Cx/Vx9 SDK Linux

Software Development Kit
Cx/Vx9
Interface Library
for
Linux

Software User's Manual

Revision 1.4

Order Codes:
CR9-SDK-LINUX
VR9-SDK-LINUX

www.sbs.com

Cx/Vx9 SDK Linux Software User's Manual

Manual Revision	By	Date	Comments
1.0	OA	Apr04	Initial revision – Cx/Vx9_SDK_Linux v1.0
1.1	JD	Nov04	Added CT9/CP9 VP9 and VR9 as this software has been developed to support all these platforms - Cx/Vx9_SDK_Linux v1.0
1.2	BG	Mai05	Added new Features and remove redundant features
1.3	BG	Jun05	Revise EEPROM and Watchdog chapter
1.4	BG	Sep05	Complete rework of the manual

For Immediate Contact

Europe

Sales:

SBS Technologies GmbH & Co.KG
Memminger Str. 14
86159 Augsburg
Germany

Phone +49 821 5034-0
Fax +49 821 5034-119
Email sales@sbs-europe.com

Technical Support

Phone +49 821 5034-170
Fax +49 821 5034-119
Email aug-support@sbs.com

USA & Rest of World

SBS Technologies, Inc.
6301 Chapel Hill Road
Raleigh, NC 27607
USA

Phone (919) 851-1101
Fax (919) 851-2844
Email sales.ec@sbs.com

Phone (919) 851-1101
Fax (919) 851-2844
Email support.sbc@sbs.com

Disclaimer

The information in this document has been carefully checked and is believed to be entirely reliable. However, no responsibility is assumed for inaccuracies.
SBS Technologies reserves the right to make changes without further notice to any products herein to improve reliability, function or design. SBS Technologies does not assume any liability arising out of the application or use of any product or circuit described herein; neither do they convey any license under their patent rights nor the rights of others. The software described herein will be provided on an "as is" basis and without warranty. SBS Technologies accepts no liability for incidental or consequential damages arising from use of the software. This Disclaimer of warranty extends to SBS Technologies licensee, to licensee's transferees and to licensee's customers or users and is in lieu of all warranties whether expressed, implied or statutory, including implied warranties of saleability or fitness for a particular purpose.
SBS Technologies does not accept any liability concerning the use of their products in life support systems.

Trademarks

Product and company names listed are trademarks or trade names of their respective companies owners.

0 Table of Contents

1 General Information

1.1 Introduction

The Cx/Vx9 Software Development Kit (SDK) package for Linux is a software library offering to the user a straight forward and easy to use API to access and control devices on the SBS Technologies Cx9/Vx9 single board computer series. Since the API has been defined for other boards as well, offering different sets of devices and/or being different in terms of hardware design, the term Unit is used in this manual for the various function blocks, thus focusing on functional blocks rather than dealing with details in hardware implementation.

The Cx9/Vx9 SDK package for Linux is a combination of a Library (.so/.a) and an underlying device driver (.(k)o) providing for low-level I/O port access and other basic interfaces to hardware and BIOS. In order to make use of the library in an application, the programmer has to do the following:

- Copy the header files and the library files (.a, .so) to the appropriate locations within your development environment
- Include the header files in C source code using the SDK package
- Include the library files in the link process

The application built with the header files and the librariy files then runs on the target Cx/Vx9 in conjunction with the device driver and the library.

1.2 Compatibility

The Cx/Vx9 SDK Linux package has been developed and tested to run on Suse Linux 9.2. Furthermore the package has been tested on Red Hat 9.0 (Kernel 2.4) and on Fedora Core 2 (Kernel 2.6). But it is also capable of supporting applications running on all other Linux Distributions. Be aware that with those Distributions there can be minor differences to what is described in these introductory chapters. However, the functionality and behavior is identical in all Linux variants.

1.3 Description of Files

The following gives a list of the files in this distribution along with a short explanation.

./driver/	
sdkdriver.c	Source file of the main device driver functions
led.c	Source file of the led functions
eeprom.c	Source file of the eeprom functions
temp.c	Source file of the functions concerning the temperature sensors
smbctl.c	Source file of the SMBus functions
wdog.c	Source file of the watchdog functions
gpio.c	Source file of the gpio functions
Makefile-2.4	Makefile to build the device driver for Kernel version 2.4.x
Makefile-2.6	Makefile to build the device driver for Kernel version 2.6.x
./lib/	
libsdkcxvx9.so	Shared library implementing the SDK API functions
libsdkcxvx9.a	Static library implementing the SDK API functions
sdkcxvx9lib.c	Source file of sdkcxvx9 libraries
Makefile	Makefile to build the library

