits
                                   */
    __IN  uint32_t Level          /* Level */
    );
```

FUNC_DEF 25: EApiGPIOSetLevel

9.6.1 Description

Write to GPIO ports. Depending on the hardware implementation writing multiple GPIO ports with the bit mask option does not guarantee a time synchronous change of the output levels.

9.6.2 Parameters

Id

__IN See '9.1.1 GPIO Ids' on page 47

Bitmask

__IN Value for a bit mask. Only selected bits are changed. Unselected bits remain unchanged.

Value	Description
EAPI_GPIO_BITMASK_NOSELECT	Do not use the selected GPIO port for this operation.
EAPI_GPIO_BITMASK_SELECT	Use the selected GPIO port for this operation.

See examples 'Single port addressing' and 'Bank addressing' on page 47.

Level

__IN Input level of the selected GPIO port. Output for single ports is on a bit level.

Value	Description
EAPI_GPIO_LOW	Set selected GPIO port to low.
EAPI_GPIO_HIGH	Set selected GPIO port to high.

9.6.3 Return Status Codes

Condition	Return Value
Library Uninitialized	***EAPI_STATUS_NOT_INITIALIZED***
Bitmask==0	***EAPI_STATUS_INVALID_PARAMETER***
Unsupported ID	***EAPI_STATUS_UNSUPPORTED***
(Bitmask&~(pInputs|pOutputs))	***EAPI_STATUS_INVALID_BITMASK***
...	***see 2.3 Status Codes on page 16***
Success	***EAPI_STATUS_SUCCESS***

10 Header Files

The documentation refers to the following header files.

10.1 EApi.h

```
/*
 *<KHeader>
 *+=========================================================================
 *I  Project Name: EApiDK Embedded Application Development Kit
 *+=========================================================================
 *I  $HeadURL: https://eapidk.svn.sourceforge.net/svnroot/eapidk/trunk/include/EApi.h $
 *+=========================================================================
 *I   Copyright: Copyright (c) 2009-2010, PICMG
 *I      Author: John Kearney,                John.Kearney@kontron.com
 *I
 *I     License: All rights reserved. This program and the accompanying
 *I              materials are licensed and made available under the
 *I              terms and conditions of the BSD License which
 *I              accompanies this distribution. The full text of the
 *I              license may be found at
 *I              http://opensource.org/licenses/bsd-license.php
 *I
 *I              THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "
 *I              AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF
 *I              ANY KIND, EITHER EXPRESS OR IMPLIED.
 *I
 *I Description: Auto Created for EApi.h
 *I
 *+-------------------------------------------------------------------------
 *I
 *I  File Name            : EApi.h
 *I  File Location        : include
 *I  Last committed       : $Revision: 74 $
 *I  Last changed by      : $Author: dethrophes $
 *I  Last changed date    : $Date: 2010-06-23 21:26:50 +0200 (Mi, 23 Jun 2010) $
 *I  ID                   : $Id: EApi.h 74 2010-06-23 19:26:50Z dethrophes $
 *I
 *+=========================================================================
 *</KHeader>
 */

#ifndef _EAPI_H_
#define _EAPI_H_

#ifdef __cplusplus
extern "C" {
#endif

/***************************************************************************/

#ifndef __IN
#  define __IN
/* __IN
 * Arg Type        | Characteristics
 * ----------------+---------------------------------------------------
 * Immediate value | Input value that must be specified and is essential
 *                 | to function operation.
 *                 |
 * Pointer         | Valid pointer to initialized buffer/variable.
 *                 |
 */
#endif
#ifndef __INOPT
#  define __INOPT
/* __INOPT
 * Arg Type        | Characteristics
 * ----------------+---------------------------------------------------
 * Pointer         | Valid pointer to initialized buffer/variable, or
 *                 | NULL Pointer.
 *                 | Note: refer to function specification for specifics.
 */
#endif
```

```
#ifndef __OUT
#  define __OUT
/* __OUT
 * Arg Type        | Characteristics
 * ----------------+------------------------------------------------------
 * Pointer         | Valid pointer to destination buffer/variable
 */
#endif
#ifndef __OUTOPT
#  define __OUTOPT

/* __OUTOPT
 * Arg Type        | Characteristics
 * ----------------+------------------------------------------------------
 * Pointer         | Valid pointer to destination buffer/variable, or
 *                 | NULL Pointer.
 *                 | Note: refer to function specification for specifics.
 */
#endif
#ifndef __INOUT
#  define __INOUT
/* __INOUT
 * Arg Type        | Characteristics
 * ----------------+------------------------------------------------------
 * Pointer         | Valid pointer to initialized buffer/variable
 *                 | Contents of buffer/variable updated before return.
 */
#endif
#ifndef EAPI_CALLTYPE
#  define EAPI_CALLTYPE
#endif
#ifndef EAPI_UINT32_C
#  define EAPI_UINT8_C(x)   ((uint8_t)(x))
#  define EAPI_UINT16_C(x)  ((uint16_t)(x))
#  define EAPI_UINT32_C(x)  ((uint32_t)(x))
#endif
/*****************************************************************************
 * All version numbers
 * +=========+================================+
 * | Bits    | Descriptions                   |
 * +=========+================================+
 * | 24 - 31 | Version Number                 |
 * +---------+--------------------------------+
 * | 16 - 23 | Revision Number                |
 * +---------+--------------------------------+
 * | 0  - 15 | Build Number                   |
 * +=========+================================+
 */
#define EAPI_VER_MASK_VER     EAPI_UINT32_C(0xFF000000)
#define EAPI_VER_MASK_REV     EAPI_UINT32_C(0x00FF0000)
#define EAPI_VER_MASK_BUILD   EAPI_UINT32_C(0x0000FFFF)
#define EAPI_VER_GET_VER(x)   EAPI_UINT8_C (((x)>>24)&UINT8_MAX )
#define EAPI_VER_GET_REV(x)   EAPI_UINT8_C (((x)>>16)&UINT8_MAX )
#define EAPI_VER_GET_BUILD(x) EAPI_UINT16_C(((x)>> 0)&UINT16_MAX)
#define EAPI_VER_CREATE(Version,Revision,Build) (\
                     EAPI_UINT32_C(\
                       (((Version )&UINT8_MAX )<<24)|\
                       (((Revision)&UINT8_MAX )<<16)|\
                       (((Build   )&UINT16_MAX)<< 0) \
                     )\
                   )

/* Embedded API Standard Revision */
#define EAPI_VER      1
#define EAPI_REVISION 0
#define EAPI_VERSION EAPI_VER_CREATE(EAPI_VER, EAPI_REVISION, 0)


/*****************************************************************************
 */


/* Embedded Application System Interface */

/*
 *  EApi Types
 */
```

```
typedef uint32_t EApiStatus_t;
typedef uint32_t EApiId_t;

/*
 *
 *
 *  S T A T U S   C O D E S
 *
 *
 */
/* Description
 *   The EAPI library is not yet or unsuccessfully initialized.
 *   EApiLibInitialize needs to be called prior to the first access of any
 *   other EAPI function.
 * Actions
 *   Call EApiLibInitialize..
 */
#define EAPI_STATUS_NOT_INITIALIZED   EAPI_UINT32_C(0xFFFFFFFF)

/* Description
 *   Library is initialized.
 * Actions
 *   none.
 */
#define EAPI_STATUS_INITIALIZED       EAPI_UINT32_C(0xFFFFFFFE)

/* Description
 *   Memory Allocation Error.
 * Actions
 *   Free memory and try again..
 */
#define EAPI_STATUS_ALLOC_ERROR       EAPI_UINT32_C(0xFFFFFFFD)

/* Description
 *   Time out in driver. This is Normally caused by hardware/software
 *   semaphore timeout.
 * Actions
 *   Retry.
 */
#define EAPI_STATUS_DRIVER_TIMEOUT    EAPI_UINT32_C(0xFFFFFFFC)

/* Description
 *  One or more of the EAPI function call parameters are out of the
 *  defined range.
 *
 *  Possible Reasons include be
 *  NULL Pointer
 *  Invalid Offset
 *  Invalid Length
 *  Undefined Value
 *
 *   Storage Write
 *    Incorrectly Aligned Offset
 *    Invalid Write Length
 * Actions
 *   Verify Function Parameters.
 */
#define EAPI_STATUS_INVALID_PARAMETER EAPI_UINT32_C(0xFFFFFEFF)

/* Description
 *   The Block Alignment is incorrect.
 * Actions
 *   Use pInputs and pOutputs to correctly select input and outputs.
 */
#define EAPI_STATUS_INVALID_BLOCK_ALIGNMENT EAPI_UINT32_C(0xFFFFFEFE)

/* Description
 *   This means that the Block length is too long.
 * Actions
 *   Use Alignment Capabilities information to correctly align write access.
 */
#define EAPI_STATUS_INVALID_BLOCK_LENGTH    EAPI_UINT32_C(0xFFFFFEFD)

/* Description
 *   The current Direction Argument attempts to set GPIOs to a unsupported
 *   directions. I.E. Setting GPI to Output.
 * Actions
```

```
 *   Use pInputs and pOutputs to correctly select input and outputs.
 */
#define EAPI_STATUS_INVALID_DIRECTION       EAPI_UINT32_C(0xFFFFFEFC)

/* Description
 *   The Bitmask Selects bits/GPIOs which are not supported for the current ID.
 * Actions
 *   Use pInputs and pOutputs to probe supported bits..
 */
#define EAPI_STATUS_INVALID_BITMASK         EAPI_UINT32_C(0xFFFFFEFB)

/* Description
 *   Watchdog timer already started.
 * Actions
 *   Call EApiWDogStop, before retrying.
 */
#define EAPI_STATUS_RUNNING                 EAPI_UINT32_C(0xFFFFFEFA)

/* Description
 *   This function or ID is not supported at the actual hardware environment.
 * Actions
 *   none.
 */
#define EAPI_STATUS_UNSUPPORTED       EAPI_UINT32_C(0xFFFFFCFF)

