Bus Interface Library
(for ISA-BIB, PCI-BIB and SBus-BIB)

Programmer's Manual

DataCell Limited

v4.0.1

Disclaimer

While every precaution has been taken in the preparation of this manual, DataCell Ltd assumes no responsibility for errors or omissions. DataCell Ltd reserves the right to change the specification of the product described within this manual and the manual itself at any time without notice and without obligation of DataCell Ltd to notify any person of such revisions or changes.

Copyright Notice

Trademarks

"Apple", "Macintosh" and "MacOS" are trademarks of Apple Computer Inc. "AMCC" is a registered trademark of Applied Micro Circuits Corporation. "Dallas" is a registered trademark of Dallas Semiconductor Corporation. "Dell" is a registered trademark of Dell Computer Corporation. "Flash Graphics" and "X-32VM" are trademarks of Flashtek Limited. "IBM", "PC/AT", "PowerPC" and "VGA" are registered trademarks of International Business Machine Corporation. "MetroWerks" and "CodeWarrior" are registered trademarks of MetroWerks Inc. "Microsoft", "CodeView", "MS" and "MS-DOS", "Windows", "Windows NT", "Windows 95", "Windows 98", "Win32", "Visual C++" are trademarks or registered trademarks of Microsoft Corporation. "National Semiconductor" is a registered trademark of National Semiconductor Corporation. "Sun", "Ultra AX" and "Solaris" are registered trademarks of Sun Microsystems Inc. All "SPARC" trademarks are trademarks or registered trademarks of SPARC International Inc. "VxWorks" and "Tornado" are registered trademarks of Wind River Systems Inc. "Xilinx" is a registered trademark of Xilinx.
All other trademarks and registered trademarks are the property of their respective owners.

Part Information

Part Number: SNP-MAN-BASE-LIB

Version v4.0.1 September 1999

Printed in the United Kingdom.

Contact Details

Europe & ROW	Web	www.datacell.co.uk
	Sales	info@datacell.co.uk
	Support	techsupport@datacell.co.uk
USA	Web	www.datacell.com
	Sales	info@datacell.com
	Support	techsupport@datacell.com

Head Office:
DataCell Limited.
Falcon Business Park, 40 Ivanhoe Road,
Finchampstead, Berkshire, RG40 4QQ, UK

Tel +44 (0) 1189 324324
Fax +44 (0) 1189 324325

Table of Contents

Introduction

This document describes the 'BASE' library of functions. This library is used to initialize the Bus Interface Board and any Snapper plugged onto it. This library provides a platform and operating system independent API (Application Programming Interface) to any Bus Interface Board in the Snapper range.

The BASE Library can, in many ways, be considered as a high level driver for Snapper libraries, although in reality the individual drivers for the different operating systems (e.g. for Windows NT or Solaris etc) are at a lower level than this library.

An overview of how to use the library is given in the next section.

Function Overview

To initialize a Bus Interface Board with or without a Snapper module fitted, *BASE_create* is used and the handle returned is used to access the board. *BASE_get_parameter* is used to get the handle to the Snapper module (if fitted) and this is used by the Snapper libraries to access the Snapper module.

BASE_get_property can be used to read back information about the Bus Interface Board.

BASE_destroy is used to free the handle and the associated memory.

Error Returns

All of the BASE library functions return a *Terr* apart from several Boolean functions. *Terr* is a 32 bit unsigned integer, with the bit positions defined as follows:

31 to 24 Hardware identifier/revision (returned on error, otherwise 0 is returned). This is used to allow a top level calling function to determine the library in which the error occurred, and is actually read from the hardware itself.

Clearing bits 26 to 24 leaves the hardware identifier, and bits 26 to 24 give the hardware revision level.

23 to 16 This contains an error number, otherwise 0 if no error.

15 to 0 Function return value.

If a function call is successful, it returns *ASL_OK* (which is defined as 0) or the requested parameter. If an error occurs, an error number is returned in bits 23 to 16 along with the library identifier in bits 31 to 24. See the "Snapper Error Handling Programmer's Manual" in the Developer's Guide section of the Snapper Developer's Manual for more details on error returns.

Sample Application

The following code is a minimal program which uses a Snapper-DIG16 to capture an image from a Kodak Megaplus 1.4 camera and save it to a file. As with all sample code in this manual, error handling has been omitted for clarity, but apart from not handling errors cleanly this is a usable program. The Snapper SDK includes sample applications, both as executables and as source code, which provide a useful reference of 'real' code and are probably the best starting point for developing custom applications.

```
#include <asl_inc.h>

int main(ui16 argc, char** argv)
{
   Thandle Hdig16;       /* Handle to Snapper-DIG16 */
   Thandle Hbase;        /* Handle to baseboard */
   Thandle Hvid_image;   /* Handle to image */

   /* Initialize baseboard and Snapper module */
   Hbase = ASL_get_ret(BASE_create(BASE_AUTO));
   Hdig16 = BASE_get_parameter(Hbase, BASE_MODULE_HANDLE);
   Hvid_image = TMG_image_create();

   /* Set required Snapper mode */
   DIG16_initialize(Hdig16, DIG16_KODAK_MPLUS14, 8);

   /* Set up image parameters */
   DIG16_set_image(Hdig16, Hvid_image);

   /* Capture image and write it to a file */
   DIG16_capture_to_image(Hdig16, Hvid_image, DIG16_START_AND_WAIT);
   TMG_image_set_outfilename(Hvid_image, "image.tif");
   TMG_image_write(Hvid_image, TMG_NULL, TMG_TIFF, TMG_RUN);

   /* Free memory and exit */
   BASE_destroy(BASE_ALL_HANDLES);
   TMG_image_destroy(TMG_ALL_HANDLES);
}
```

Function List

BASIC FUNCTIONS

SERIAL COMMUNICATIONS FUNCTIONS

MEMORY HANDLING FUNCTIONS

MISCELLANEOUS FUNCTIONS

The functions are described in alphabetical order in the following pages.