./include/	
sdkcxvx9.h	Header file defining types and prototyping functions used in the SDK package (this is the only header file to be included in a source file using the SDK library)
sdkresult.h	File defining the SDK result codes
sh_data.h	File defining shared data modules used in the SDK package
or_types.h	File defining various base types used in the SDK API definition
./test/	
sdkcxvx9test	Console based test and sample application for the SDK library
sdkcxvx9test.c	Source file of sdkcxvx9test application
Makefile	Makefile to build the test and sample application
./docs/	
readme.txt	Text file containing revision information and latest changes
CxVx9_sdk_linux.pdf	Cx/Vx9 SDK software user manual (this file)
ipmi_readme.txt	Readme file giving information about the handling of the IPMI controller in Linux (only relevant for the CT9)
pnp_readme.txt	Readme file giving information about the handling of Plug&Play in Linux
sdkcxvx9test.txt	Readme file giving an overview of sdkcxvx9test

1.4 Installation

The Device Driver has to be compiled and installed on the target system. The library files also has to be installed on the target system where the application using the SDK package is run. The installation of the device driver and the library files has to be done manually. Please note that it is strongly recommended to login as root for the installation and executing the application using the SDK package. These steps are necessary for the complete installation:

- Create a character device node file called "/dev/sdkcx9_ctl" with major number 253 and minor number 0:
  #mknod /dev/sdkcx9_ctl c 253 0
- Copy the sdkcxvx9.tar.gz to your {INSTALLPATH}-directory and change to this directory:
  #cp sdkcxvx9.tar.gz {INSTALLPATH}
  #cd {INSTALLPATH}
- Extract the files in sdkcxvx9.tar.gz to the {INSTALLPATH}-directory:
  #tar xfz sdkcxvx9.tar.gz
- Choose the Makefile according to your Kernelversion:
  #cd driver
  If you use a Kernel 2.4.x:
  #cp Makefile-2.4 Makefile
  If you use a Kernel 2.6.x:
  #cp Makefile-2.6 Makefile
- Build and load the device driver:
  #make
  #make install (this includes "insmod" of the device driver!)
- Now the device driver is compiled and loaded, now prepare the library:
  #cd {INSTALLPATH}/lib/
- Copy the two library files to /usr/lib:
  #make install (this includes "cp libsdkcxvx9.* /usr/lib")

2 API Reference

The Application Programming Interface exposed by the SDK library can be divided into several units and a set of common functions.

The following Units are available:

- Common Functions
- Unit Real-time Clock
- Unit LED Control
- Unit Temperature Sensors
- Unit Product Info
- Unit User EEPROM
- Unit Watchdog
- Unit GPIO

Any of these functions returns a result code as defined in the sdkresult.h header file. The sdkcxvx9.h file prototypes the library's functions and defines the parameter data types used for these functions. A source module using any of the functions only needs to

```
#include <usr/sdkcxvx9.h>
```

Be sure to add both the include path and the lib path to the appropriate search paths of your development environment (e.g. Makefile).

The function descriptions following are presenting the arguments with IN, OUT, or IN/OUT prepended. These terms describe whether the arguments are used as input and/or output parameters.

2.1 Common Functions

The Common Functions group gathers all functions not specific to a certain unit:

- sdkInit()
- sdkGetLibVersion()

2.1.1 sdkInit

Description:

Global user callable Init function for the SDK package. The sdkInit calls internal initialization routines for the various Units. It must be called prior to any other Unit's function. Note that functions in the Common group can be called without first executing sdkInit.

Prototype:

```
sdk_result    sdkInit (int InitValue);
```

Parameters:

IN InitValue	Reserved value, must be zero

Return:

sdk_result code	
SDK_OK	Okay, each module's init function successful
SDK_ERROR_PARAM	InitValue != 0 used
SDK_ERROR_INIT_xxx	Unit <xxx> init function error
SDK_ERROR_CONNECT	The device driver is not loaded or the sdkcx9_ctl device does not exist

Notes:

The header file sdkresult.h defines the SDK result codes. SDK_ERROR_INIT_xxx stands for an init error result code for unit <xxx> (e.g. SDK_ERROR_INIT_RTC designates an initialization problem with the Real-time clock unit).

2.1.2 sdkGetLibVersion

Description:

The function retrieves the SDK package library's version numbers. The version number format is in BCD in the form "m.nn". To display the version number on the screen with a printf, use the following:

printf("%x.%02x", major, minor);

This formatting will distinguish for example v1.40 from v1.04.

Prototype:

```
sdk_result    sdkGetLibVersion (Uint8 * pMajorVersion, Uint8 * pMinorVersion);
```

Parameters:

OUT pMajorVersion	Pointer to a variable receiving the major version number
OUT pMinorVersion	Pointer to a variable receiving the minor version number

Return:

sdk_result code	
SDK_OK	Success

Notes:

--

2.2 Unit RTC

The RTC functions read the system's date and time from the Real-time clock. The information read is presented as binary data values in a caller supplied buffer. Note that the RTC has a flag indicating if the time and date setting is valid. If the flag is not set (i.e. RTC settings not valid) the RTC module will not initialize successfully. In this case you have to enter a valid date and time in the system BIOS setup pages.