/* Description
 *   I2C Device Error
 *   No Acknowledge For Device Address, 7Bit Address Only
 *   10Bit Address may cause Write error if 2 10Bit addressed devices
 *   present on the bus.
 * Actions
 *   none.
 */
#define EAPI_STATUS_NOT_FOUND         EAPI_UINT32_C(0xFFFFFBFF)

/* Description
 *   I2C Time-out
 *   Device Clock stretching time-out, Clock pulled low by device
 *   for too long
 * Actions
 *   none.
 */
#define EAPI_STATUS_TIMEOUT           EAPI_UINT32_C(0xFFFFFBFE)

/* Description
 *   EApi  I2C functions specific. The addressed I2C bus is busy or there
 *   is a bus collision.
 *   The I2C bus is in use. Either CLK or DAT are low.
 *   Arbitration loss or bus Collision, data remains low when writing a 1
 * Actions
 *   Retry.
 */
#define EAPI_STATUS_BUSY_COLLISION    EAPI_UINT32_C(0xFFFFFBFD)

/* Description
 *   I2C Read Error
 *    Not Possible to detect.
 *   Storage Read Error
 *    ....
 * Actions
 *   Retry.
 */
#define EAPI_STATUS_READ_ERROR        EAPI_UINT32_C(0xFFFFFAFf)

/* Description
 *   I2C Write Error
 *     No Acknowledge received after writing any Byte after the First Address
 *     Byte.
 *     Can be caused by
 *     unsupported Device Command/Index
 *     Ext Command/Index used on Standard Command/Index Device
 *     10Bit Address Device Not Present
 *   Storage Write Error
 *    ...
 * Actions
 *   Retry.
 */
```

```
#define EAPI_STATUS_WRITE_ERROR       EAPI_UINT32_C(0xFFFFFAFE)


/* Description
 *   The amount of available data exceeds the buffer size.
 *   Storage buffer overflow was prevented. Read count was larger then
 *   the defined buffer length.
 *   Read Count > Buffer Length
 * Actions
 *   Either increase the buffer size or reduce the block length.
 */
#define EAPI_STATUS_MORE_DATA         EAPI_UINT32_C(0xFFFFF9FF)

/* Description
 *   Generic error message. No further error details are available.
 * Actions
 *   none.
 */
#define EAPI_STATUS_ERROR             EAPI_UINT32_C(0xFFFFF0FF)

/* Description
 *   The operation was successful.
 * Actions
 *   none.
 */
#define EAPI_STATUS_SUCCESS           EAPI_UINT32_C(0)
#define EAPI_TEST_SUCCESS(x)        (!(x))

/* Library */
/*
 * EApiLibInitialize
 *
 * Condition                              | Return Values
 * ---------------------------------------+------------------------------
 * Library Already initialized            | EAPI_STATUS_INITIALIZED
 * Common Error                           | Common Error Code
 * Else                                   | EAPI_STATUS_SUCCESS
 */
EApiStatus_t
EAPI_CALLTYPE
EApiLibInitialize(void)   ; /* Should be called before
                             * calling any other API
                             * function is called
                             */
/*
 * EApiLibUnInitialize
 *
 * Condition                              | Return Values
 * ---------------------------------------+------------------------------
 * Library Uninitialized                  | EAPI_STATUS_NOT_INITIALIZED
 * Common Error                           | Common Error Code
 * Else                                   | EAPI_STATUS_SUCCESS
 */
EApiStatus_t
EAPI_CALLTYPE
EApiLibUnInitialize(void) ; /* Should be called before
                             * program exit
                             */


/*
 *
 * Plug and Play Identifiers
 *
 *
 *
 */
/*
 * The following MACROS are for Creating OEM IDs
 * OEM ID macros should be named
 * EAPI_[PNPID]_ID_[TYPE]_[NAME]
 * E.G.
 * EAPI_PMG_ID_BOARD_CUSTOMERID
 */
#define EAPI_BYTE_SWAP_W(a) EAPI_UINT16_C(\
      (((a)<<8)&0xFF00)|\
      (((a)>>8)&0x00FF) \
```

```
      )

#define EAPI_CREATE_PNPID(a, b, c) \
  EAPI_BYTE_SWAP_W((((  a -'A'+1)&0x1F)<<10)|\
              ((( b -'A'+1)&0x1F)<< 5)|\
              ((( c -'A'+1)&0x1F)<< 0) \
             )
#define EAPI_CREATE_CUST_ID(a, b, c, Id)\
      EAPI_UINT32_C((0xF<<28)|(EAPI_CREATE_PNPID(a, b, c)<<12)|(Id&0xFFF))

#define EAPI_PNPID_PICMG EAPI_CREATE_PNPID('P', 'M', 'G')  /* PICMG Should
                                                            * register this.
                                                            */

/*
 *
 *      B O A R D   I N F O M A T I O N   S T R I N G S
 *
 */
/* IDS */
#define EAPI_ID_BOARD_MANUFACTURER_STR     EAPI_UINT32_C(0) /* Board Manufacturer
                                                             * Name String
                                                             */
#define EAPI_ID_BOARD_NAME_STR             EAPI_UINT32_C(1) /* Board Name String */
#define EAPI_ID_BOARD_REVISION_STR         EAPI_UINT32_C(2) /* Board Name String */
#define EAPI_ID_BOARD_SERIAL_STR           EAPI_UINT32_C(3) /* Board Serial
                                                             * Number String
                                                             */
#define EAPI_ID_BOARD_BIOS_REVISION_STR    EAPI_UINT32_C(4) /* Board Bios Revision
                                                             * String
                                                             */
#define EAPI_ID_BOARD_HW_REVISION_STR      EAPI_UINT32_C(5) /* Board Hardware
                                                             * Revision String
                                                             */
#define EAPI_ID_BOARD_PLATFORM_TYPE_STR    EAPI_UINT32_C(6) /* Platform ID
                                                             * (ETX, COM Express,
                                                             * etc...)
                                                             */


/*
 * EApiBoardGetStringA
 *
 * Condition                                | Return Values
 * -----------------------------------------+------------------------------
 * Library Uninitialized                    | EAPI_STATUS_NOT_INITIALIZED
 * pBufLen==NULL                            | EAPI_STATUS_INVALID_PARAMETER
 * pBufLen!=NULL&&*pBufLen&&pBuffer==NULL   | EAPI_STATUS_INVALID_PARAMETER
 * unknown Id                               | EAPI_STATUS_UNSUPPORTED
 * strlen(Id)+1>*pBufLen                    | EAPI_STATUS_MORE_DATA
 * Common Error                             | Common Error Code
 * Else                                     | EAPI_STATUS_SUCCESS
 */
EApiStatus_t
EAPI_CALLTYPE
EApiBoardGetStringA(
    __IN      EApiId_t  Id      , /* Name Id */
    __OUT     char     *pBuffer , /* Destination pBuffer */
    __INOUT   uint32_t *pBufLen   /* pBuffer Length */
    );


/*
 *
 *      B O A R D   I N F O M A T I O N   V A L U E S
 *
 */
/* IDS */
#define EAPI_ID_GET_EAPI_SPEC_VERSION         EAPI_UINT32_C(0) /* EAPI Specification
                                                                * Revision I.E. The
                                                                * EAPI Spec Version
                                                                * Bits 31-24, Revision
                                                                * 23-16, 15-0 always 0
                                                                * Used to implement
                                                                * this interface
                                                                */
```

```
#define EAPI_ID_BOARD_BOOT_COUNTER_VAL        EAPI_UINT32_C(1) /* Units = Boots */
#define EAPI_ID_BOARD_RUNNING_TIME_METER_VAL  EAPI_UINT32_C(2) /* Units = Minutes */
#define EAPI_ID_BOARD_PNPID_VAL               EAPI_UINT32_C(3) /* Encoded PNP ID
                                                                * Format
                                                                * (Compressed ASCII)
                                                                */
#define EAPI_ID_BOARD_PLATFORM_REV_VAL        EAPI_UINT32_C(4) /* Platform Revision
                                                                * I.E. The PICMG Spec
                                                                * Version Bits 31-24,
                                                                * Revision 23-16,
                                                                * 15-0 always 0
                                                                */

#define EAPI_ID_BOARD_DRIVER_VERSION_VAL  EAPI_UINT32_C(0x10000) /* Vendor Specific
                                                                  * (Optional)
                                                                  */
#define EAPI_ID_BOARD_LIB_VERSION_VAL     EAPI_UINT32_C(0x10001) /* Vendor Specific
                                                                  * (Optional)
                                                                  */

#define EAPI_ID_HWMON_CPU_TEMP            EAPI_UINT32_C(0x20000) /* 0.1 Kelvins */
#define EAPI_ID_HWMON_CHIPSET_TEMP        EAPI_UINT32_C(0x20001) /* 0.1 Kelvins */
#define EAPI_ID_HWMON_SYSTEM_TEMP         EAPI_UINT32_C(0x20002) /* 0.1 Kelvins */

#define EAPI_KELVINS_OFFSET 2731
#define EAPI_ENCODE_CELCIUS(Celsius) EAPI_UINT32_C((((Celsius)*10))+EAPI_KELVINS_OFFSET)
#define EAPI_DECODE_CELCIUS(Celsius) ((Celsius)- EAPI_KELVINS_OFFSET)/10

#define EAPI_ID_HWMON_VOLTAGE_VCORE       EAPI_UINT32_C(0x21004) /* millivolts */
#define EAPI_ID_HWMON_VOLTAGE_2V5         EAPI_UINT32_C(0x21008) /* millivolts */
#define EAPI_ID_HWMON_VOLTAGE_3V3         EAPI_UINT32_C(0x2100C) /* millivolts */
#define EAPI_ID_HWMON_VOLTAGE_VBAT        EAPI_UINT32_C(0x21010) /* millivolts */
#define EAPI_ID_HWMON_VOLTAGE_5V          EAPI_UINT32_C(0x21014) /* millivolts */
#define EAPI_ID_HWMON_VOLTAGE_5VSB        EAPI_UINT32_C(0x21018) /* millivolts */
#define EAPI_ID_HWMON_VOLTAGE_12V         EAPI_UINT32_C(0x2101C) /* millivolts */