BASE_create

USAGE

Terr BASE_create(ui16 base_address)

ARGUMENTS

base_address For ISA boards this parameter is the actual ISA bus address, for example, 0x300. For PCI and SBus boards the parameter can be *BASE_AUTO* or *BASE_DEVICE* | <*board number*>.

DESCRIPTION

This function initializes the Bus Interface Board and Snapper, and returns a unique handle to be used for accessing the board. For SBus and PCI Bus Interface Boards, *base_address* should be *BASE_AUTO* or *BASE_DEVICE*. For ISA Bus Interface Boards, *base_address* should be the actual hardware address corresponding to that set by the on-board jumpers J10, J11 and J12. (See the Snapper Installation Guide for more details on these jumper settings). The Snapper handle, needed for the Snapper libraries, can be retrieved using *BASE_get_parameter*.

BASE_AUTO will automatically open the first available PCI or SBus bus interface board. If one or more boards are already open, the next available board will be opened. Thus for opening multiple boards, either *BASE_AUTO* could be used repeatedly or *BASE_DEVICE*, 'OR'ed with the number of the board to open.

If using *BASE_DEVICE* note that the device number does not always match the physical slot number on the motherboard. This is at the discretion of the hardware manufacturer.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_INCOMPATIBLE_LIBRARIES	The different Snapper libraries and drivers all contain version information. On calling *BASE_create*, the main Snapper library compares version information with the other libraries and returns an error if they do not match. If an application receives this error it means that the current Snapper installation is corrupt and must be re-installed.
ASLERR_NOT_SUPPORTED	This error is generated if the application attempts to create more than one handle reference to a single hardware instance.
ASLERR_OUT_OF_MEMORY	This error is generated if there is insufficient system memory available to contain the various internal structures associated with the Base and Snapper libraries.

As *BASE_create* automatically configures and initializes a Snapper module that is fitted to it, this function may return error messages associated with initializing a Snapper module.

EXAMPLES

The following code initializes a Bus Interface Board (PCI or SBus) and gets the respective handles to the Bus Interface Board and the Snapper.

```
Thandle hSnapper;        /* Handle to Snapper-DIG16 */
Thandle hBaseboard;      /* Handle to baseboard */

/* Initialize baseboard and Snapper module */
if ( ASL_is_err(hBaseboard = BASE_create(BASE_AUTO)) )
```

```
   exit(0);   /* failed to find/initialize board */
else
   hSnapper = BASE_get_parameter(Hbase, BASE_MODULE_HANDLE);

/* we now have valid handles to access the hardware */
```

The following code fragment initializes the second PCI or SBus device:

```
if ( ASL_is_err(hBaseboard = BASE_create(BASE_DEVICE | 2)) )
   exit(0);   /* failed to find/initialize board */
else
   hSnapper = BASE_get_parameter(Hbase, BASE_MODULE_HANDLE);
```

BUGS / NOTES

The <baseboard handle> is returned in the lower 16 bits, if successful.

There are no known bugs.

SEE ALSO

BASE_destroy, *BASE_get_parameter*.

BASE_delay

USAGE

Terr BASE_delay(Thandle Hbase, ui32 dwMicroSecs)

ARGUMENTS

Hbase	Handle to a Bus Interface Board.
dwMicroSecs	Timed delay in micro-seconds.

DESCRIPTION

This function generates a timed delay in micro-seconds. However the granularity of the timer is operating system dependent, and small delay requests may generate significantly larger actual timed delays.

The difference between *BASE_delay* and *BASE_set_timer* is that the latter uses Snapper hardware resources, but is not available on all Bus Interface Boards.

RETURNS

This function returns the following codes:

ASL_OK	If successful.

EXAMPLES

To use the software timer in conjunction with a Snapper-24 to enable an active high pulse of 1ms:

```
/* Generate 1ms pulse */
SNP24_set_trigger(Hsnp24, SNP24_TRIG_OUT_HI);
BASE_delay(Hbase, (ui32) 1000);
SNP24_set_trigger(Hsnp24, SNP24_TRIG_OUT_LO);
```

BUGS / NOTES

Not only does the granularity of the time delay depend upon the specific operating system, but so also does the behaviour. In threaded operating systems, (ie Unix, Windows NT, etc) the delay does not use any system resources and frees the CPU to perform other processing. However on non threaded operating systems, the CPU will not be available to perform other processing.

There are no known bugs.

SEE ALSO

BASE_destroy, *BASE_get_parameter*.

BASE_destroy

USAGE

Terr BASE_destroy(Thandle Hbase)

ARGUMENTS

Hbase	Handle to a Bus Interface Board or *BASE_ALL_HANDLES* to destroy all handles.

DESCRIPTION

Frees the memory associated with the Bus Interface Board structure, *Hbase*, and any associated Snapper module. The handle is deassigned.

