Host Interface Library

Reference Manual

DSP21k-SF Toolkit 7.4 Interface Libraries & Utilities for
BittWare DSP Boards with SharcFIN DSP-PCI Bridge

BittWare

BittWare, Inc.
31B S. Main St.
Concord, NH 03301 USA
603.226.0404

If you have comments or suggestions about this manual or find any errors in it, please contact us via e-mail at techpubs@bittware.com.

For technical support, contact us using any of the following methods:
Phone: 603.226.0404
FAX: 603.226.6667
E-mail: support@bittware.com

BittWare also maintains the following Internet sites:

http://www.bittware.com	Contains product information, technical notes, support files available for download, and answers to frequently asked questions (FAQ).
ftp://ftp.bittware.com	Contains technical notes and support files. Log in as "anonymous," and use your e-mail address for the password.

Host Interface Library Reference Manual

Release 7.4
32-bit version for Windows® and Linux

The information in this manual has been carefully checked and is believed to be accurate and reliable. However, BittWare assumes no responsibility for any inaccuracies, errors, or omissions that may be contained in this manual. In no event will BittWare be liable for direct, indirect, special, incidental, or consequential damages resulting from any defect or omission in this manual. BittWare reserves the right to revise this document and to make changes from time to time in the content hereof without obligation of BittWare to notify any person or persons of such revision or changes.

Printed in the USA
July 8, 2005 Edition (*UG-DTSF74-00-2*)

Document History

Document Number	Document Revision	Date	Changes	Section
UG-DTSF74-00-2	0	07/08/2005	Initial release	

This page intentionally left blank.

Contents

Chapter 1
Introduction

1.1 About this Reference Manual 12
1.1.1 Topics Covered in this Reference Manual 12
1.1.2 Conventions in this Reference Manual. 12
1.1.3 Chapter Overviews 13

1.2 Other Related Documents and Products 14
1.2.1 Documents 14
1.2.2 Products 14

Chapter 2
Using the Host Interface Library

2.1 Upgrading From an Earlier Version of the HIL 18

2.2 Compiling Programs with the HIL 19
2.2.1 Compiling Programs with the HIL Using Windows 19
2.2.2 Compiling Programs with the HIL Using Linux 19
2.2.3 Using HIL Functions 20
2.2.4 HIL Message Handler 23
2.2.5 HIL Error Messages 23

2.3 Accessing DSP Memory 25
2.3.1 Processor Memory Addresses 25
2.3.2 Processor Resources Used 25
2.3.3 Processor Memory-Mapped Registers 26

2.4 Accessing Host Physical Memory with the HIL 27

2.5 Handling PCI Interrupts with the HIL 29
2.5.1 Using an Interrupt Handler for Handling PCI Interrupts 29
2.5.2 Using the dsp21k_int_wait Function for Handling PCI Interrupts . . 31

2.6 Host Interface Library Function Groups 32
2.6.1 Board Control Functions 32
2.6.2 Data Transfer Functions 32
2.6.3 Host Physical Memory Buffer Functions 33
2.6.4 DSP Information Functions 33
2.6.5 DSP Error and Message Functions 34
2.6.6 Processor Control Functions 34
2.6.7 Program Control Functions 35
2.6.8 DSP Broadcast Functions 35

Chapter 3
Host Interface Library Function Descriptions

dsp21k_alloc_phys_memory 38
dsp21k_bank_depth 41
dsp21k_bank_size 42
dsp21k_bank_start 43
dsp21k_bank_width 44
dsp21k_bc_cfg_proc 45
dsp21k_bc_dl_32s 46
dsp21k_bc_reset_proc 47
dsp21k_bc_start 48
dsp21k_bc_wiop 49
dsp21k_block_depth 50
dsp21k_block_start 51
dsp21k_board_name 52
dsp21k_board_type 53
dsp21k_build 54
dsp21k_cfg_proc 55
dsp21k_close 57
dsp21k_close_all 58
dsp21k_dbl_to_xflt 59
dsp21k_device_num 60
dsp21k_dir 61
dsp21k_dl_8s 62
dsp21k_dl_16s 63
dsp21k_dl_32s 64
dsp21k_dl_48 66
dsp21k_dl_48s 68
dsp21k_dl_64s 69
dsp21k_dl_dbl 70
dsp21k_dl_flt 71
dsp21k_dl_int 72