#define EAPI_ID_HWMON_FAN_CPU             EAPI_UINT32_C(0x22000) /* RPM */
#define EAPI_ID_HWMON_FAN_SYSTEM          EAPI_UINT32_C(0x22001) /* RPM */

/*
 * EApiBoardGetValue
 *
 * Condition                              | Return Values
 * ---------------------------------------+------------------------------
 * Library Uninitialized                  | EAPI_STATUS_NOT_INITIALIZED
 * pValue==NULL                           | EAPI_STATUS_INVALID_PARAMETER
 * unknown Id                             | EAPI_STATUS_UNSUPPORTED
 * Common Error                           | Common Error Code
 * Else                                   | EAPI_STATUS_SUCCESS
 */
EApiStatus_t
EAPI_CALLTYPE
EApiBoardGetValue(
    __IN  EApiId_t  Id      , /* Value Id */
    __OUT uint32_t *pValue    /* Return Value */
    );



/*
 *
 *      B A C K L I G H T
 *
 */
/* IDS */
#define EAPI_ID_BACKLIGHT_1           EAPI_UINT32_C(0)
#define EAPI_ID_BACKLIGHT_2           EAPI_UINT32_C(1)
#define EAPI_ID_BACKLIGHT_3           EAPI_UINT32_C(2)

/* Backlight Values */
#define EAPI_BACKLIGHT_SET_ON         EAPI_UINT32_C(0)
#define EAPI_BACKLIGHT_SET_OFF        EAPI_UINT32_C(0xFFFFFFFF)
#define EAPI_BACKLIGHT_SET_DIMEST     EAPI_UINT32_C(0)
#define EAPI_BACKLIGHT_SET_BRIGHTEST  EAPI_UINT32_C(255)
/*
 * EApiVgaGetBacklightEnable
 *
```

```
 * Condition                              | Return Values
 * ---------------------------------------+-------------------------------
 * Library Uninitialized                  | EAPI_STATUS_NOT_INITIALIZED
 * pEnable==NULL                          | EAPI_STATUS_INVALID_PARAMETER
 * unknown Id                             | EAPI_STATUS_UNSUPPORTED
 * Common Error                           | Common Error Code
 * Else                                   | EAPI_STATUS_SUCCESS
 */
EApiStatus_t
EAPI_CALLTYPE
EApiVgaGetBacklightEnable(
    __IN  EApiId_t  Id      , /* Backlight Id */
    __OUT uint32_t *pEnable   /* Backlight Enable */
    );
/*
 * EApiVgaSetBacklightEnable
 *
 * Condition                              | Return Values
 * ---------------------------------------+-------------------------------
 * Library Uninitialized                  | EAPI_STATUS_NOT_INITIALIZED
 * Enable!=EAPI_BACKLIGHT_SET_ON&&
 * Enable!=EAPI_BACKLIGHT_SET_OFF         | EAPI_STATUS_INVALID_PARAMETER
 * unknown Id                             | EAPI_STATUS_UNSUPPORTED
 * Common Error                           | Common Error Code
 * Else                                   | EAPI_STATUS_SUCCESS
 */
EApiStatus_t
EAPI_CALLTYPE
EApiVgaSetBacklightEnable(
    __IN  EApiId_t  Id      , /* Backlight Id */
    __IN  uint32_t  Enable    /* Backlight Enable */
    );
/*
 * EApiVgaGetBacklightBrightness
 *
 * Condition                              | Return Values
 * ---------------------------------------+-------------------------------
 * Library Uninitialized                  | EAPI_STATUS_NOT_INITIALIZED
 * pBright==NULL                          | EAPI_STATUS_INVALID_PARAMETER
 * unknown Id                             | EAPI_STATUS_UNSUPPORTED
 * Common Error                           | Common Error Code
 * Else                                   | EAPI_STATUS_SUCCESS
 */
EApiStatus_t
EAPI_CALLTYPE
EApiVgaGetBacklightBrightness(
    __IN  EApiId_t  Id      , /* Backlight Id */
    __OUT uint32_t *pBright   /* Backlight Brightness */
    );
/*
 * EApiVgaSetBacklightBrightness
 *
 * Condition                              | Return Values
 * ---------------------------------------+-------------------------------
 * Library Uninitialized                  | EAPI_STATUS_NOT_INITIALIZED
 * Bright>EAPI_BACKLIGHT_SET_BRIGHTEST    | EAPI_STATUS_INVALID_PARAMETER
 * unknown Id                             | EAPI_STATUS_UNSUPPORTED
 * Common Error                           | Common Error Code
 * Else                                   | EAPI_STATUS_SUCCESS
 */
EApiStatus_t
EAPI_CALLTYPE
EApiVgaSetBacklightBrightness(
    __IN  EApiId_t  Id      , /* Backlight Id */
    __IN  uint32_t  Bright    /* Backlight Brightness */
    );


/*
 *
 *      S T O R A G E
 *
 */
/* IDs */
#define EAPI_ID_STORAGE_STD             EAPI_UINT32_C(0)
/* Dummy Example */
#define EAPI_PMG_ID_STORAGE_SAMPLE      EAPI_CREATE_CUST_ID('P', 'M', 'G', 0)
```

```
/*
 * EApiStorageCap
 *
 * Condition                                    | Return Values
 * ---------------------------------------------+-----------------------------
 * Library Uninitialized                        | EAPI_STATUS_NOT_INITIALIZED
 * ((pStorageSize==NULL)&&(pBlockLength==NULL)) | EAPI_STATUS_INVALID_PARAMETER
 * Unsupported Id                               | EAPI_STATUS_UNSUPPORTED
 * Common Error                                 | Common Error Code
 * Else                                         | EAPI_STATUS_SUCCESS
 */
EApiStatus_t
EAPI_CALLTYPE
EApiStorageCap(
    __IN  EApiId_t  Id           , /* Storage Area Id */
    __OUT uint32_t  *pStorageSize , /* Total */
    __OUT uint32_t  *pBlockLength   /* Write Block Length
                                     * & Alignment
                                     */
    );
/*
 * EApiStorageAreaRead
 *
 * Condition                              | Return Values
 * ---------------------------------------+-----------------------------
 * Library Uninitialized                  | EAPI_STATUS_NOT_INITIALIZED
 * pBuffer==NULL                          | EAPI_STATUS_INVALID_PARAMETER
 * ByteCnt==0                             | EAPI_STATUS_INVALID_PARAMETER
 * BufLen==0                              | EAPI_STATUS_INVALID_PARAMETER
 * unknown Id                             | EAPI_STATUS_UNSUPPORTED
 * Offset+ByteCnt>pStorageSize            | EAPI_STATUS_INVALID_BLOCK_LENGTH
 * Read Error                             | EAPI_STATUS_READ_ERROR
 * ByteCnt>BufLen                         | EAPI_STATUS_MORE_DATA
 * Common Error                           | Common Error Code
 * Else                                   | EAPI_STATUS_SUCCESS
 */
EApiStatus_t
EAPI_CALLTYPE
EApiStorageAreaRead(
    __IN  EApiId_t  Id      , /* Storage Area Id */
    __IN  uint32_t  Offset  , /* Byte Offset */
    __OUT     void *pBuffer , /* Pointer to Data pBuffer */
    __IN  uint32_t  BufLen  , /* Data pBuffer Size in
                               * bytes
                               */
    __IN  uint32_t  ByteCnt   /* Number of bytes to read */
    );
/*
 * EApiStorageAreaWrite
 *
 * Condition                              | Return Values
 * ---------------------------------------+-----------------------------
 * Library Uninitialized                  | EAPI_STATUS_NOT_INITIALIZED
 * pBuffer==NULL                          | EAPI_STATUS_INVALID_PARAMETER
 * ByteCnt==0                             | EAPI_STATUS_INVALID_PARAMETER
 * unknown Id                             | EAPI_STATUS_UNSUPPORTED
 * Offset%pBlockLength                    | EAPI_STATUS_INVALID_BLOCK_ALIGNMENT
 * ByteCnt%pBlockLength                   | EAPI_STATUS_INVALID_BLOCK_ALIGNMENT
 * Offset+ByteCnt>pStorageSize            | EAPI_STATUS_INVALID_BLOCK_LENGTH
 * Write Error                            | EAPI_STATUS_WRITE_ERROR
 * Common Error                           | Common Error Code
 * Else                                   | EAPI_STATUS_SUCCESS
 */
EApiStatus_t
EAPI_CALLTYPE
EApiStorageAreaWrite(
    __IN  EApiId_t  Id      , /* Storage Area Id */
    __IN  uint32_t  Offset  , /* Byte Offset */
    __IN      void *pBuffer , /* Pointer to Data pBuffer */
    __IN  uint32_t  ByteCnt   /* Number of bytes to write */
    );


/*
 *
 *      I 2 C
 *
```

```
 */
/* IDs */
#define EAPI_ID_I2C_EXTERNAL       EAPI_UINT32_C(0) /* Baseboard I2C Interface
                                                     * required
                                                     */
#define EAPI_ID_I2C_LVDS_1         EAPI_UINT32_C(1) /* LVDS/EDP 1 Interface
                                                     * (optional)
                                                     */
#define EAPI_ID_I2C_LVDS_2         EAPI_UINT32_C(2) /* LVDS/EDP 2 Interface
                                                     * (optional)
                                                     */