BASE_ALL_HANDLES is a useful parameter to use instead of the handle to destroy all Bus Interface Board and Snapper handles with one function call.

This function should be called on program exit (or when the Bus Interface Board is no longer needed) to free the memory allocated in internal structures.

RETURNS

This function returns the following codes:

ASL_OK	If successful. As *BASE_DESTROY* is generally the last function to be called, it does not generate any errors as there is nothing left for an application to do.

EXAMPLES

The following code fragment destroys an individual board handle (i.e. a Bus Interface Board and its associated Snapper):

```
Thandle hBaseboard = BASE_NULL_HANDLE;

/* Initialize and use the Snapper */
.
.

/* Then free the memory etc when we've finished */
BASE_destroy(hBaseboard);
hBaseboard = BASE_NULL_HANDLE;   /* good practice to track status */
```

The following code fragment destroys all Bus Interface Board handles and all Snapper handles:

```
BASE_destroy(BASE_ALL_HANDLES);
```

BUGS / NOTES

There are no known bugs.

SEE ALSO

BASE_create.

BASE_get_parameter

USAGE

Terr BASE_get_parameter(Thandle Hbase, ui16 parameter)

ARGUMENTS

Hbase	Handle to a Bus Interface Board.
parameter	The parameter to return.

DESCRIPTION

This function returns various parameters from the internal structure associated with the Bus Interface Board handle.

The parameters are as follows:

BASE_BASEBOARD_TYPE	This returns the Bus Interface Board type and can be one of the following (Type *ui16*):
	BASE_PCI_BIB — PCI Bus Interface Board, including combined single board Snappers, ie Snapper-PCI-SNP24, Snapper-PMC-DIG16, etc
	BASE_ISA_BIB — ISA Bus Interface Board.
	BASE_ISA_JPG — ISA with JPEG compression - "Crunch-ISA".
	BASE_SBUS_BIB — SBus Bus Interface Board.
BASE_LIBRARY_REV_LEVEL	This returns the revision level of the combined BASE and Snapper libraries as a 32 bit unsigned integer. The format is M.mm.ss , ie Major.minor.sub-minor. For example 32002 means 3.2 (Rev. 2). (Type *ui32).*
BASE_ID_VALUE	This returns the ID as read from the hardware, which distinguishes between all the different baseboard variants. (Type *ui8*).
BASE_REV_VALUE	This returns the board revision as read from the hardware, which distinguishes between the hardware revisions of the particular baseboard. (Type *ui8*).
BASE_IDREV_VALUE	This returns the ID and board revision as read from the hardware, which distinguishes between the different hardware revisions of the different baseboard variants. (Type *ui8*).
BASE_MODULE_HANDLE	The handle to the Snapper module. (Type *Thandle, ui32).*
BASE_MODULE_IDREV	The hardware identifier and revision level of the Snapper module. The "IDrev" is 8 bits - the upper 5 represent the ID (unique to each board) and the lower 3 the revision level. (Type *ui8*).
BASE_MODULE_FAMILY_VALUE	This returns the module family and can be one of the following (Type *ui8*):
	DIG16_FAMILY_ID — Any rev of Snapper-DIG16 or Snapper-PMC-DIG16
	SNP24_FAMILY_ID — Any rev of Snapper-24, Snapper-PCI24 or Snapper-PMC24.
	SNP16_FAMILY_ID — Any rev of Snapper-16 or Snapper-PCI16.
	SNP8_FAMILY_ID — Any rev of Snapper-8, Snapper-PCI8 or Snapper-PMC8.

BASE_IRQ	This parameter is set by the driver and represents the interrupt level that has been allocated to the baseboard. It does not necessarily match the physical interrupt, ie on an x86 PCI system the baseboard may be assigned physical interrupt 12, but BASE_IRQ may return a number different to this.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The baseboard handle is invalid.
ASLERR_BAD_PARAM	The parameter value is invalid.
ASLERR_NOT_RECOGNIZED	The ID value read back from the Snapper or baseboard hardware is not recognized.

EXAMPLES

The following code initializes a Bus Interface Board (PCI or SBus) and determines which family of Snapper boards is fitted.

```
Thandle hBaseboard;      /* Handle to baseboard */
ui8 bSnapperFamily;

/* Initialize baseboard and Snapper module */
if ( ASL_is_err(hBaseboard = BASE_create(BASE_AUTO)) )
   exit(0);   /* failed to find/initialize board */

/* Determine which Snapper family is fitted */
bSnapperFamily = BASE_get_parameter(Hbase, BASE_MODULE_FAMILY_VALUE);

switch ( (int) bSnapperFamily )
{
   case (int) SNP8_FAMILY_ID:
      printf("Snapper-8 type board found"); break;
   case (int) SNP16_FAMILY_ID:
      printf("Snapper-16 type board found"); break;
   case (int) SNP24_FAMILY_ID:
      printf("Snapper-24 type board found"); break;
   case (int) DIG16_FAMILY_ID:
      printf("Snapper-DIG16 type board found"); break;
   default:
      printf("New Snapper family requires software upgrade"); break;
}
```

BUGS / NOTES

The function returns a type *Terr* (*ui32* - an unsigned 32 bit integer). Therefore a cast may be need depending on the parameter type (given above for each parameter).

There are no known bugs.

SEE ALSO

BASE_get_property.