Functions Unit RTC:

- rtcGetTime()
- rtcGetDate()

2.2.1 rtcGetTime

Description:

The function reads the RTC and returns the hour:minute:second in the user buffer in three consecutive bytes, as follows

Buffer[0] - Seconds
Buffer[1] - Minutes
Buffer[2] - Hours

The user buffer must be at least 3 bytes in size. The time value returned is in binary 24-hours format.

Example:

Byte0	1	2
0x34	0x09	0x11 → 17:09:52

Prototype:

```
sdk_result    rtcGetTime (Uint8 Buffer[ ]);
```

Parameters:

OUT Buffer[ ]	User buffer for receiving the RTC time

Return:

sdk_result code	
SDK_OK	Reading okay
SDK_ERROR_NO_INIT	No successful init before
SDK_ERROR_UNAVAILABLE	Reading invalid

Notes:

This SDK function is using the standard Linux RTC device driver to get access to the RTC registers.

2.2.2 rtcGetDate

Description:

The function reads the RTC and returns the date in the user buffer in four consecutive bytes, as follows

Buffer[0] - Date of month
Buffer[1] - Month
Buffer[2] - Year
Buffer[3] - Century (19 or 20)

Example:

Byte0	1	2	3
0x12	0x0A	0x04	0x14 → October, 18 2004

The user buffer must be at least 4 bytes in size.

Prototype:

```
sdk_result    rtcGetDate (Uint8 Buffer[ ]);
```

Parameters:

OUT Buffer[ ]	User buffer for receiving the RTC date

Return:

sdk_result code	
SDK_OK	Reading okay
SDK_ERROR_NO_INIT	No successful init before
SDK_ERROR_UNAVAILABLE	Reading invalid

Notes:

This SDK function is using the standard Linux RTC device driver to get access to the RTC registers.

2.3 Unit LED Control

The LED on the Cx9/Vx9 can be set to three different colors: green, red, and a combination of both resulting in amber color. LED control is implemented as two logical LEDs as defined in the ledType enum type: ledGreen and ledRed. Each of the LEDs can be switched individually on, off, or can be set to one of six hardware blink modes. The settings defined in the ledMode enum type are as follows:

ledOff	LED switched off
ledOn	LED switched on

	Blink Rate	Duty Cycle
ledBlink1	0.25 Hz	12.5%
ledBlink2	0.5 Hz	25%
ledBlink3	1 Hz	50%
ledBlink4	2 Hz	50%
ledBlink5	3 Hz	50%
ledBlink6	4 Hz	50%

2.3.1 ledSet

Description:

The function specifies a new mode for the designated LED. Both *Led* and *Mode* can be specified as defined in the enum types ledType and ledMode, respectively. The mode of the other LED is not affected.

Prototype:

```
sdk_result    ledSet (ledType Led, ledMode Mode);
```

Parameters:

IN Led	LED to set
IN Mode	Mode to set

Return:

sdk_result code	
SDK_OK	Okay, new mode set for given LED
SDK_ERROR_NO_INIT	No successful init before
SDK_ERROR_PARAM	Parameter error

Notes:

--

2.3.2 ledGetStatus

Description:

The function returns the current mode for the designated LED. The *Led* argument selects the LED as defined in ledType, *pMode* must point to a variable of ledMode type.

Prototype:

```
sdk_result    ledGetStatus (ledType Led, ledMode * pMode);
```

Parameters:

IN Led	LED to get status for
OUT pMode	Pointer to a variable to receive Mode

Return:

sdk_result code	
SDK_OK	Okay, *pMode has current mode for given LED
SDK_ERROR_NO_INIT	No successful init before
SDK_ERROR_PARAM	Parameter error

Notes:

--

2.4 Unit Temperature Sensors

The Temperature unit is used to read the current temperature values on the Cx9/Vx9. It exports a single function. The Cx9/Vx9 measures three temperatures: one is the temperature directly on the CPU die, the two others are board temperatures. Both board temperature sensors are located under the heatsink, but without thermal coupling. The first board temperature sensor sits between the CPU and the Northbridge E7501. The second one is near of the PMC socket P6203. Which temperature to read is specified by the tempSensor enum type as defined in file sdkcxvx9.h. The tempSensor type enumerates the temperature sensors as follows:

Name	Sensor
tempSensorCpu	Sensor diode on the CPU die
tempSensorBoard1	Sensor U1985 (refer to Placement Plan in Hardware User's Manual)
tempSensorBoard2	Sensor U1986 (refer to Placement Plan in Hardware User's Manual)

2.4.1 tempReadSensor

Description:

The function returns the current temperature for the designated *Sensor*. The value read is returned in the integer variable *pTemp* is pointing to. The value is presented as signed integer in Degrees Celsius.