/*
 * I2C Address Format
 *
 *  L = Set to 0
 *  H = Set to 1
 *  X = Don't Care(Direction Bit)
 *  0-F Address Bit
 *
 *  Bits 31-16 are Reserved and should be set to 0
 *
 *  Bit Offset      | F E D C B A 9 8 7 6 5 4 3 2 1 0
 *  ----------------+--------------------------------
 *  7  Bit Address  | L L L L L L L 6 5 4 3 2 1 0 X
 *  10 Bit Address  | H H H H L 9 8 X 7 6 5 4 3 2 1 0
 *
 *  Examples where Don't Care bits set to 0
 *            Encoded Encoded
 *  Address     7Bit    10Bit
 *    0x01      0x02   0xF001
 *    0x58      0xA0   0xF058
 *    0x59      0xA2   0xF059
 *    0x77      0xEE   0xF077
 *    0x3FF            0xF6FF
 *
 */
#define EAPI_I2C_ENC_7BIT_ADDR(x)  EAPI_UINT8_C(((x)&0x07F)<<1)
#define EAPI_I2C_DEC_7BIT_ADDR(x)  EAPI_UINT8_C(((x)>>1)&0x07F)
/*
 * EApiI2CGetBusCap
 *
 * Condition                              | Return Values
 * ---------------------------------------+------------------------------
 * Library Uninitialized                  | EAPI_STATUS_NOT_INITIALIZED
 * pMaxBlkLen==NULL                       | EAPI_STATUS_INVALID_PARAMETER
 * unknown Id                             | EAPI_STATUS_UNSUPPORTED
 * Common Error                           | Common Error Code
 * Else                                   | EAPI_STATUS_SUCCESS
 */
EApiStatus_t
EAPI_CALLTYPE
EApiI2CGetBusCap(
    __IN  EApiId_t  Id         , /* I2C Bus Id */
    __OUT uint32_t *pMaxBlkLen   /* Max Block Length
                                  * Supported on this
                                  * interface
                                  */
    );

/*
 * EApiI2CWriteRead
 *
 * Condition                               | Return Values
 * ----------------------------------------+------------------------------
 * Library Uninitialized                   | EAPI_STATUS_NOT_INITIALIZED
 * (WriteBCnt>1)&&(pWBuffer==NULL)         | EAPI_STATUS_INVALID_PARAMETER
 * (ReadBCnt>1)&&(pRBuffer==NULL)          | EAPI_STATUS_INVALID_PARAMETER
 * (ReadBCnt>1)&&(RBufLen==0)              | EAPI_STATUS_INVALID_PARAMETER
 * ((WriteBCnt==0)&&(ReadBCnt==0))         | EAPI_STATUS_INVALID_PARAMETER
 * unknown Id                              | EAPI_STATUS_UNSUPPORTED
 * WriteBCnt>(pMaxBlkLen+1)                | EAPI_STATUS_INVALID_BLOCK_LENGTH
 * ReadBCnt>(pMaxBlkLen+1)                 | EAPI_STATUS_INVALID_BLOCK_LENGTH
 * Bus Busy  SDA/SDC low                   | EAPI_STATUS_BUSY_COLLISION
 * Arbitration Error/Collision Error       | EAPI_STATUS_BUSY_COLLISION
 *   On Write 1 write cycle                |
 *   SDA Remains low                       |
```

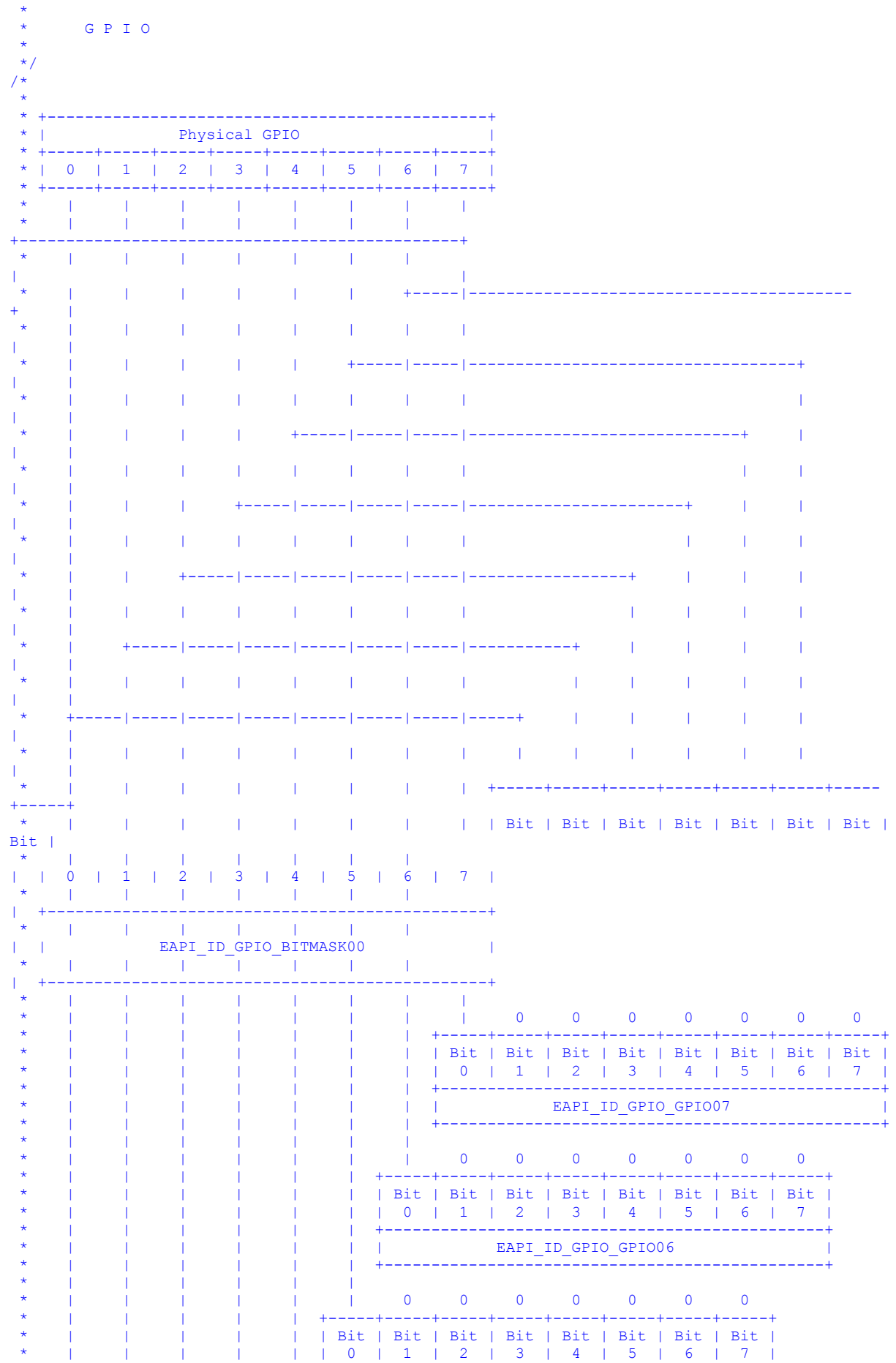
```
 * Timeout due to clock stretching          | EAPI_STATUS_TIMEOUT
 * start<Addr Byte><W>Nak                   | EAPI_STATUS_NOT_FOUND
 * start<Addr Byte><R>Nak                   | EAPI_STATUS_NOT_FOUND
 * start<Addr Byte><W>Ack<Data Byte 1>Nak   | EAPI_STATUS_WRITE_ERROR
 * ReadBCnt>(RBufLen+1)                     | EAPI_STATUS_MORE_DATA
 * Common Error                             | Common Error Code
 * Else                                     | EAPI_STATUS_SUCCESS
 */
EApiStatus_t
EAPI_CALLTYPE
EApiI2CWriteReadRaw(
    __IN     EApiId_t  Id       , /* I2C Bus Id */
    __IN     uint8_t   Addr     , /* Encoded 7Bit I2C
                                   * Device Address
                                   */
    __INOPT  void     *pWBuffer , /* Write Data pBuffer */
    __IN     uint32_t  WriteBCnt, /* Number of Bytes to
                                   * write plus 1
                                   */
    __OUTOPT void     *pRBuffer , /* Read Data pBuffer */
    __IN     uint32_t  RBufLen  , /* Data pBuffer Length */
    __IN     uint32_t  ReadBCnt   /* Number of Bytes to
                                   * Read plus 1
                                   */
    );

#define EApiI2CWriteRaw(Id, Addr, pBuffer, ByteCnt)         \
          EApiI2CWriteReadRaw(Id, Addr, pBuffer, ByteCnt, NULL, 0, 0)
#define EApiI2CReadRaw(Id, Addr, pBuffer, BufLen, ByteCnt) \
          EApiI2CWriteReadRaw(Id, Addr, NULL, 0, pBuffer, BufLen, ByteCnt)


#define EAPI_I2C_ENC_10BIT_ADDR(x) EAPI_UINT32_C(((x)&0xFF)|(((x)&0x0300)<<1)|0xF000)
#define EAPI_I2C_DEC_10BIT_ADDR(x) EAPI_UINT32_C(((x)&0xFF)|(((x)>>1)&0x300))
#define EAPI_I2C_IS_10BIT_ADDR(x)  (((x)&0xF800)==0xF000)
#define EAPI_I2C_IS_7BIT_ADDR(x)   (!EAPI_I2C_IS_10BIT_ADDR(x))

/*
 * I2C Transfer Types
 * Bits 31 & 30 Selects Command Type
 *
 * Transfer Type 1:
 * Address Format : 7Bit
 * Command Type   : None
 * Data Direction : Write
 * Start<Addr Byte><W>Ack<Data Byte 1>Ack Stop
 *
 * Transfer Type 2:
 * Address Format : 7Bit
 * Command Type   : None
 * Data Direction : Read
 * Start<Addr Byte><R>Ack<Data Byte 1>Nak Stop
 *
 * Transfer Type 3:
 * Address Format : 7Bit
 * Command Type   : Standard
 * Data Direction : Write
 * Start<Addr Byte><W>Ack<CMD Byte>Ack<Data Byte 1>Ack Stop
 *
 * Transfer Type 4:
 * Address Format : 7Bit
 * Command Type   : Standard
 * Data Direction : Read
 * Start<Addr Byte><W>Ack<CMD Byte>Ack
 * Start<Addr Byte><R>Ack<Data Byte 1>Nak Stop
 *
 * Transfer Type 5:
 * Address Format : 7Bit
 * Command Type   : Extended
 * Data Direction : Write
 * Start<Addr Byte><W>Ack<CMD Byte MSB>Ack<CMD Byte LSB>Ack<Data Byte 1>Ack Stop
 *
 * Transfer Type 6:
 * Address Format : 7Bit
 * Command Type   : Extended
 * Data Direction : Read
 * Start<Addr Byte><W>Ack<CMD Byte MSB>Ack<CMD Byte LSB>Ack
```