dsp21k_dl_sctn32 73
dsp21k_dl_sctn48 74
dsp21k_dl_xflt 75
dsp21k_dma_start 76
dsp21k_dsp_name 79
dsp21k_dsp_rev 80
dsp21k_dsp_type 81
dsp21k_err_msgs 82
dsp21k_fast_extmem_xfers 83
dsp21k_free_labels 85
dsp21k_free_phys_memory 86
dsp21k_get_addr 88
dsp21k_get_board_name 90
dsp21k_get_device_info 91
dsp21k_get_dsp_name 92
dsp21k_get_last_error 94
dsp21k_get_next_symbol 95
dsp21k_get_pcirq 97
dsp21k_get_phys_memory 98
dsp21k_get_proc 99
dsp21k_get_symbol 100
dsp21k_int_disable 101
dsp21k_int_dsp 103
dsp21k_int_enable 104
dsp21k_int_wait 107
dsp21k_is_bc_capable 109
dsp21k_is_dma_capable 110
dsp21k_is_dma_complete 111
dsp21k_labels_defined 112
dsp21k_load_exe 113
dsp21k_load_symbols 115
dsp21k_loaded_file 116
dsp21k_mem_width 117
dsp21k_mpid 118
dsp21k_msg 119
dsp21k_msg_func 120
dsp21k_num_dsps 121
dsp21k_open 123
dsp21k_open_all 125
dsp21k_open_by_id 126
dsp21k_phys_memory_count 128
dsp21k_prn_copyright 129
dsp21k_prn_version 130
dsp21k_proc_num 131
dsp21k_proc_running 132

dsp21k_rd_bd_setting 133
dsp21k_rd_bdreg 134
dsp21k_rd_phys_memory 135
dsp21k_rd_phys_memory8 136
dsp21k_reset_bd 137
dsp21k_reset_proc 139
dsp21k_riop 140
dsp21k_serial_num 141
dsp21k_sleep 142
dsp21k_start 143
dsp21k_symbol_count 144
dsp21k_symbol_size 145
dsp21k_symbol_width 147
dsp21k_target 149
dsp21k_ul_8s 150
dsp21k_ul_16s 151
dsp21k_ul_32s 152
dsp21k_ul_48 153
dsp21k_ul_48s 155
dsp21k_ul_64s 156
dsp21k_ul_dbl 157
dsp21k_ul_flt 158
dsp21k_ul_int 159
dsp21k_ul_sctn32 160
dsp21k_ul_sctn48 161
dsp21k_ul_xflt 162
dsp21k_usleep 163
dsp21k_version 164
dsp21k_wiop 165
dsp21k_wr_bd_setting 166
dsp21k_wr_bdreg 167
dsp21k_wr_phys_memory 168
dsp21k_wr_phys_memory8 169
dsp21k_xflt_to_dbl 170

Chapter 4
Redistributing/Installing a HIL-based Application

4.1 Redistribution/Installation in Linux 172
4.1.1 Linux System Requirements 172
4.1.2 Installing the HIL in Linux 172

4.2 Redistribution/Installation in Windows 173
4.2.1 Windows System Requirements 173
4.2.2 Installing the HIL in Windows 173

Index 175

This page intentionally left blank.

Chapter 1
Introduction

This document describes Release 7.4 of the Host Interface Library (HIL), which is the primary component of BittWare's DSP21k-SF Toolkit. The DSP21k-SF Toolkit is a collection of libraries, applications, and diagnostics that provides the glue between your application and the hardware, allowing you to develop DSP applications more quickly and easily. Nearly all of the utilities in the DSP21k-SF Toolkit are based on the HIL.