BASE_get_property

USAGE

*Terr BASE_get_property(Thandle Hbase, char *property, char *value)*

ARGUMENTS

Hbase	Handle to a Bus Interface Board.
property	A character string or name of the property to access.
value	The property result string. (Must point to a buffer of at least 128 bytes.)

DESCRIPTION

This function returns various property strings associated with the Bus Interface Board. Not all property strings are supported on all Bus Interface Boards. All the property strings are stored in PROM or NVRAM (non-volatile RAM) on the Bus Interface Board, apart from the "mapdate" property, which is read from the design currently in use for the data mapper FPGA (Field Programmable Gate Array).

PROPERTY

The property strings are as follows:

"mapdate"	**Data Mapper Date**: This retrieves the date and time string associated with the current data mapper in use. It is unlikely that this function will ever be needed, but it can be useful to solve technical support issues. (i.e. the date string is used as a revision level.). The data mapper performs pixel mapping in hardware to enable high speed display update.
"mapfpga_name"	**Data Mapper Name**: This retrieves the design name of the current data mapper in use. It is unlikely that this function will ever be needed, but it can be useful to solve technical support issues. Typical names are of the form "da00020a.rbt".
"mapfpga"	**Data Mapper FPGA**: This gives the type and speed of the "Data Mapper" FPGA (field programmable gate array). The software drivers may use this information to determine the maximum speed at which data can be acquired. For example, "3142A-4" means a Xilinx XC3142A with speed grade '-4'.
"micro"	**Microcontroller**: This gives the type and nominal clock speed of the microcontroller fitted to the Bus Interface Board. The clock speed may be relevant when using serial communications, as the serial interface may be embedded in the microcontroller . This 'master' clock speed is factory set, but it can be used to generate a variety of communication baud rates. For example, "80C320-18.4" means a Dallas 80C320 microcontroller with an 18.432 MHz clock.
"micro_revlevel"	**Microcontroller Revision Level**: This gives the revision of the microcontroller firmware fitted to the Bus Interface Board.
"revlevel"	**Revision Level**: This gives the overall revision level of the board, for example "2.1".
"snapclk"	**Snapper Clock**: This gives the type of Snapper clock generator provided on the Bus Interface Board and the nominal master frequency. This clock generator is configured by the software drivers to generate a Snapper read out frequency optimised to the type of Snapper/Bus Interface Board combination. For example, "9107-03-14.3" means the device type is a 9107-3 with a 14.31818MHz clock. "PCI" means that the Snapper clock is generated from the PCI clock.
"pcidev"	**PCI Interface Device:** This is used to determine the type and revision of the PCI Interface device. For example "S5933QE" means AMCC S5933 PCI Interface Device, die revision "QE". This parameter is only supported on later PCI boards.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The baseboard handle is invalid.
ASLERR_NOT_IMPLEMENTED	The parameter value is invalid or not supported on this baseboard.

EXAMPLES

The following code fragment prints out all the properties:

```
char szPropery[256];

BASE_get_property(hBase, "revlevel", szPropery);
printf("Revision Level String: %s", szPropery);

BASE_get_property(hBase, "micro", szPropery);
printf("Microprocessor: %s", szPropery);

BASE_get_property(hBase, "micro_revlevel", szPropery);
printf("Microprocessor Revision Level: %s", szPropery);

BASE_get_property(hBase, "mapfpga", szPropery);
printf("Data Mapper FPGA: %s", szPropery);

BASE_get_property(hBase, "mapfpga_name", szPropery);
printf("Data Mapper Name: %s", szPropery);

BASE_get_property(hBase, "mapdate", szPropery);
printf("Data Mapper Rev: %s", szPropery);

BASE_get_property(hBase, "snapclk", szPropery);
printf("Snapper Clock: %s", szPropery);
```

BUGS / NOTES

There are no known bugs.

SEE ALSO

BASE_get_parameter.

BASE_PhysMem_Lock

USAGE

*Terr BASE_PhysMem_Lock(Thandle Hbase, void *pData, ui32 dwByteCount, ui32 **ppSGTable)*

ARGUMENTS

Hbase	Handle to a Bus Interface Board.
pData	Pointer to a virtual memory buffer allocated by the application.
dwByteCount	Byte count of the virtual memory buffer pointed to by *pData*.
ppSGTable	The address of a pointer whose value will be set to the address of a scatter-gather table on completion.

DESCRIPTION

This function takes the virtual memory buffer pointed to by *pData*, and forces it to be resident in system physical memory or "locked down". It then allocates and initializes a buffer containing a structure of physical address information, ie a scatter-gather table. Finally the pointer referenced by *ppSGTable* is set to point to the base address of the scatter-gather table.

This method can be used to ensure that the virtual memory buffer is resident in physical memory before acquisition commences. If the acquisition is running in a loop processing a number of images using locked down memory, using *BASE_PhysMem_Lock* eliminates the need to generate a physical address mapping from a virtual memory buffer each time the virtual memory buffer is accessed.

BASE_PhysMem_Unlock frees the scatter-gather table allocated by the function. The data structure of the table, and macros to access it, are described in the file, "shared.h"

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The baseboard handle is invalid.
ASLERR_OUT_OF_MEMORY	This error is generated if there is insufficient system memory available to contain the Scatter-Gather table.
ASLERR_SYSTEM_CALL_FAILED	The call to the underlying operating system to lock the virtual memory buffer in physical memory, or to generate the Scatter-Gather table failed.