```
 * Start<Addr Byte><R>Ack<Data Byte 1>Nak Stop
 *
 * Transfer Type 7:
 * Address Format : 10Bit
 * Command Type   : None
 * Data Direction : Write
 * Start<Addr Byte MSB><W>Ack<Addr Byte LSB>Ack<Data Byte 1>Ack Stop
 *
 * Transfer Type 8:
 * Address Format : 10Bit
 * Command Type   : None
 * Data Direction : Read
 * Start<Addr Byte MSB><W>Ack<Addr Byte LSB>Ack
 * Start<Addr Byte MSB><R>Ack<Data Byte 1>Nak Stop
 *
 * Transfer Type 9:
 * Address Format : 10Bit
 * Command Type   : Standard
 * Data Direction : Write
 * Start<Addr Byte MSB><W>Ack<Addr Byte LSB>Ack<CMD Byte>Ack<Data Byte 1>Ack Stop
 *
 * Transfer Type 10:
 * Address Format : 10Bit
 * Command Type   : Standard
 * Data Direction : Read
 * Start<Addr Byte MSB><W>Ack<Addr Byte LSB>Ack<CMD Byte>Ack
 * Start<Addr Byte MSB><R>Ack<Data Byte 1>Nak Stop
 *
 * Transfer Type 11:
 * Address Format : 10Bit
 * Command Type   : Extended
 * Data Direction : Write
 * Start<Addr Byte MSB><W>Ack<Addr Byte LSB>Ack<CMD Byte MSB>Ack<CMD Byte LSB>Ack<Data Byte
1>Ack Stop
 *
 * Transfer Type 12:
 * Address Format : 10Bit
 * Command Type   : Extended
 * Data Direction : Read
 * Start<Addr Byte MSB><W>Ack<Addr Byte LSB>Ack<CMD Byte MSB>Ack<CMD Byte LSB>Ack
 * Start<Addr Byte MSB><R>Ack<Data Byte 1>Nak Stop
 *
 */
#define EAPI_I2C_STD_CMD          EAPI_UINT32_C(0<<30)
#define EAPI_I2C_EXT_CMD          EAPI_UINT32_C(2<<30)
#define EAPI_I2C_NO_CMD           EAPI_UINT32_C(1<<30)
#define EAPI_I2C_CMD_TYPE_MASK    EAPI_UINT32_C(3<<30)

#define EAPI_I2C_ENC_STD_CMD(x)  EAPI_UINT32_C(((x)&0xFF)|EAPI_I2C_STD_CMD)
#define EAPI_I2C_ENC_EXT_CMD(x)  EAPI_UINT32_C(((x)&0xFFFF)|EAPI_I2C_EXT_CMD)
#define EAPI_I2C_IS_EXT_CMD(x)
(EAPI_UINT32_C((x)&(EAPI_I2C_CMD_TYPE_MASK))==EAPI_I2C_EXT_CMD)
#define EAPI_I2C_IS_STD_CMD(x)
(EAPI_UINT32_C((x)&(EAPI_I2C_CMD_TYPE_MASK))==EAPI_I2C_STD_CMD)
#define
EAPI_I2C_IS_NO_CMD(x)    (EAPI_UINT32_C((x)&(EAPI_I2C_CMD_TYPE_MASK))==EAPI_I2C_NO_CMD)
/*
 * EApiI2CReadTransfer
 * Addr Byte 1 Below Designates Addr MSB in a 10bit address transfer and
 * the complete address in an 7bit address transfer.
 *
 * Condition                                | Return Values
 * -----------------------------------------+-----------------------------
 * Library Uninitialized                    | EAPI_STATUS_NOT_INITIALIZED
 * pBuffer==NULL                            | EAPI_STATUS_INVALID_PARAMETER
 * BufLen==0                                | EAPI_STATUS_INVALID_PARAMETER
 * ByteCnt==0                               | EAPI_STATUS_INVALID_PARAMETER
 * unknown Id                               | EAPI_STATUS_UNSUPPORTED
 * ByteCnt>pMaxBlkLen                       | EAPI_STATUS_INVALID_BLOCK_LENGTH
 * Bus Busy  SDA/SDC low                    | EAPI_STATUS_BUSY_COLLISION
 * Arbitration Error/Collision Error        | EAPI_STATUS_BUSY_COLLISION
 *   On Write 1 write cycle                 |
 *   SDA Remains low                        |
 * Time-out due to clock stretching         | EAPI_STATUS_TIMEOUT
 * start<Addr Byte 1><W>Nak                 | EAPI_STATUS_NOT_FOUND
 * start<Addr Byte 1><R>Nak                 | EAPI_STATUS_NOT_FOUND
 * start<Addr Byte 1><W>Ack<Addr Byte 2>Nak | EAPI_STATUS_WRITE_ERROR or
```

```
 *                                          | EAPI_STATUS_NOT_FOUND
 * start<Addr Byte 1><W>Ack<CMD Byte 1>Nak  | EAPI_STATUS_WRITE_ERROR
 * start<Addr Byte 1><W>Ack<Data Byte 1>Nak | EAPI_STATUS_WRITE_ERROR
 * ByteCnt>BufLen                           | EAPI_STATUS_MORE_DATA
 * Common Error                             | Common Error Code
 * Else                                     | EAPI_STATUS_SUCCESS
 */
EApiStatus_t
EAPI_CALLTYPE
EApiI2CReadTransfer(
    __IN  EApiId_t  Id      , /* I2C Bus Id */
    __IN  uint32_t  Addr    , /* Encoded 7/10Bit I2C
                               * Device Address
                               */
    __IN  uint32_t  Cmd     , /* I2C Command/Offset */
    __OUT     void *pBuffer , /* Transfer Data pBuffer */
    __IN  uint32_t  BufLen  , /* Data pBuffer Length */
    __IN  uint32_t  ByteCnt   /* Byte Count to read */
    );
/*
 * EApiI2CWriteTransfer
 * Addr Byte 1 Below Designates Addr MSB in a 10bit address transfer and
 * the complete address in an 7bit address transfer.
 *
 * Condition                                | Return Values
 * -----------------------------------------+-----------------------------
 * Library Uninitialized                    | EAPI_STATUS_NOT_INITIALIZED
 * pBuffer==NULL                            | EAPI_STATUS_INVALID_PARAMETER
 * ByteCnt==0                               | EAPI_STATUS_INVALID_PARAMETER
 * unknown Id                               | EAPI_STATUS_UNSUPPORTED
 * ByteCnt+(overhead)>pMaxBlkLen            | EAPI_STATUS_INVALID_BLOCK_LENGTH
 * Bus Busy  SDA/SDC low                    | EAPI_STATUS_BUSY_COLLISION
 * Arbitration Error/Collision Error        | EAPI_STATUS_BUSY_COLLISION
 *   On Write 1 write cycle                 |
 *   SDA Remains low                        |
 * Time-out due to clock stretching         | EAPI_STATUS_TIMEOUT
 * start<Addr Byte 1><W>Nak                 | EAPI_STATUS_NOT_FOUND
 * start<Addr Byte 1><R>Nak                 | EAPI_STATUS_NOT_FOUND
 * start<Addr Byte 1><W>Ack<Addr Byte 2>Nak | EAPI_STATUS_WRITE_ERROR or
 *                                          | EAPI_STATUS_NOT_FOUND
 * start<Addr Byte 1><W>Ack<CMD Byte 1>Nak  | EAPI_STATUS_WRITE_ERROR
 * start<Addr Byte 1><W>Ack<Data Byte 1>Nak | EAPI_STATUS_WRITE_ERROR
 * Common Error                             | Common Error Code
 * Else                                     | EAPI_STATUS_SUCCESS
 */
EApiStatus_t
EAPI_CALLTYPE
EApiI2CWriteTransfer(
    __IN  EApiId_t  Id      , /* I2C Bus Id */
    __IN  uint32_t  Addr    , /* Encoded 7/10Bit I2C
                               * Device Address
                               */
    __IN  uint32_t  Cmd     , /* I2C Command/Offset */
    __IN      void *pBuffer , /* Transfer Data pBuffer */
    __IN  uint32_t  ByteCnt   /* Byte Count to write */
    );

/*
 * I2C Probe Types
 *
 * Probe Type 1:
 * Address Format : 7Bit
 * Start<Addr Byte><W>Ack Stop
 *
 * Probe Type 2:
 * Address Format : 10Bit
 * Start<Addr Byte MSB><W>Ack<Addr Byte LSB>Ack Stop
 *
 */

/*
 * EApiI2CProbeDevice
 *
 * Condition                                  | Return Values
 * -------------------------------------------+-----------------------------
 * Library Uninitialized                      | EAPI_STATUS_NOT_INITIALIZED
 * Bus Busy  SDA/SDC low                      | EAPI_STATUS_BUSY_COLLISION
```

```
 * Arbitration Error/Collision Error            | EAPI_STATUS_BUSY_COLLISION
 *   On Write 1 write cycle                     |
 *   SDA Remains low                            |
 * Time-out due to clock stretching             | EAPI_STATUS_TIMEOUT
 *                                              |
 * 7Bit Address                                 |
 * start<Addr Byte><W>Nak                       | EAPI_STATUS_NOT_FOUND
 *                                              |
 * 10Bit Address                                |
 * start<Addr Byte MSB><W>Nak                   | EAPI_STATUS_NOT_FOUND
 * start<Addr Byte MSB><W>Ack<Addr Byte LSB>Nak | EAPI_STATUS_NOT_FOUND
 *                                              |
 * Common Error                                 | Common Error Code
 * Else                                         | EAPI_STATUS_SUCCESS
 */
EApiStatus_t
EAPI_CALLTYPE
EApiI2CProbeDevice(
    __IN  EApiId_t  Id   , /* I2C Bus Id */
    __IN  uint32_t  Addr   /* Encoded 7/10Bit
                            * I2C Device Address
                            */
    );