Prototype:

```
sdk_result    tempReadSensor (tempSensor Sensor, int * pTemp);
```

Parameters:

IN Sensor	Temperature sensor to read (tempSensor enum type)
OUT pTemp	Pointer to a variable to receive temperature reading

Return:

sdk_result code	
SDK_OK	Okay, *pTemp has a valid temperature reading
SDK_ERROR_NO_INIT	No successful init before
SDK_ERROR_PARAM	Parameter error
SDK_ERROR_BUSY	SMBus busy, retry

Notes:

--

2.5 Unit Product Info

The Cx9/Vx9 has a special serial EEPROM containing production data, such as Product ID, Serial Number and Board Revision. There may be another Product Info item if the board is an integral part of a complete system: the System Serial Number. The Product Info unit can be used to read this information using a single function: pinfGetItem(). Which item to read is determined by the pinfItemType enum type as defined in sdkcxvx9.h. The values read are presented as ASCIIZ strings, ready to printf to the screen. The following item types are defined:

SerialNo	Board serial number
ProductId	Product ID string showing the board's configuration as defined by the assembly options
BoardVers	Board hardware revision number

2.5.1 pinfGetItem

Description:

The function reads the data selected by the *Item* parameter out of the Factory EEPROM and stores them as NULL terminated string in the caller supplied *Buffer*. Before reading is performed, the buffer size is checked to ensure it is large enough. If it is not, the function returns the required size in the **pSize* variable.

Prototype:

```
sdk_result    pinfGetItem (pinfItemType Item, Uint8 Buffer[ ], Uint32 * pSize);
```

Parameters:

IN Item	Product info item to read
OUT Buffer	Buffer to receive the data read
IN/OUT pSize	Pointer to a variable holding buffer size upon function entry, and, on function return, required buffer size if buffer too small

Return:

sdk_result code	
SDK_OK	Okay, Buffer holds valid reading
SDK_ERROR_NO_INIT	No successful init before
SDK_ERROR_PARAM	Parameter error
SDK_ERROR_LENGTH	Buffer too small, *pSize contains size required
SDK_ERROR_UNAVAILABLE	Product info item not available
SDK_ERROR_BUSY	SMBus busy, retry

Notes:

--

2.6 Unit User EEPROM

The Cx9/Vx9 is equipped with a serial EEPROM for user purposes. The User EEPROM unit exposes a set of functions to write to and read from the user serial EEPROM. Any of the functions checks if the desired access is within the EEPROM boundaries, that is if both the start offset and access length do not reach beyond the upper end of serial EEPROM. There are functions which inhibit such an access and there are functions processing the request up to the EEPROM's end and returning SDK_WARNING_TRUNC result code.

Functions Unit User EEPROM:

- eprmGetSize()
- eprmGet()
- eprmSet()
- eprmFill()
- eprmReadByte()
- eprmReadWord()
- eprmReadDWord()
- eprmWriteByte()
- eprmWriteWord()
- eprmWriteDWord()

2.6.1 eprmGetSize

Description:

Retrieve the user EEPROM size in bytes. The value is written to the variable argument *pSize* is pointing to.

Prototype:

```
sdk_result    eprmGetSize (eprmType eeType, Uint32 * pSize);
```

Parameters:

IN eeType	EEPROM Type to get size
OUT pSize	Pointer to a variable receiving the serial EEPROM size

Return:

sdk_result code	
SDK_OK	Okay, Buffer holds valid reading
SDK_ERROR_NO_INIT	No successful init before
SDK_ERROR_PARAM	Parameter error

Notes:

--

2.6.2 eprmGet

Description:

Copy a number of bytes from the user EEPROM to the caller's buffer. The first byte copied is at *Offset* from the start of EEPROM. The user has to ensure that the buffer pointed to by *pBuffer* is large enough. The *pLength* parameter points to a variable serving two purposes: upon function entry, it holds the number of bytes to copy, upon function exit, it holds the number of bytes actually copied. Note that the function will always proceed data up to the last byte of EEPROM, not beyond. If the data processed is not of length specified by the user, a special warning result ("truncated") is returned.