EXAMPLES

The Snapper API is the same, except that *BASE_set_ptr* is used to send the target memory location information to the driver. (The fact the internal structure member that holds the pointer will then not be *NULL* is used by the library/driver to know what to do.)

The essential steps are:

- Build the scatter/gather table for the target device.
- Use *BASE_set_ptr* to give the Snapper driver access to the table. If this entry is *NULL*, then the driver will DMA to host memory as usual.
- Use the DIG16 API as usual.

The following code is an example of this.

```
{
```

```
    ui8  *pbData;
    ui32  dwNumBytes = 768*484;  /* For testing with Pulnix 9700 */
    ui32 *pSGTable;

    pbData = (ui8*) _ASL_Malloc(dwNumBytes);  /* Calls malloc */

   /* Lock memory (and return scatter/gather table) note this function
    * allocates memory for the scatter/gather table and BASE_PhysMem_Unlock
    * frees it.
    */
    BASE_PhysMem_Lock(D16.m_hBase, (void*) pbData, dwNumBytes, &pSGTable);

    /* Point D16.m_hSrcImage to our (physically locked) data area */
    TMG_image_set_ptr(D16.m_hSrcImage, TMG_IMAGE_DATA, (void*) pbData);

    /* Now protect the memory so that library will not re-allocate it */
    TMG_image_set_flags(D16.m_hSrcImage, TMG_LOCKED, TRUE);

    /* Give the Snapper driver access to our physical device scatter-gather
     * table
     */
    BASE_set_ptr(D16.m_hBase, BASE_SG_TABLE, pSGTable);

    /* Finally capture an image */
    DIG16_capture_to_image(hSnapper, hSrcImage, DIG16_START_AND_WAIT);

    /* Process the image as required */
    …

    /* Tidy up - free physical memory etc */
    BASE_PhysMem_Unlock(D16.m_hBase, (void*) pbData, pSGTable);
    BASE_set_ptr(D16.m_hBase, BASE_SG_TABLE, NULL);
    _ASL_Free(pbData);
    TMG_image_set_ptr(D16.m_hSrcImage, TMG_IMAGE_DATA, (void*) NULL);

    return;
}
```

Memory may be allocated by the user application and locked by the Snapper driver as follows. The Snapper driver returns a scatter/gather table of addresses and lengths. Note only one area of memory (per Snapper handle) may be locked at any one time. The scatter/gather table format is described in "shared.h".

```
int   n;
ui8  *pbData;
ui32  dwNumBytes = 100000;
ui32 *pSGTable;   /* User's DMA Scatter/gather table */

pbData = (ui8*) _ASL_Malloc(dwNumBytes);  /* Our buffer */

BASE_PhysMem_Lock(D16.m_hBase, (void*) pbData, dwNumBytes, &pSGTable);

printf("Allocated size = %d bytes\n",
       _SGTable_Get_AllocatedSizeBytes(pSGTable) );
printf("Num Entries = %d\n", _SGTable_Get_NumEntries(pSGTable) );
printf("Transfer size = %d bytes\n", _SGTable_Get_TransferSizeBytes(pSGTable)
       );

for (n = 0; n < _SGTable_Get_NumEntries(pSGTable); n++)
{
   printf("PhysAddress[%d] = %d\n", n, _SGTable_Get_PhysAddress(pSGTable, n)
       );
   printf("PhysLength[%d]  = %d\n", n, _SGTable_Get_PhysLength(pSGTable, n) );
}
```

```
BASE_PhysMem_Unlock(D16.m_hBase, (void*) pbData, pSGTable);
_ASL_Free(pbData);
```

BUGS / NOTES

It is only possible to generate a single Scatter-Gather table, even if the application requires more than 1 virtual memory image buffer. However, it is possible to generate a single table which contains sufficient memory allocation for all the required buffers, and then only reference part of the buffer for each image.

This function is currently only available on Windows NT. However the nature of VxWorks means that this function is not required – see the "VxWorks Programmer's Manual" for a description of *ALenLists*.

SEE ALSO

BASE_PhysMem_Unlock.

BASE_PhysMem_Unlock

USAGE

*Terr BASE_PhysMem_Unlock(Thandle Hbase, void *pData, ui32 **ppSGTable)*

ARGUMENTS

Hbase	Handle to a Bus Interface Board.
pData	Pointer to a virtual memory buffer allocated by the application.
ppSGTable	The address of a pointer whose value contains the address of an existing Scatter-Gather table.

DESCRIPTION

This function unlocks the virtual memory buffer pointed to by *pData*, but does not free the data space. It also de-allocates the scatter-gather table and frees the memory space associated with it.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The baseboard handle is invalid.
ASLERR_SYSTEM_CALL_FAILED	The call to the underlying operating system to unlock the virtual memory buffer in physical memory, or to de-allocate the Scatter-Gather table failed.

EXAMPLES

See *BASE_PhysMem_Lock* for an example.

BUGS / NOTES

This function is currently only available on Windows NT. However the nature of VxWorks means that this function is not required – see the "VxWorks Programmer's Manual" for a description of *ALenLists*.

SEE ALSO

BASE_PhysMem_Lock.

BASE_serial_get_parameter

USAGE

ui32 BASE_serial_get_parameter(Thandle Hbase, ui32 parameter)

ARGUMENTS

Himage	Handle to a Bus Interface Board.
parameter	Parameter type.