/*
 *
 *      W A T C H D O G
 *
 */

/*
 *
 * After EApiWDogStart
 *
 * |<- Delay ->|<- Event Timeout ->|<- Reset Timeout ->|
 * A-----------B-------------------C-------------------D
 *
 *
 * After EApiWDogTrigger
 *
 * |<- Event Timeout ->|<- Reset Timeout ->|
 * E-------------------F-------------------G
 *
 *
 * Stage A
 *  Watchdog is started.
 *
 * Stage B
 *  Initial Delay Period is exhausted.
 *
 * Stage C/F
 *  Event is triggered, NMI, IRQ, or PIN is Triggered.
 *  To Allow for possible Software Recovery.
 *
 * Stage D/G
 *  System is reset.
 *
 * Stage E
 *  Watchdog is Triggered.
 *
 * EApiWDogTrigger/EApiWDogStop Must be called before Stage C/F
 * to prevent event from being generated.
 *
 * EApiWDogTrigger/EApiWDogStop Must be called before Stage D/G
 * to prevent The system from being reset.
 *
 */

/*
 * EApiWDogGetCap
 *
 * Condition                              | Return Values
 * ---------------------------------------+------------------------------
 * Library Uninitialized                  | EAPI_STATUS_NOT_INITIALIZED
 * Unsupported                            | EAPI_STATUS_UNSUPPORTED
 * pMaxDelay==NULL&&                      |
 * pMaxResetTimeout==NULL&&               |
```

```
 * pMaxEventTimeout==NULL                | EAPI_STATUS_INVALID_PARAMETER
 * Common Error                          | Common Error Code
 * Else                                  | EAPI_STATUS_SUCCESS
 */
EApiStatus_t
EAPI_CALLTYPE
EApiWDogGetCap(
    __OUTOPT uint32_t *pMaxDelay       , /* Maximum Supported
                                         * Delay in milliseconds
                                         */
    __OUTOPT uint32_t *pMaxEventTimeout, /* Maximum Supported
                                         * Event Timeout in
                                         * milliseconds
                                         * 0 == Unsupported
                                         */
    __OUTOPT uint32_t *pMaxResetTimeout  /* Maximum Supported
                                         * Reset Timeout in
                                         * milliseconds
                                         */
    );
/*
 * EApiWDogStart
 *
 * Condition                           | Return Values
 * ------------------------------------+------------------------------
 * Library Uninitialized               | EAPI_STATUS_NOT_INITIALIZED
 * Unsupported                         | EAPI_STATUS_UNSUPPORTED
 * Delay>pMaxDelay                     | EAPI_STATUS_INVALID_PARAMETER
 * EventTimeout>pMaxEventTimeout       | EAPI_STATUS_INVALID_PARAMETER
 * ResetTimeout>pMaxResetTimeout       | EAPI_STATUS_INVALID_PARAMETER
 * Already Running                     | EAPI_STATUS_RUNNING
 * Common Error                        | Common Error Code
 * Else                                | EAPI_STATUS_SUCCESS
 */
EApiStatus_t
EAPI_CALLTYPE
EApiWDogStart(
    __IN  uint32_t Delay       , /* Delay in milliseconds */
    __IN  uint32_t EventTimeout, /* Event Timeout in
                                  * milliseconds
                                  */
    __IN  uint32_t ResetTimeout  /* Reset Timeout in
                                  * milliseconds
                                  */
    );

/*
 * EApiWDogTrigger
 *
 * Condition                           | Return Values
 * ------------------------------------+------------------------------
 * Library Uninitialized               | EAPI_STATUS_NOT_INITIALIZED
 * Unsupported                         | EAPI_STATUS_UNSUPPORTED
 * Watchdog Not Started                | EAPI_STATUS_ERROR
 * Common Error                        | Common Error Code
 * Else                                | EAPI_STATUS_SUCCESS
 */
EApiStatus_t
EAPI_CALLTYPE
EApiWDogTrigger(void);

/*
 * EApiWDogStop
 *
 * Condition                           | Return Values
 * ------------------------------------+------------------------------
 * Library Uninitialized               | EAPI_STATUS_NOT_INITIALIZED
 * Unsupported                         | EAPI_STATUS_UNSUPPORTED
 * Common Error                        | Common Error Code
 * Else                                | EAPI_STATUS_SUCCESS
 */
EApiStatus_t
EAPI_CALLTYPE
EApiWDogStop(void);


/*
```


G P I O
Physical GPIO
0 1 2 3 4 5 6 7
Bit 0 Bit 1 Bit 2 Bit 3 Bit 4 Bit 5 Bit 6 Bit 7
EAPI_ID_GPIO_BITMASK00
EAPI_ID_GPIO_GPIO07
EAPI_ID_GPIO_GPIO06

```
 *    |     |     |     |     |  +-------------------------------------------+
 *    |     |     |     |     |  |            EAPI_ID_GPIO_GPIO05            |
 *    |     |     |     |     |  +-------------------------------------------+
 *    |     |     |     |     |
 *    |     |     |     |     |     0     0     0     0     0     0     0     0
 *    |     |     |     |     |  +-----+-----+-----+-----+-----+-----+-----+-----+
 *    |     |     |     |     |  | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit |
 *    |     |     |     |     |  |  0  |  1  |  2  |  3  |  4  |  5  |  6  |  7  |
 *    |     |     |     |     |  +-------------------------------------------+
 *    |     |     |     |     |  |            EAPI_ID_GPIO_GPIO04            |
 *    |     |     |     |     |  +-------------------------------------------+
 *    |     |     |     |     |
 *    |     |     |     |     |     0     0     0     0     0     0     0
 *    |     |     |     |  +-----+-----+-----+-----+-----+-----+-----+-----+
 *    |     |     |     |  | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit |
 *    |     |     |     |  |  0  |  1  |  2  |  3  |  4  |  5  |  6  |  7  |
 *    |     |     |     |  +-------------------------------------------+
 *    |     |     |     |  |            EAPI_ID_GPIO_GPIO03            |
 *    |     |     |     |  +-------------------------------------------+
 *    |     |     |     |
 *    |     |     |     |     0     0     0     0     0     0     0
 *    |     |     |  +-----+-----+-----+-----+-----+-----+-----+-----+
 *    |     |     |  | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit |
 *    |     |     |  |  0  |  1  |  2  |  3  |  4  |  5  |  6  |  7  |
 *    |     |     |  +-------------------------------------------+
 *    |     |     |  |            EAPI_ID_GPIO_GPIO02            |
 *    |     |     |  +-------------------------------------------+
 *    |     |     |
 *    |     |     |     0     0     0     0     0     0     0
 *    |     |  +-----+-----+-----+-----+-----+-----+-----+-----+
 *    |     |  | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit |
 *    |     |  |  0  |  1  |  2  |  3  |  4  |  5  |  6  |  7  |
 *    |     |  +-------------------------------------------+
 *    |     |  |            EAPI_ID_GPIO_GPIO01            |
 *    |     |  +-------------------------------------------+
 *    |     |
 *    |     |     0     0     0     0     0     0     0
 *    | +-----+-----+-----+-----+-----+-----+-----+-----+
 *    | | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit |
 *    | |  0  |  1  |  2  |  3  |  4  |  5  |  6  |  7  |
 *    | +-------------------------------------------+
 *    | |            EAPI_ID_GPIO_GPIO00            |
 *    | +-------------------------------------------+
 */
/* IDs */
/*
 * Individual ID Per GPIO Mapping
 */
#define EAPI_GPIO_GPIO_ID(GPIO_NUM)    EAPI_UINT32_C(GPIO_NUM)
#define EAPI_GPIO_GPIO_BITMASK  EAPI_UINT32_C(1)


#define EAPI_ID_GPIO_GPIO000     EAPI_GPIO_GPIO_ID(0) /* (Optional) */
#define EAPI_ID_GPIO_GPIO001     EAPI_GPIO_GPIO_ID(1) /* (Optional) */
#define EAPI_ID_GPIO_GPIO002     EAPI_GPIO_GPIO_ID(2) /* (Optional) */
#define EAPI_ID_GPIO_GPIO003     EAPI_GPIO_GPIO_ID(3) /* (Optional) */

/*
 * Multiple GPIOs Per ID Mapping
 */
#define EAPI_GPIO_BANK_ID(GPIO_NUM)     EAPI_UINT32_C(0x10000|((GPIO_NUM)>>5))
#define EAPI_GPIO_BANK_MASK(GPIO_NUM) EAPI_UINT32_C((1<<((GPIO_NUM)&0x1F))
#define EAPI_GPIO_BANK_TEST_STATE(GPIO_NUM, TState, TValue) \
                      (((TValue>>((GPIO_NUM)&0x1F))&1)==(TState))

#define EAPI_ID_GPIO_BANK00    EAPI_GPIO_BANK_ID( 0) /* GPIOs  0 - 31
                                                      * (optional)
                                                      */
#define EAPI_ID_GPIO_BANK01    EAPI_GPIO_BANK_ID(32) /* GPIOs 32 - 63
                                                      * (optional)
                                                      */
#define EAPI_ID_GPIO_BANK02    EAPI_GPIO_BANK_ID(64) /* GPIOs 64 - 95
                                                      * (optional)
                                                      */