Prototype:

```
sdk_result eprmGet (eprmType eeType, Uint32 Offset, void * pBuffer, Uint32 * pLength);
```

Parameters:

IN eeType	EEPROM Type to get
IN Offset	Start offset from EEPROM beginning
OUT pBuffer	Pointer to user buffer receiving copied data
IN/OUT pLength	Pointer to a variable specifying number of bytes to process and returning number of bytes actually processed

Return:

sdk_result code	
SDK_OK	Okay, buffer holds valid reading
SDK_ERROR_NO_INIT	No successful init before
SDK_ERROR_PARAM	Parameter error
SDK_WARNING_TRUNC	Warning: processed not as many bytes as specified by *pLength
SDK_ERROR_BUSY	SMBus busy, retry

Notes:

--

2.6.3 eprmSet

Description:

Copy a number of bytes from the caller's buffer to user EEPROM. The first byte copied to is at *Offset* from the start of the EEPROM. The user has to ensure that the buffer pointed to by *pBuffer* is large enough to hold at least the number of bytes specified by **pLength*. Handling of the *pLength* parameter is similar to the description in function eprmGet.

Prototype:

```
sdk_result eprmSet (eprmType eeType, Uint32 Offset, void * pBuffer, Uint32 * pLength);
```

Parameters:

IN eeType	EEPROM Type to set
IN Offset	Start offset from EEPROM beginning
IN pBuffer	Pointer to user buffer containing the data to copy
IN/OUT pLength	Pointer to a variable specifying number of bytes to process and returning number of bytes actually processed

Return:

sdk_result code	
SDK_OK	Okay, data in buffer copied successfully
SDK_ERROR_NO_INIT	No successful init before
SDK_ERROR_PARAM	Parameter error
SDK_WARNING_TRUNC	Warning: processed not as many bytes as specified by *pLength
SDK_ERROR_BUSY	SMBus busy, retry

Notes:

--

2.6.4 eprmFill

Description:

Fill a number of bytes in the user EEPROM with the *Char* value. This function can be used for initializing purposes. The first byte filled is at *Offset* from the EEPROM start. Handling of the *pLength* parameter is similar to the description in function eprmGet.

Prototype:

```
sdk_result    eprmFill (eprmType eeType, Uint32 Offset, Uint8 Char, Uint32 * pLength);
```

Parameters:

IN eeType	EEPROM Type to fill
IN Offset	Start offset from EEPROM beginning
IN Char	Value to fill EEPROM with
IN/OUT pLength	Pointer to a variable specifying number of bytes to fill and returning number of bytes actually filled

Return:

sdk_result code	
SDK_OK	Okay, EEPROM bytes filled
SDK_ERROR_NO_INIT	No successful init before
SDK_ERROR_PARAM	Parameter error
SDK_WARNING_TRUNC	Warning: filled not as many bytes as specified by *pLength
SDK_ERROR_BUSY	SMBus busy, retry

Notes:

--

2.6.5 eprmReadByte / Word / DWord

Description:

Read a byte/word/dword value from the user EEPROM at *Offset* into a variable in the caller's program context. The variable to store the value read is pointed to by the *pByte*/ *pWord/pDWord* parameter, respectively. The functions check the *Offset* parameter and size of the datum to read against the EEPROM length. If the access (or part of it) would exceeded the EEPROM's size, a "parameter error" will be returned.

Prototype:

```
sdk_result    eprmReadByte (eprmType eeType, Uint32 Offset, Uint8 * pByte);
```

```
sdk_result    eprmReadWord (eprmType eeType, Uint32 Offset, Uint16 * pWord);
```

```
sdk_result    eprmReadDWord (eprmType eeType, Uint32 Offset, Uint32 * pDWord);
```

Parameters:

IN eeType	EEPROM Type to read at
IN Offset	Offset in EEPROM to read at
OUT pByte / pWord / pDWord	Pointer to a variable receiving the datum read

Return:

sdk_result code	
SDK_OK	Okay, byte/word/dword read
SDK_ERROR_NO_INIT	No successful init before
SDK_ERROR_PARAM	Parameter error, access exceeds EEPROM boundaries
SDK_ERROR_BUSY	SMBus busy, retry

Notes:

--

2.6.6 eprmWriteByte / Word / DWord

Description:

Write a byte/word/dword value to the user EEPROM at *Offset*. The functions check the Offset parameter and size of the datum to write against the EEPROM length. If the access (or part of it) would exceeded the EEPROM's length, a "parameter error" will be returned.

Prototype:

```
sdk_result    eprmWriteByte (eprmType eeType, Uint32 Offset, Uint8 Byte);
```

```
sdk_result    eprmWriteWord (eprmType eeType, Uint32 Offset, Uint16 Word);
```

```
sdk_result    eprmWriteDWord (eprmType eeType, Uint32 Offset, Uint32 DWord);
```

Parameters:

IN eeType	EEPROM Type to write to
IN Offset	Offset in EEPROM to write to
IN Byte / Word / DWord	Data value to write

Return:

sdk_result code	
SDK_OK	Okay, byte/word/dword written
SDK_ERROR_NO_INIT	No successful init before
SDK_ERROR_PARAM	Parameter error, access exceeds EEPROM boundaries
SDK_ERROR_BUSY	SMBus busy, retry

Notes:

--

2.7 Unit Watchdog

The Watchdog Unit is used to recover from a hanging system or application without user intervention. Typical watchdog operation includes setting a certain timeout interval and starting the watchdog timer. The timer will start counting down immediately, and software must restart (or retrigger) the timer periodically, thus preventing the watchdog timeout interval from elapsing. When the system or the application stalls for some reason, the watchdog timer will count down to zero and then issue a system reset. Each time the watchdog timer is reset, counting down will start from the beginning.