DESCRIPTION

This function returns the current serial setting as selected by *parameter*. The parameter is always returned as a 32 bit unsigned integer although some of the parameters are stored as 16 bit unsigned integers internally.

PARAMETER

The parameter takes the following values:

BASE_SERIAL_XON_XOFF_STATUS	This option reads the XON/XOFF status. If TRUE, then data may be transmitted and if FALSE then it cannot (i.e. an XOFF has been received).
BASE_SERIAL_RX_COUNT	This option reads the number of characters present in the receive buffer. Any characters in the hardware receive buffer will be flushed into the software buffer in the driver as part of making this request. Calling *BASE_serial_set_parameter* with parameter *BASE_SERIAL_INIT* may be used to clear down this count to zero as part of resetting the serial port.
BASE_SERIAL_PARITY_ERRORS	This option reads the number of parity errors that have occurred since the serial port was initialised. The parity error count is incremented whenever a parity error is received, irrespective of whether *BASE_SERIAL_PARITY_IGNORE* is set to *TRUE* or *FALSE*. It is initialised to zero by *BASE_create* or a call to *BASE_serial_set_parameter* with *BASE_SERIAL_INIT*.
BASE_SERIAL_OVERFLOW_ERRORS	This option reads the number of overflow errors that have occurred since the serial port was initialised. It is initialised to zero by *BASE_create* or a call to *BASE_serial_set_parameter* with *BASE_SERIAL_INIT*.
BASE_SERIAL_BAUDRATE	The current setting of the baud rate.
BASE_SERIAL_DATA_BITS	The current setting of the number of data bits: 7 or 8.
BASE_SERIAL_STOP_BITS	The current setting of the number of stop bits: 1 or 2.
BASE_SERIAL_XON_XOFF_ENABLE	The current state of the XON/XOFF flow control; returns *TRUE* if XON/XOFF flow control is enabled, *FALSE* otherwise.
BASE_SERIAL_XON_CHAR	The current XON character.
BASE_SERIAL_XOFF_CHAR	The current XOFF character.

BASE_SERIAL_PARITY_MODE	Allows the following parameters that apply to both transmit and receive: *BASE_SERIAL_ODD* *BASE_SERIAL_EVEN* *BASE_SERIAL_MARK* (always '1') *BASE_SERIAL_SPACE* (always '0') *BASE_SERIAL_NONE* (default).
BASE_SERIAL_PARITY_IGNORE	The current setting of the parity control flag; returns *TRUE* if parity is ignored, *FALSE* otherwise.

RETURNS

The parameter selected by *parameter* as an unsigned 32 bit integer (*ui32*).

EXAMPLES

The following code fragment sets then reads the baud rate setting:

```
BASE_serial_set_parameter(hBase, BASE_SERIAL_BAUDRATE, 19200);
BaudRate = BASE_serial_get_parameter(hBase, BASE_SERIAL_BAUDRATE);
printf("Requested 19200, set %d\n", BaudRate);
```

BUGS / NOTES

There are no known bugs.

SEE ALSO

BASE_serial_set_parameter.

BASE_serial_set_parameter

USAGE

Terr BASE_serial_set_parameter(Thandle Hbase, ui32 parameter, ui32 value)

ARGUMENTS

Hbase	Handle to a Bus Interface Board.
parameter	The parameter to set.
value	The value of the parameter to set.

DESCRIPTION

This function sets the serial control parameters on the Bus Interface Board referenced by *Hbase*. Serial communications are controlled by a microcontroller on the Bus Interface Board.

PARAMETER

The parameter takes the following values:

BASE_SERIAL_INIT	This option resets the serial port and sets all the parameters to their default option. *value* should be set to 0.
BASE_SERIAL_BAUDRATE	The baud rate of the device. See options below (default 9600).
BASE_SERIAL_DATA_BITS	The number of data bits: 7 or 8 (default 8).
BASE_SERIAL_STOP_BITS	The number of stop bits: 1 or 2 (default 1).
BASE_SERIAL_XON_XOFF_ENABLE	Enable or disable XON/XOFF flow control. Set to *TRUE* to enable and *FALSE* to disable. The default is disabled.
BASE_SERIAL_XON_CHAR	This allows the user to change the XON character from the default of 17.
BASE_SERIAL_XOFF_CHAR	This allows the user to change the XOFF character from the default of 19.
BASE_SERIAL_PARITY_MODE	Allows the following parameters that apply to both transmit and receive: *BASE_SERIAL_ODD* *BASE_SERIAL_EVEN* *BASE_SERIAL_MARK* (always '1') *BASE_SERIAL_SPACE* (always '0') *BASE_SERIAL_NONE* (default).
BASE_SERIAL_PARITY_IGNORE	This is set to *TRUE* or *FALSE* (default) to control the behaviour of *BASE_serial_receive_buffer*.

The frequency of the crystal fitted can be determined using the function *BASE_get_property*.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The baseboard handle is invalid.
ASLERR_BAD_PARAMETER	The parameter value is invalid.
ASLERR_NOT_SUPPORTED	The required baud rate is not available on the baseboard.