/* Bit mask Bit States */
```

```
#define EAPI_GPIO_BITMASK_SELECT   1
#define EAPI_GPIO_BITMASK_NOSELECT 0

/* Levels */
#define EAPI_GPIO_LOW     0
#define EAPI_GPIO_HIGH    1

/* Directions */
#define EAPI_GPIO_INPUT   1
#define EAPI_GPIO_OUTPUT  0

/*
 * EApiGPIOGetDirectionCaps
 *
 * Condition                          | Return Values
 * -----------------------------------+------------------------------
 * Library Uninitialized              | EAPI_STATUS_NOT_INITIALIZED
 * ((pOutputs==NULL)&&(pInputs==NULL))| EAPI_STATUS_INVALID_PARAMETER
 * Unsupported Id                     | EAPI_STATUS_UNSUPPORTED
 * Common Error                       | Common Error Code
 * Else                               | EAPI_STATUS_SUCCESS
 */
EApiStatus_t
EAPI_CALLTYPE
EApiGPIOGetDirectionCaps(
    __IN     EApiId_t Id        , /* GPIO Id */
    __OUTOPT uint32_t *pInputs  , /* Supported GPIO Input
                                   * Bit Mask
                                   */
    __OUTOPT uint32_t *pOutputs   /* Supported GPIO Output
                                   * Bit Mask
                                   */
    );
/*
 * EApiGPIOGetDirection
 *
 * Condition                          | Return Values
 * -----------------------------------+------------------------------
 * Library Uninitialized              | EAPI_STATUS_NOT_INITIALIZED
 * Bitmask==0                         | EAPI_STATUS_INVALID_PARAMETER
 * Unsupported Id                     | EAPI_STATUS_UNSUPPORTED
 * (Bitmask&~(pInputs|pOutputs))      | EAPI_STATUS_INVALID_BITMASK
 * Common Error                       | Common Error Code
 * Else                               | EAPI_STATUS_SUCCESS
 */
EApiStatus_t
EAPI_CALLTYPE
EApiGPIOGetDirection(
    __IN  EApiId_t Id          , /* GPIO Id */
    __IN  uint32_t Bitmask     , /* Bit mask of Affected
                                  * Bits
                                  */
    __OUT uint32_t *pDirection   /* Current Direction */
    );
/*
 * EApiGPIOSetDirection
 *
 * Condition                          | Return Values
 * -----------------------------------+------------------------------
 * Library Uninitialized              | EAPI_STATUS_NOT_INITIALIZED
 * Bitmask==0                         | EAPI_STATUS_INVALID_PARAMETER
 * Unsupported Id                     | EAPI_STATUS_UNSUPPORTED
 * (Bitmask&~(pInputs|pOutputs))      | EAPI_STATUS_INVALID_BITMASK
 * (Bitmask&pDirection)&~pInputs      | EAPI_STATUS_INVALID_DIRECTION
 * (Bitmask&~pDirection)&~pOutputs    | EAPI_STATUS_INVALID_DIRECTION
 * Common Error                       | Common Error Code
 * Else                               | EAPI_STATUS_SUCCESS
 */
EApiStatus_t
EAPI_CALLTYPE
EApiGPIOSetDirection(
    __IN  EApiId_t Id          , /* GPIO Id */
    __IN  uint32_t Bitmask     , /* Bit mask of Affected
                                  * Bits
                                  */
    __IN  uint32_t Direction     /* Direction */
    );
```

```
/*
 * EApiGPIOGetLevel
 *
 * Condition                          | Return Values
 * -----------------------------------+-----------------------------
 * Library Uninitialized              | EAPI_STATUS_NOT_INITIALIZED
 * Bitmask==0                         | EAPI_STATUS_INVALID_PARAMETER
 * Unsupported Id                     | EAPI_STATUS_UNSUPPORTED
 * (Bitmask&~(pInputs|pOutputs))      | EAPI_STATUS_INVALID_BITMASK
 * Common Error                       | Common Error Code
 * Else                               | EAPI_STATUS_SUCCESS
 */
EApiStatus_t
EAPI_CALLTYPE
EApiGPIOGetLevel(
    __IN  EApiId_t Id          , /* GPIO Id */
    __IN  uint32_t Bitmask     , /* Bit mask of Affected
                                  * Bits
                                  */
    __OUT uint32_t *pLevel       /* Current Level */
    );
/*
 * EApiGPIOSetLevel
 *
 * Condition                          | Return Values
 * -----------------------------------+-----------------------------
 * Library Uninitialized              | EAPI_STATUS_NOT_INITIALIZED
 * Bitmask==0                         | EAPI_STATUS_INVALID_PARAMETER
 * Unsupported Id                     | EAPI_STATUS_UNSUPPORTED
 * Common Error                       | Common Error Code
 * Else                               | EAPI_STATUS_SUCCESS
 */
EApiStatus_t
EAPI_CALLTYPE
EApiGPIOSetLevel(
    __IN  EApiId_t Id          , /* GPIO Id */
    __IN  uint32_t Bitmask     , /* Bit mask of Affected
                                  * Bits
                                  */
    __IN  uint32_t Level         /* Level */
    );


#include "EApiCOM0.h"    /* COMExpress Platform Specific ID Mappings   */
#include "EApiTCA.h"     /* TCA Platform Specific ID Mappings          */
#include "EApiETX.h"     /* ETX Platform Specific ID Mappings          */
#include "EApiQ7.h"      /* Q Seven Platform Specific ID Mappings      */
#include "EApiMOPS.h"    /* MOPS Platform Specific ID Mappings         */
#include "EApiPISA.h"    /* PISA Platform Specific ID Mappings         */
#include "EApiEPIC.h"    /* EPIC Platform Specific ID Mappings         */
/*****************************************************************************/

#ifdef __cplusplus
}
#endif

#endif /* _EAPI_H_ */
```

10.2 EApiCOM0.h

```
/*
 *<KHeader>
 *+=========================================================================
 *I  Project Name: EApiDK Embedded Application Development Kit
 *+=========================================================================
 *I  $HeadURL: https://eapidk.svn.sourceforge.net/svnroot/eapidk/trunk/include/EApiCOM0.h $
 *+=========================================================================
 *I   Copyright: Copyright (c) 2009-2010, PICMG
 *I      Author: John Kearney,                John.Kearney@kontron.com
 *I
 *I     License: All rights reserved. This program and the accompanying
 *I              materials are licensed and made available under the
 *I              terms and conditions of the BSD License which
 *I              accompanies this distribution. The full text of the
 *I              license may be found at
 *I              http://opensource.org/licenses/bsd-license.php
 *I
 *I              THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "
 *I              AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF
 *I              ANY KIND, EITHER EXPRESS OR IMPLIED.
 *I
 *I Description: Auto Created for EApiCOM0.h
 *I
 *+-------------------------------------------------------------------------
 *I
 *I  File Name            : EApiCOM0.h
 *I  File Location        : include
 *I  Last committed       : $Revision: 74 $
 *I  Last changed by      : $Author: dethrophes $
 *I  Last changed date    : $Date: 2010-06-23 21:26:50 +0200 (Mi, 23 Jun 2010) $
 *I  ID                   : $Id: EApiCOM0.h 74 2010-06-23 19:26:50Z dethrophes $
 *I
 *+=========================================================================
 *</KHeader>
 */

#ifndef _EAPICOM0_H_
#define _EAPICOM0_H_


/*
 *
 *      B O A R D   I N F O M A T I O N   S T R I N G S
 *
 */
/* IDS */
#define EAPI_COM0_ID_BOARD_MANUFACTURER_STR     EAPI_ID_BOARD_MANUFACTURER_STR
#define EAPI_COM0_ID_BOARD_NAME_STR             EAPI_ID_BOARD_NAME_STR
#define EAPI_COM0_ID_BOARD_SERIAL_STR           EAPI_ID_BOARD_SERIAL_STR
#define EAPI_COM0_ID_BOARD_BIOS_REVISION_STR    EAPI_ID_BOARD_BIOS_REVISION_STR
#define EAPI_COM0_ID_BOARD_PLATFORM_TYPE_STR    EAPI_ID_BOARD_PLATFORM_TYPE_STR

#define EAPI_COM0_PLATFORM_STR     "COMExpress"


/*
 *
 *      B O A R D   I N F O M A T I O N   V A L U E S
 *
 */
/* IDS */
#define EAPI_COM0_ID_BOARD_BOOT_COUNTER_VAL        EAPI_ID_BOARD_BOOT_COUNTER_VAL
#define EAPI_COM0_ID_BOARD_RUNNING_TIME_METER_VAL  EAPI_ID_BOARD_RUNNING_TIME_METER_VAL
#define EAPI_COM0_ID_BOARD_PNPID_VAL               EAPI_ID_BOARD_PNPID_VAL
#define EAPI_COM0_ID_BOARD_PLATFORM_REV_VAL        EAPI_ID_BOARD_PLATFORM_REV_VAL


#define EAPI_COM0_ID_BOARD_DRIVER_VERSION_VAL    EAPI_ID_BOARD_DRIVER_VERSION_VAL
#define EAPI_COM0_ID_BOARD_LIB_VERSION_VAL       EAPI_ID_BOARD_LIB_VERSION_VAL

#define EAPI_COM0_REV_1_0      EAPI_VER_CREATE(1, 0, 0)
```

```
#define EAPI_COM0_REV_2_0      EAPI_VER_CREATE(2, 0, 0)


#define EAPI_COM0_ID_HWMON_CPU_TEMP       EAPI_ID_HWMON_CPU_TEMP
#define EAPI_COM0_ID_HWMON_CHIPSET_TEMP   EAPI_ID_HWMON_CHIPSET_TEMP
#define EAPI_COM0_ID_HWMON_SYSTEM_TEMP    EAPI_ID_HWMON_SYSTEM_TEMP

#define EAPI_COM0_ID_HWMON_VOLTAGE_VCORE EAPI_ID_HWMON_VOLTAGE_VCORE
#define EAPI_COM0_ID_HWMON_VOLTAGE_2V5    EAPI_ID_HWMON_VOLTAGE_2V5
#define EAPI_COM0_ID_HWMON_VOLTAGE_3V3    EAPI_ID_HWMON_VOLTAGE_3V3
#define EAPI_COM0_ID_HWMON_VOLTAGE_VBAT   EAPI_ID_HWMON_VOLTAGE_VBAT
#define EAPI_COM0_ID_HWMON_VOLTAGE_5V     EAPI_ID_HWMON_VOLTAGE_5V
#define EAPI_COM0_ID_HWMON_VOLTAGE_5VSB   EAPI_ID_HWMON_VOLTAGE_5VSB
#define EAPI_COM0_ID_HWMON_VOLTAGE_12V    EAPI_ID_HWMON_VOLTAGE_12V