The Cx9/Vx9 supports two watchdog timers. The first watchdog timer supports a variable timeout interval from 4,8s to 75,6s in steps of 1,2s. The second watchdog timer supports also a variable timeout interval. In steps of 1 min the timeout includes all values from 1 min to 255 min.

Note that Linux is not a real-time operating system. Therefore code execution is not deterministic, i.e. certain processes cannot be guaranteed to run at least once in a required period of time. This might lead to a watchdog-initiated system reset, especially when other tasks are given higher priorities than that of the watchdog timer restarting task.

Also note that resetting Linux asynchronously without a proper shut-down is always problematic and should be regarded as last resort for recovering from a critical error condition. Any unsaved data will be lost.

Functions Unit Watchdog:

- wdgGetStatus()
- wdgSetTime()
- wdgGetTime()
- wdgStart()
- wdgStop()
- wdgReTrigger()

2.7.1 wdgGetStatus

Description:

Return the current status for the watchdog *Timer* specified in the variable *pStatus* is pointing to. The wdgStatus type is defined in the sdkcxvx9.h file. The status can be checked by testing various bits. Currently the following bits are defined:

WDG_ENABLED	Watchdog currently running
WDG_RESET	Last system reset was issued by watchdog timeout condition

Prototype:

```
sdk_result    wdgGetStatus (wdgTimer Timer, wdgStatus * pStatus);
```

Parameters:

IN Timer	Watchdog timer to read status for
OUT pStatus	Pointer to a variable receiving the current status

Return:

sdk_result code	
SDK_OK	Success
SDK_ERROR_NO_INIT	No init prior to this function
SDK_ERROR_PARAM	Invalid timer or NULL pointer

Notes:

The WDG_RESET bit indicates if the most recent system reset was due to timeout of the *Timer* specified. Note that this bit is a copy of a bit set by hardware when the watchdog timer expires. When the SDK library starts up it stores the value of the bit internally and resets the hardware bit.

If the application using the SDK library exits, the library is removed from memory thus losing the reset bit information. A subsequent run of an application using the SDK library will therefore find the bit reset, thus indicating "last reset not initiated by watchdog".

2.7.2 wdgSetTime

Description:

Define the watchdog timeout interval without actually starting the watchdog timer. The timer to set the timeout interval for is specified with the *Timer* argument as defined in file sdkcxvx9.h.

In case of wdgTimer1 the possible timeout interval is between 4.8s and 75.6s. It is set by specifying a numerical value between 4 and 63. The timeout interval is calculated by scaling the value with 1.2s.

In case of wdgTimer2 the timeout interval is defined in minutes. The possible range is from 1 min to 255 min.

In either case, if the input value is out of range, it will be set to the smallest or largest possible value. The interval of both timers can only be set if the desired timer is not running. If the watchdog is already started a “busy” error will be returned.

Prototype:

```
sdk_result    wdgSetTime (wdgTimer Timer, Uint16 Value);
```

Parameters:

IN Timer	Timer to set time for (as defined in sdkcxvx9.h)
IN Value	Valid watchdog timeout value

Return:

sdk_result code	
SDK_OK	Success
SDK_ERROR_NO_INIT	No init prior to this function
SDK_ERROR_PARAM	No valid timer specified
SDK_ERROR_BUSY	Watchdog already running

Notes:

--

2.7.3 wdgGetTime

Description:

Read the current watchdog timeout interval for a specified *Timer*. The value read is returned in the variable *pValue* is pointing to.

For wdgTimer1 the value returned is the number representing the timeout interval in steps of 1.2s, for wdgTimer2 the value returned is the timeout interval in minutes.

Prototype:

```
sdk_result    wdgGetTime (wdgTimer Timer, Uint16 * pValue);
```

Parameters:

IN Timer	Timer to read value for (as defined in sdkcxvx9.h)
OUT pValue	Pointer to variable receiving the current watchdog timeout value

Return:

sdk_result code	
SDK_OK	Success
SDK_ERROR_NO_INIT	No init prior to this function
SDK_ERROR_PARAM	Null pointer passed or invalid timer

Notes:

--

2.7.4 wdgStart

Description:

Start the specified watchdog timer. The timer will immediately start counting down and the periodic retriggering has to start instantly. If the watchdog is already started a "busy" error will be returned.