EXAMPLES

This example checks the crystal frequency, then sets the baud rate to 125k:

```
BASE_get_property(hBase, "micro", String);  /* Read the crystal type */
pString = strrchr( String, "-" );     /* e.g. String = "80C320-16.0" */
if ((pString[1] == '1') && (pString[2] == '6'))
   BASE_serial_set_parameter(hBase, BASE_SERIAL_BAUDRATE, 125000);
else
   printf("125K baud not available - micro: %s", String);
```

BUGS / NOTES

The supported baud rates depend on the frequency of the crystal fitted to the board. The standard crystal fitted is 18.432 MHz which provides the following baud rates: 9600, 14400, 19200, 28800 and 57600. Two special build options are available; 22.1184 MHz crystal may be fitted to provide 9600, 14400, 19200, 28800, 38400, 57600, 115200 or 16 MHz crystal may be fitted to provide 9600, 19200 and 125000 (although the actual frequencies are 19231 and 9615 for 19200 and 9600 respectively).

Serial communications are not available on ISA Bus Interface Boards.

SEE ALSO

BASE_get_parameter, *BASE_serial_transmit_buffer*.

BASE_serial_receive_buffer

USAGE

*Terr BASE_serial_receive_buffer(Thandle Hbase, ui8 *pBuffer, ui32 *pRxCount, ui32 Timeout)*

ARGUMENTS

Hbase	Handle to a Bus Interface Board.
pBuffer	Pointer to a receive character buffer.
pRxCount	Pointer to the number of characters to be read and placed into the receive buffer.
Timeout	Timeout in milliseconds.

DESCRIPTION

This function receives a stream of characters and places them into the buffer pointed to by *pBuffer*. The number of characters requested is determined by the contents of a 32 bit unsigned integer pointed to by *pRxCount*. *pRxCount* will be updated with the number of characters successfully received. If less than the requested number of characters are successfully received, then the warning code *ASLWRN_TIMEOUT* will be returned.

If *Timeout* is set to 0, then the function will wait indefinitely until all characters have been received, unless an error occurs, in which case the function will return with the appropriate error code.

If *Timeout* is set to 1, then the function will return immediately with the number of characters already present in the driver/hardware serial receive buffer. (Any characters required to be read from the hardware buffer will be read out as necessary in order to return the requested number of characters.) If no characters are present the function will return immediately with a receive count of zero.

If a timeout occurs, the function will return with a timeout warning.

If the XON/XOFF protocol is being used, these characters are interpreted in the driver/hardware and are not counted as received characters.

If a parity error occurs and *BASE_SERIAL_PARITY_IGNORE* is set to *FALSE* (see *BASE_serial_set_parameter*), the function returns *ASLERR_PARITY* and the number of characters successfully read up to and including the character with the parity error. The parity error count is incremented irrespective of the state of *BASE_SERIAL_PARITY_IGNORE* and can be read using *BASE_serial_get_parameter*. A further call to *BASE_serial_receive_buffer* is necessary to read the additional characters that were originally required, after the character that generated the parity error.

If an overflow error occurs, the function returns with the error *ASLERR_OVERFLOW*. If errors occur simultaneously, overflow errors take precedent over parity which takes precedent over underflow.

RETURNS

This function returns the following codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The baseboard handle is invalid.
ASLWRN_TIMEOUT	The timeout expired before the required number of characters were received. This is NOT an error, and will not generate a call to the default error handler. This is necessary as some serial protocols require receive calls to timeout to determine the status of the serial link.

EXAMPLES

This example receives a single character:

```
char pBuffer[256];
ui32 Count;
ui32 *pCount;

pCount = &Count;   /* set up pointer */
Count = 1;         /* 1 characters to receive */
res = BASE_serial_receive_buffer(hBase, pBuffer, pCount, 0);
if ( res == ASL_OK )
   printf("Successfully received: %c", pBuffer[0]);
```

BUGS / NOTES

Under MS-DOS and Windows 3.1, the only acceptable values of *Timeout* are 0 and 1.

Only one thread may be in the receive buffer call at any one time. Subsequent calls will generate a busy error and return.

The timeout value must be carefully chosen, allowing for the current baud rate and possible added delays due to flow control events.

Note that if binary data is being transmitted or received then XON/XOFF flow control must not be used (i.e. because valid binary data may contain XON/XOFF characters).

SEE ALSO

BASE_serial_transmit_buffer, *BASE_serial_set_parameter*.

BASE_serial_transmit_buffer

USAGE

*Terr BASE_serial_transmit_buffer(Thandle Hbase, ui8 *pBuffer, ui32 *pTxCount, ui32 Timeout)*

ARGUMENTS

Hbase	Handle to a Bus Interface Board.
pBuffer	Pointer to a character buffer.
pTxCount	Pointer to the number of characters in the buffer to transmit.
Timeout	Timeout in milliseconds.

DESCRIPTION

This function transmits a stream of characters from the buffer pointed to by *pBuffer*. The number of characters is determined by the contents of a 32 bit unsigned integer pointed to by *pTxCount*. The function will not return until the whole buffer has been sent or a timeout occurs. *pTxCount* will be updated with the number of characters successfully sent.

If *Timeout* is set to 0, then the function will wait indefinitely until all characters have been transmitted. The event that may cause a transmit delay in this instance would be the reception of an XOFF character whilst transmitting.

If *Timeout* is set to 1, then the function will return immediately if it cannot transmit (with the error code *ASLERR_TIMEOUT*) or with the number of characters successfully transmitted before XOFF was received.