The Host Interface Library is a library of C-callable functions for Windows programs that allows your programs to control the DSP board, read from and write to the DSP board's memory, and control other board functions. It also provides complete control for loading and starting programs on the DSP, configuring and generating interrupts, and accessing the DSP's data structures symbolically.

Release 7.4 of the Host Interface Library supports all of BittWare's products that feature the SharcFIN DSP-PCI bridge and is compatible with most C compilers and other development tools. The HIL provides a common set of functions for all of the supported boards and automatically performs the required interface operations.

1.1 About this Reference Manual

This document is a reference to all of the functions of the Host Interface Library (HIL). It is part of a set of two manuals for Release 7.4 of the DSP21k-SF Toolkit: the *DSP21k-SF Toolkit User's Guide* and the *Host Interface Library Reference Manual*. The HIL is included in Release 7.4 of the 32-bit version of the DSP21k-SF Toolkit for Windows® and in the Toolkit for Linux.

Note *Release 7.4 of the DSP21k-SF Toolkit is compatible only with the BittWare boards that support the SharcFIN DSP-PCI bridge.*

We assume that you are already familiar with the SHARC® or TigerSHARC® family of digital signal processors, that you have installed the DSP21k-SF Toolkit, and that you have installed and configured your BittWare DSP board. Please refer to the release note and the file `readme.txt` for updated information that was not available when this document was printed.

1.1.1 Topics Covered in this Reference Manual

This document covers:

- Compiling programs with the Host Interface Library
- An example program that shows how to use the HIL's functions
- The HIL's error and message handling functions
- The HIL function groups
- A description of each HIL function

1.1.2 Conventions in this Reference Manual

Below is a list of conventions we have used throughout this document.

- Functions and commands that the user enters appear in **`bold`** font.
- Executable programs that are compiled for the Analog Devices ADSP-21xxx and ADSP-TSxxx processors have a .dxe extension instead of a .exe extension to avoid confusing them with executable files for the PC.

1.1.3 Chapter Overviews

Chapter 2: Using the Host Interface Library

Chapter 2 gives an overview of the HIL. It discusses compiling programs with the HIL, runs through an example program of several HIL functions, and discusses the HIL function groups.

Chapter 3: Host Interface Library Function Descriptions

Chapter 3 describes each HIL function. All functions are listed in alphabetical order.

Chapter 4: Redistributing/Installing a HIL-based Application

Chapter 4 describes the steps involved in installing or redistributing a HIL-based application in both Linux and Windows environments.

1.2 Other Related Documents and Products

1.2.1 Documents

This document is a complete reference manual to Release 7.4 of the Host Interface Library, which is included in the DSP21k-SF Toolkit. Please refer to the following documents for more information:

- ADSP-21xxx SHARC or ADSP-TSxxx Documentation – Analog Devices
- *DSP21k-SF Toolkit User's Guide (Release 7.4)*– BittWare, Inc.
- User's guide for your BittWare DSP hardware – BittWare, Inc.

1.2.2 Products

Analog Devices' VisualDSP++®

Analog Devices' VisualDSP++ includes an integrated development environment (IDE) and a debugger that delivers efficient project management so programmers can move easily between editing, building and debugging. Key features include a C++ compiler, an enhanced user interface, advanced plotting tools that enable programmers to visually measure software performance, and statistical profiling to easily identify programming bottlenecks. VisualDSP++ offers programmers a powerful programming tool that significantly decreases the time required to port software code to a DSP, thereby reducing time-to-market.

BittWare's VisualDSP Target

BittWare's VisualDSP Target allows you to debug your DSP application right on your BittWare board without a hardware emulator. A plug-in to Analog Devices' VisualDSP++ IDE, the Target allows the VisualDSP debugger to communicate directly with BittWare's DSP boards.

BittWare's Porting Kit

BittWare's DSP21k Porting Kit gives you the freedom to use your BittWare board on the operating system of your choice. It allows you to easily adapt the DSP21k-SF Toolkit to fit your system, so you can develop DSP applications for your BittWare board.