Prototype:

```
sdk_result    wdgStart (wdgTimer Timer);
```

Parameters:

IN Timer	Watchdog timer to start

Return:

sdk_result code	
SDK_OK	OK - watchdog timer running
SDK_ERROR_NO_INIT	No init prior to this function
SDK_ERROR_PARAM	No valid timer specified
SDK_ERROR_BUSY	Watchdog already running

Notes:

--

2.7.5 wdgStop

Description:

Stop the running watchdog timer. The wdgStop command is accepted at any time, even if the watchdog is not running.

Prototype:

```
sdk_result    wdgStop (wdgTimer Timer);
```

Parameters:

IN Timer	Watchdog timer to stop

Return:

sdk_result code	
SDK_OK	OK - watchdog stopped
SDK_ERROR_NO_INIT	No init prior to this function
SDK_ERROR_PARAM	No valid timer specified

Notes:

Watchdog Timer 2 cannot be stopped once it is started. So Error Code 3 (Function not allowed) is returned if you execute wdgStop() for Watchdog Timer2.

2.7.6 wdgReTrigger

Description:

Retrigger the running watchdog, i.e. restart the watchdog timeout interval.

Prototype:

```
sdk_result    wdgReTrigger (wdgTimer Timer);
```

Parameters:

IN Timer	Watchdog timer to retrigger

Return:

sdk_result code	
SDK_OK	OK - watchdog timeout interval restarted
SDK_ERROR_NO_INIT	No init prior to this function
SDK_ERROR_PARAM	No valid timer specified
SDK_ERROR_INVALID	Specified watchdog timer not running

Notes:

--

2.8 Unit GPIO

The Cx9 in general, and the Vx9 as an assembly option, provide 8 pins for General Purpose I/O. Each pin can be configured independently by means of defining various attributes for the GP pin. These attributes are defined as bits in an *Attribute* value that can be written (Set) or read (Get) for a specified pin.

The attribute bits are defined in header file sdkcxvx9.h. The following table summarizes the attributes along with their values and function.

Attibute	Value / Function
GPIO_ATTR_PULLUP_ENABLE	1 A pull-up resistor in the GPIO line is enabled; this feature can be used for wired-OR signalling. 0 Pull-up resistor disabled.
GPIO_ATTR_OUT_ENABLE	1 The output driver in the circuitry of the GP pin is enabled; a value written to the output bit is driven to the pin according to the setting of attribute GPIO_ATTR_OUT_TYPE. 0 Output driver disabled; writings to the output bit do not drive the pin in any way.
GPIO_ATTR_OUT_TYPE	0 Open-drain: a 0 written to the bit is driving the output pin low, a 1 is setting the pin to high-impedance. 1 Push-pull: a 0 in the bit is driving the pin low, a 1 is driving the pin high.

For electrical information on the GPIO pins refer to the board's Hardware User's Manual. Note that there are bit combinations of less or no importance but yet possible. For instance, enabling PULLUP and setting the OUT_TYPE to Push-pull is not a reasonable usage. Also, defining a setting for the OUT_TYPE is not significant unless the OUT_ENABLE is set in order to activate the output driver of the pin.

The GP pin configuration functions operate on a single bit (or pin) specified as a numerical argument in the function calls, whereas the input/output functions take a mask value specifying the bits to read or write. For the gpioWrite() function, a set bit in the mask is specifying that the corresponding bit value in the *Value* argument be written to the output pin. For the gpioRead() function, a set bit in the mask causes the value of it's corresponding pin be copied to the *InValue* argument. Any bit not set in the mask is left unchanged on both input and output.

Availability of GPIO on the Vx9

On the Vx9 GPIO support is an assembly option. The device driver and the library are not able to programmatically detect whether GPIO is available or not since the GPIO hardware registers are always visible to software. The Vx9 product code informs if GPIO is assembled or not. Inspect letter 4 of the product code to determine if GPIOs are available: 0 = GPIO / 1 = TMDS

Example - Product Code for VR9 with GPIO option: VR9**0**B3C30AGC

Functions Unit GPIO:

- gpioSetPinAttributes()
- gpioGetPinAttributes()
- gpioWrite()
- gpioRead()

2.8.1 gpioSetPinAttributes

Description:

The function sets new attributes for the specified GPIO pin. The valid attributes are defined in the sdkcxvx9.h header file.