#define EAPI_COM0_ID_HWMON_FAN_CPU        EAPI_ID_HWMON_FAN_CPU
#define EAPI_COM0_ID_HWMON_FAN_SYSTEM     EAPI_ID_HWMON_FAN_SYSTEM

/*
 *
 *      B A C K L I G H T
 *
 */

/*
 * COM Express Backlight Fill Order
 *
 * Internal PWM
 * EAPI_COM0_ID_I2C_LVDS_1 I2C Device
 * SDVOB PWM
 * SDVOB I2C Device
 * SDVOC PWM
 * SDVOC I2C Device
 * DDI1 I2C Device
 * DDI2 I2C Device
 * DDI3 I2C Device
 *
 */
/* IDS */
#define EAPI_COM0_ID_BACKLIGHT_1              EAPI_ID_BACKLIGHT_1
#define EAPI_COM0_ID_BACKLIGHT_2              EAPI_ID_BACKLIGHT_2



/*
 *
 *      S T O R A G E
 *
 */
/* IDs */
#define EAPI_COM0_ID_STORAGE_STD              EAPI_ID_STORAGE_STD

/*
 *
 *      I 2 C
 *
 */
/* IDs */
#define EAPI_COM0_ID_I2C_EXTERNAL       EAPI_ID_I2C_EXTERNAL
#define EAPI_COM0_ID_I2C_LVDS_1         EAPI_ID_I2C_LVDS_1
#define EAPI_COM0_ID_I2C_LVDS_2         EAPI_ID_I2C_LVDS_2


/*
 *
 *      G P I O
 *
 */
/* IDs */
#define EAPI_COM0_ID_GPIO_BANK    EAPI_ID_GPIO_BANK00
#define EAPI_COM0_ID_GPIO_GPI0    EAPI_GPIO_GPIO_ID(0)
#define EAPI_COM0_ID_GPIO_GPI1    EAPI_GPIO_GPIO_ID(1)
#define EAPI_COM0_ID_GPIO_GPI2    EAPI_GPIO_GPIO_ID(2)
#define EAPI_COM0_ID_GPIO_GPI3    EAPI_GPIO_GPIO_ID(3)
#define EAPI_COM0_ID_GPIO_GPO0    EAPI_GPIO_GPIO_ID(4)
#define EAPI_COM0_ID_GPIO_GPO1    EAPI_GPIO_GPIO_ID(5)
#define EAPI_COM0_ID_GPIO_GPO2    EAPI_GPIO_GPIO_ID(6)
```

```
#define EAPI_COM0_ID_GPIO_GPO3    EAPI_GPIO_GPIO_ID(7)


#endif /* _EAPICOM0_H_ */
```

11 Vendor Specific IDs

In order to allow vendor specific device IDs for the EAPI without overlapping definition PNPID[2] are used.

11.1.1 Vendor Id breakdown

To allow for Vendor unique Ids the following Id Range is reserved 0xF0000000 – 0xFFFFFFF.

Bits	Description
[31:28]	0xF
[27:12]	Compressed ASCII PNPID(see chapter 12.1 page 77)
[11:0]	12Bit Vendor Specific ID

11.1.2 Support MACROS

Name	Description
EAPI_CREATE_CUST_ID	Allows creation of Custom Ids

11.1.3 Example

Id	Hex
EAPI_CREATE_CUST_ID('P', 'M', 'G', 0x123)	0xFA741123

2 For PNPID see Chapter 1.6 page 11

12 Standard Data Formats

12.1 Compressed ASCII PNPID

12.1.1 Definition

Compressed ASCII is defined as 5 bits per character, "00001b" = "A" ... "11010b" = "Z".

Byte	Bits	Description
0	[7]	0
0	[6:2]	First character in compressed ASCII
0	[1:0]	Second character in compressed ASCII bits[4:3]
1	[7:5]	Second character in compressed ASCII bits[2:0]
1	[4:0]	Third character in compressed ASCII

12.1.2 Example

Hex	Interpreted
0xA741	'PMG'

12.2 Specification Version Number Format

12.2.1 Definition

Bits	Description
[31:24]	Version
[23:16]	Revision
[15:0]	0

12.2.2 Example

Hex	Interpreted
0x03040000	3.4
0x01100000	1.16
0x02010000	2.1

12.3 General Version number Format

12.3.1 Definition

Bits	Description
[31:24]	Major Version
[23:16]	Minor Version
[15:0]	Build Number

12.3.2 Example

Hex	Interpreted
0x03040010	3.4.16
0x01100100	1.16.256
0x02010001	2.1.1

13 OS Specific Requirements

13.1 Windows

13.1.1 DLL Naming Convention

EAPI_X.dll

X represents the EAPI Specification Version Number

Example

EAPI_1.dll

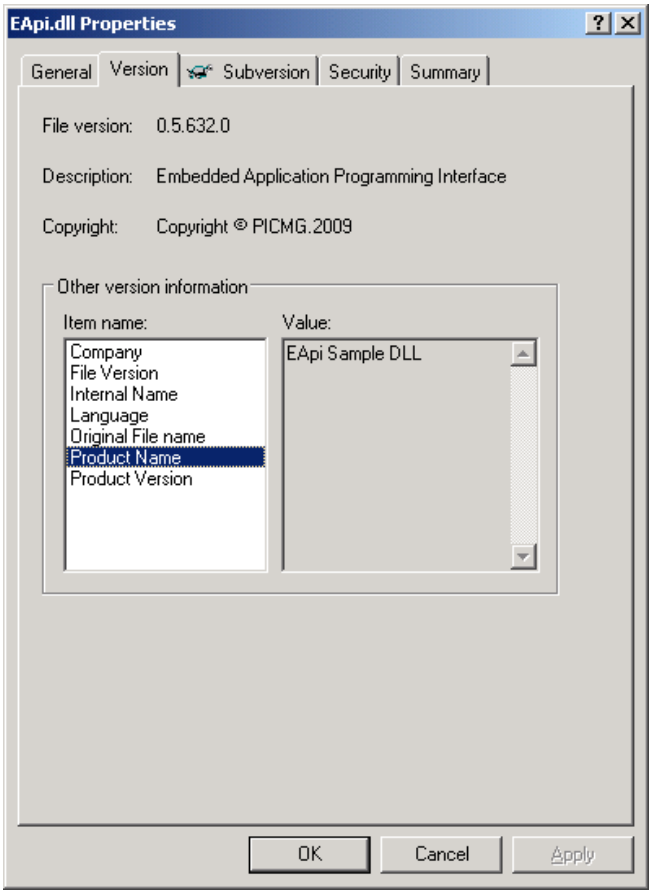
13.1.2 Version Resource Information

Problem

Due to the nature of the EAPI DLLs in Microsoft Windows it may not be possible to distinguish one manufacturers DLL from another. Although it would possible to do this using a tool that uses the API, It may not be possible to load the DLL, due to missing dependencies.

Solution

The solution is to require that Version Resource Information be present for every EAPI DLL. It is then easily possible to check Manufacturer and versions in Windows Explorer Properties window.



Sample

For an example implementation see EApi.rc in the EApiDK.

13.2 Linux/Unix Shared Library Naming Convention

Problem

Due to the nature of ELF Shared Libraries in Linux/UNIX/... it may not be possible to distinguish one manufacturer's DLL from another. Although it would possible to do this using a tool that uses the API, It may not be possible to load the shared library, due to missing dependencies.

Solution

Filename Convention

libEApiYYY.so.W.Z

Part	Example	Description
YYY	PMG	Vendor PNPID
W	1	EAPI Specification Version number
X	0	EAPI Specification Revision number

Soname Convention

libEApi.so.W

Part	Example	Description
W	1	EAPI Specification Version number

Example

Shared Library

Filename = libEapiPMG.so.1.0

soname = libEApi.so.1

in file system.

/usr/lib/libEApi.so.1 → /usr/lib/libEapiPMG.so.1.0

/usr/lib/libEApiPMG.so.1.0

see EApiDK for sample implementation.

13.3 ELF/a.out Format Shared Libraries

13.3.1 Library Output Format

Problem

Due to the nature of ELF Dynamic Link Libraries in Linux/UNIX/... it may not be possible to distinguish one manufacturer's DLL from another. Although it would possible to do this using a tool that uses the API, It may not be possible to load the DLL, due to missing dependencies.

Solution

The solution is to require the shared libraries be executable. Upon Execution the library should then print out the following information.

```
The Output format is
<Variable name>=Value
and should be matched by this regular expression. m/^\s*(\w+)\s*=\s*(.+)\s*$/
where
$1=Variable name
$2=Value/Data.
```

Variable Name	Description
SVersion	EApi Standard Version used to create Library
LVersion	Vendor Specific Library Version
Manufacturer	Library Manufacturer
MPNPID	Manufacturer PNPID
OFilename	Original File name
Description	Library Description

Sample

For an example implementation see Message.c in EApiDK.

```
+-----------------------------------------------------------------------------+
I                      Copyright (C) PICMG 2009                               I
+-----------------------------------------------------------------------------+
SVersion=0.5
LVersion=0.5.721
Manufacturer=PICMG
MPNPID=PMG
OFilename=libEApiPMG.so.0.5
Description=Embedded Application Programming Interface
```

14 Example Code

Example code and test software is available here http://eapidk.sourceforge.net/.

To Check out the EApiDK use

svn co https://eapidk.svn.sourceforge.net/svnroot/eapidk/trunk

15 Revision History

Revision	Date	Notes
0.00	Jun 25, 09	Initial Version
0.01	Jun 30, 09	multiple updates
0.02		
0.03	Jul 10, 09	Changed Format
0.04	Jul 14, 09	Minor changes
0.05	Jul 22, 09	Subcommittee review version
0.06	Sept 10, 09	Some CR and cleanup
D0.81	Sept 29, 09	closed most open CRs. Added PICMG boilerplate. Merged double glossary sections. Only open item is the multi stage feature for the watchdog timer.
D082	Oct 12, 09	Added 8.1 EApiWDogGetCap and second stage for watchdog timer Better explanation and examples for 9 GPIO Functions