Remote Client Toolkit

BittWare's Remote Client Toolkit allows for remote access of BittWare hardware anywhere in the world providing the host computer is accessible by TCP/IP. The Remote Client Toolkit provides client versions of all of BittWare's standard tools. Remote Client Toolkit is the perfect tool for the DSP developer who does not always have access to the hardware and its host computer directly. Those familiar with BittWare's standard tools will find that using the Remote Client Toolkit is almost as if the hardware was in the client machine.

BittWare's VisualDSP Remote Target

BittWare's VisualDSP Remote Target allows you to remotely debug your DSP application right on your BittWare board without a hardware emulator. If the host computer provides TCP/IP access, you can use the Remote Target under Analog Devices' VisualDSP ++ IDE to debug code remotely. This can be especially useful for when the host computer is running an operating system on which the VisualDSP++ IDE cannot run.

Chapter 2.
Using the Host Interface Library

This chapter gives an overview of Release 7.4 of the DSP21k Host Interface Library (HIL). The Host Interface Library (HIL) provides a high-level interface to your SharcFIN-compatible BittWare DSP board for programs that run on the PC. This comprehensive set of functions provides routines for downloading code, transferring data, controlling the processor, and handling messages. This chapter covers:

- Compiling programs with the Host Interface Library
- The example program `runprime.c`, which shows how to use the HIL's functions
- The HIL's error and message handling functions
- The HIL function groups

Note *Use only the functions documented in Chapter 3 to control and access the DSP board. Using these library functions will help to ensure future compatibility with hardware and software upgrades.*

2.1 Upgrading From an Earlier Version of the HIL

BittWare makes an effort to keep new releases of the HIL backwards compatible with older versions. We recommend using the latest functions instead of deprecated ones, however, older functions will still work as originally advertised. The core part of the API is much the same as it has been since it was first released over ten years ago.

Although the API remains the same, the internal workings of the HIL have gone through and continue to go through some dramatic changes. Because of this, applications that use the HIL usually require a rebuild against a newer release. We strongly recommend that all applications using the HIL be rebuilt with each new release, whether it is a major or minor release. This applies to both Porting Kit users on any operating system and Toolkit users on Linux or Windows operating systems. It is only not necessary to rebuild against a patched library.

2.2 Compiling Programs with the HIL

This section provides instructions for compiling a program using the Host Interface Library in either a Windows- or Linux-based environment.

2.2.1 Compiling Programs with the HIL Using Windows

Compiling a program in Windows requires two steps:

1. Include the header file `dsp21k.h` in your source file. The DSP21k-SF Toolkit stores the header file in the `\dsp21ksf\inc` directory during installation. The header file provides prototypes for all of the Host Interface Library functions and describes important constants and structures.
2. Link the Host Interface Library (specific to your compiler) with your program.

The `\dsp21ksf\lib` directory contains all libraries, and the `\dsp21ksf\bin` directory contains the Windows DLL. When using the Microsoft tools, link projects with `dsp21ksf\lib\hil.lib`. Since the library is a 32-bit Windows DLL, 32-bit programs (such as Delphi, Labview, and Visual Basic) that can access 32-bit DLLs can access the library. The Host Interface Library is compatible with all of BittWare's products that support the SharcFIN DSP-PCI bridge.

2.2.2 Compiling Programs with the HIL Using Linux

Compiling a program with the HIL using Linux requires two steps:

1. Include the header file `dsp21k.h` in your source file. DSP21k-SF Toolkit stores the header file in the `/usr/local/bittware/inc` directory during installation. The header file provides prototypes for all of the Host Interface Library functions and describes important constants and structures.
2. Link your program with the -lhil flag to link against `/usr/lib/libhil.so`, which is the Host Interface Library. Also link with the -lpthread flag to link with the pthread library. The Host Interface Library uses the pthread library internally

The `/usr/local/bittware/lib` directory contains all libraries, and the `/usr/local/bittware/bin` directory contains the Linux

executables. The Host Interface Library is compatible with all of BittWare's products that support the SharcFIN DSP-PCI bridge.