Any characters received whilst transmitting the contents of the buffer will be buffered up in the driver (and/or hardware receive buffer) ready to be read by *BASE_serial_receive_buffer*.

If a timeout occurs, the transmission is aborted, and an error status is returned.

RETURNS

This function returns the following codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The baseboard handle is invalid.
ASLERR_SYSTEM_CALL_FAILED	The timeout expired before the required number of characters were transmitted.

EXAMPLES

This example sets up and transmits a buffer containing an imaginary camera command:

```
char pCameraOnCommand[] = "CamOn";
ui32 Count;
ui32 *pCount;

pCount = &Count;   /* set up pointer */
Count = 5;         /* 5 characters to transmit */
res = BASE_serial_transmit_buffer(hBase, pCameraOnCommand, pCount, 0);
if ( res == ASL_OK )
   printf("Successfully sent string %s", pCameraOnCommand);
```

BUGS / NOTES

Under MS-DOS and Windows 3.1, the only acceptable values of *Timeout* are 0 and 1.

Only one thread may be in the transmit buffer call at any one time. Subsequent calls will generate a busy error and return.

The timeout value must be carefully chosen, allowing for the current baud rate and possible added delays due to flow control events.

Note that if binary data is being transmitted or received then XON/XOFF flow control must not be used (i.e. because valid binary data may contain XON/XOFF characters).

SEE ALSO

BASE_serial_receive_buffer, *BASE_serial_set_parameter*.

BASE_set_ptr

USAGE

*Terr BASE_set_ptr(Thandle Hbase, ui32 dwType, void *pData)*

ARGUMENTS

Hbase	Handle to a Bus Interface Board.
dwType	Memory buffer type
pData	Pointer to the memory buffer

DESCRIPTION

This function sets pointers within the Base structure to the value of *pData*.

The parameters are as follows:

BASE_SG_TABLE	This sets the structure member within the Base structure which points to the scatter-gather table, to the value of *pData*.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The baseboard handle is invalid.
ASLERR_BAD_PARAM	The parameter value is invalid.

EXAMPLES

See *BASE_PhysMem_Lock* for an example.

BUGS / NOTES

There are no known bugs.

SEE ALSO

BASE_PhysMem_Lock, *BASE_PhysMem_Unlock*.

BASE_set_timer

USAGE

Terr BASE_set_timer(Thandle Hbase, Tparam mode, ui32 time_us)

ARGUMENTS

Hbase	Handle to a Bus Interface Board.
mode	The required timer mode.
time_us	The required time interval in microseconds.

DESCRIPTION

This function controls the hardware timer fitted to the Bus Interface Board.

MODE

BASE_TIMER_MONOSTABLE	A single timed pulse is generated of width specified by *time_us*.
BASE_TIMER_ASTABLE	A continuous square wave is generated, whose period is twice that of *time_us*.
BASE_TIMER_START_AND_WAIT	The function starts the hardware timer running, but does not return until the end of the pulse. This mode is only valid in conjunction with *BASE_TIMER_MONOSTABLE*.
BASE_TIMER_START_AND_RETURN	The function starts the hardware timer running and then returns immediately.

RETURNS

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Bus Interface Board's handle is invalid.
ASLERR_BAD_PARAM	The mode parameter is invalid.
ASLERR_PARAMETER_CONFLICT	Two of the parameters conflict with each other.
ASLERR_NOT_SUPPORTED	The Bus Interface Board does not have a hardware timer, or a firmware upgrade is needed.

EXAMPLES

To use the hardware timer in conjunction with a Snapper-24 to enable an active high pulse of 1ms, followed by a low pulse of 1s, followed by a high pulse of 1ms:

```
/* Generate first pulse.
  * By using SNP24_TRIG_OUT_TIMER_HI, the output pulse is purely controlled
  * by the hardware and will therefore have minimal jitter.
 */
SNP24_set_trigger(Hsnp24, SNP24_TRIG_OUT_TIMER_HI);
BASE_set_timer(Hbase, BASE_TIMER_MONOSTABLE | BASE_TIMER_START_AND_WAIT,
       (ui32) 1000);

/* Hold exposure line low for 1 second
  * By using SNP24_TRIG_OUT_LO, although the timer pulse is generated
  * in hardware, there may be software delays before the output is cleared
  * which may give a small jitter on the delay time.
 */
SNP24_set_trigger(Hsnp24, SNP24_TRIG_OUT_LO);
BASE_set_timer(Hbase, BASE_TIMER_MONOSTABLE | BASE_TIMER_START_AND_WAIT,
       (ui32) 1000000L);
```

```
/* Finally generate second pulse */
SNP24_set_trigger(Hsnp24, SNP24_TRIG_OUT_TIMER_HI);
BASE_set_timer(Hbase, BASE_TIMER_MONOSTABLE | BASE_TIMER_START_AND_WAIT,
       (ui32) 1000);
```

BUGS / NOTES

The range of time delays supported by the hardware timer is large - typically from around 30 μs to 24 hours, with a resolution of microseconds for short delays, although the use of a 32 bit integer for *time_us* limits the maximum monostable delay to around 1 hour 11 minutes. If the time requested is smaller than that which is supported then the minimum supported delay is generated.

The ISA-BIB and ISA-JPG do not support hardware timers, and early versions of PCI-BIB and SBUS-BIB need a firmware upgrade for all timer functions to work.

There are no known bugs.

SEE ALSO

-