Prototype:

```
sdk_result    gpioGetPinAttributes (int PinNo, gpioAttrib Attributes);
```

Parameters:

IN PinNo	GPIO pin to set attributes for (0-based)
IN Attributes	Attributes to set for the specified pin

Return:

sdk_result code	
SDK_OK	Success
SDK_ERROR_NO_INIT	No init prior to this function
SDK_ERROR_PARAM	No valid GPIO pin specified

Notes:

The bits accepted are only those specified with the GPIO_VALID_ATTRIBUTES bit mask. Any other bit set in the *Attributes* argument is masked off before the attributes are used to set the GPIO pin's properties. Note that there is no error returned in this case.

2.8.2 gpioGetPinAttributes

Description:

The function reads the attributes defined for the specified GPIO pin. The attributes read are the bits specified with the GPIO_VALID_ATTRIBUTES bit mask in file sdkcxvx9.h. The attributes read are stored in the variable argument *pAttributes* is pointing to.

Prototype:

```
sdk_result    gpioGetPinAttributes (int PinNo, gpioAttrib * pAttributes);
```

Parameters:

IN PinNo	Pin to read attributes for (0-based)
OUT pAttributes	Pointer to a variable receiving the attributes read

Return:

sdk_result code	
SDK_OK	Success
SDK_ERROR_NO_INIT	No init prior to this function
SDK_ERROR_PARAM	No valid GPIO pin specified or NULL pointer passed

Notes:

--

2.8.3 gpioWrite

Description:

The function takes a mask and a value argument to write new values for GPO pins. A value of a bit in the *OutValue* argument is only written to the GPIO output register, if the corresponding bit in the *Mask* argument is set. If the corresponding mask bit is cleared the bit value in the output register is left unchanged. Valid bits in the *Mask* and *OutValue* argument are specified by the GPIO_VALID_PINS constant defined in sdkcxvx9.h .

Prototype:

```
sdk_result    gpioWrite (Uint32 Mask, Uint32 OutValue);
```

Parameters:

IN Mask	Mask specifying the bits affected
IN OutValue	Value specifying the state of each bit affected

Return:

sdk_result code	
SDK_OK	Success
SDK_ERROR_NO_INIT	No init prior to this function
SDK_ERROR_PARAM	No valid timeout value specified

Notes:

Note that the value of a certain pin can be written even if the pin is not defined as GPO pin, i.e. the GPIO_ATTR_OUT_ENABLE attribute is not set for the pin. If the output is not enabled, the value of the bit in the output register is not propagated to the pin itself.

2.8.4 gpioRead

Description:

The function reads bit values from the GPI pins and stores them in the variable pointed to by the *pInValue* argument. Which bits to read and store in the variable are specified with the *Mask* argument. If a mask bit is set, the corresponding bit is read from the GPIO input register and its value is stored in the *pInValue variable. Bits in the input variable with the corresponding bit cleared in the *Mask* argument are left untouched. Valid bits to read in the *Mask* argument are specified by the GPIO_VALID_PINS constant defined in sdkcxvx9.h .

Prototype:

```
sdk_result    gpioRead (Uint32 Mask, Uint32 * pInValue);
```

Parameters:

IN Mask	Mask specifying the bits affected
OUT pInValue	Pointer to a variable receiving the bit values read

Return:

sdk_result code	
SDK_OK	Success
SDK_ERROR_NO_INIT	No init prior to this function
SDK_ERROR_PARAM	No valid timeout value specified

Notes:

Note that a pin's value can be read even if the pin is defined as GPO pin, i.e. the GPIO_ATTR_OUT_ENABLE attribute is set for the pin. In this case the reading reflects the state of the pin, not the value written to the output register. For instance, if a GPO with output type open-drain is set locally ("1") and an external node would pull down the line, then the reading for this bit would be "0".

Error Report Form (Europe)

SBS Technologies GmbH & Co. KG

Memminger Str. 14

86159 Augsburg

Phone Number:**+49 821 5034-0**

Fax Number **+49 821 5034-119**

Company Name

Department

Contact Person

Mailing Address

Phone Number

Fax Number

Email Address

Part. N. _ _ _ _ _ _ _ _ _ _ Version: V _ . _ Date _ _ _ _ Serial N. _ _ _ _ _ _ _ _

Error Description:

Hardware Environment:

Operating System/Software:

Warranty repair: ❑ YES ❑ NO (Please see section 'Warranty')

Error Report Form (US)

SBS Technologies, Inc.
Embedded Computers
6301 Chapel Hill Road

Raleigh, NC 27607-5115

Phone Number: **+1 919 851-1101**

Fax Number **+1 919 851-2844**

Company Name

Department

Contact Person

Mailing Address

Phone Number

Fax Number

Email Address

Part. N. _ _ _ _ _ _ _ _ _ _ Version: V _ . _ Date _ _ _ _ Serial N. _ _ _ _ _ _ _ _

Error Description:

Hardware Environment:

Operating System/Software:

Warranty repair: ❑ YES ❑ NO (Please see section 'Warranty')