2.2.3 Using HIL Functions

The example program in the `\dsp21ksf\examples\host\runprime` directory explains how to use the HIL's functions. The example program, `runprime.c` (see the example "runprime.c" on page 21), downloads a program to the DSP board, starts the program, and then uploads the results.

Notice the example program's variable **`processor`**, which is of type **`PDSP21K`** and is a pointer to a type **`DSP21K`**. **`DSP21K`** is a structure that holds all of the important information about a DSP that the library functions must reference. Almost all of the functions in the HIL require this pointer as an argument.

The *dsp21k_open_by_id* function, which returns a pointer to a **`DSP21K`** processor structure, initializes the **`DSP21K`** structure. The *dsp21k_close* function destroys the processor structure and is called when the processor is no longer needed (see the end of the example program on page 12).

The *dsp21k_reset_bd* function performs a hardware reset of the board, resetting all processors on the board. Then, *dsp21k_cfg_proc* configures the processor. Be sure to perform the *dsp21k_cfg_proc* function on all processors.

The *dsp21k_load_exe* function resets the DSP and automatically downloads the specified executable file to the appropriate locations in the DSP's memory. The return value is checked and the program prints an error message to the screen if *dsp21k_load_exe* fails. The *dsp21k_start* function releases reset to start executing the program.

The while loop demonstrates the *dsp21k_ul_int* and *dsp21k_get_addr* functions. The *dsp21k_ul_int* function reads an integer value from the DSP's memory at the address specified in the second argument. In this example program, the *dsp21k_get_addr* function supplies the address. It can provide addresses for the global variables and program labels that it captured while downloading the executable with the *dsp21k_load_exe* function.

Note *Global variable names have leading underscores added to match the C naming convention.*

Once the program detects the appropriate value in the **_done** variable, it uploads the results with the *dsp21k_ul_32s* function. The *dsp21k_ul_32s* function is similar to *dsp21k_ul_int*, except *dsp21k_ul_32s* uploads multiple 32-bit integers into a buffer. The HIL contains a comprehensive set of functions to upload and download various types of data.

Example 2–1 *runprime.c*

```
#include <stdio.h>
#include <stdlib.h>

#include "dsp21k.h"

// Device number to open
#define DEVICE_NUMBER 0

//*******************************************************
//
//
//
//*******************************************************
int main(void)
{
    PDSP21K processor;
    int primesbuf[20];
    int indx, error;
    char file_buf[20];

    // Open the 1st processor on device.
    if ((processor = dsp21k_open(DEVICE_NUMBER)) == NULL)
    {
        printf("problem opening 1st processor on device %d\n", DEVICE_NUMBER);
        return 1;
    }

    // Reset the board.
    dsp21k_reset_bd( processor );

    // Configure the processor.
    dsp21k_cfg_proc( processor );

    // Determine which DSP program to load
    switch(dsp21k_dsp_type(processor))
    {
```

```
    case ADSP_21065:
        sprintf(file_buf, "prime65.dxe");
        break;
    case ADSP_21160:
        sprintf(file_buf, "prm21160.dxe");
        break;
    case ADSP_TS101:
        sprintf(file_buf, "prmTS101.dxe");
        break;
    default:
        printf("processor type %d not supported by this program\n",
dsp21k_dsp_type(processor));
        dsp21k_close(processor);
        return 1;
    }

    // Download the dsp program.
    error = dsp21k_load_exe(processor, file_buf);

    //check error
    if(error < 0)
    {
        // Display last error, always close processor to
        // free memory before exiting program
        printf("%s\n", dsp21k_err_msgs(error));

        // Close the processor and exit
        dsp21k_close(processor);
        return 1;
    }

    // Start processor running.
    dsp21k_start(processor);

    // Wait for DSP program to set done=1.
    while (!dsp21k_ul_int(processor, dsp21k_get_addr(processor, "_done")))
        ;

    // Upload the results.
    dsp21k_ul_32s(processor, dsp21k_get_addr(processor, "_primes"), 20, primesbuf);

    // Print out the results.
    printf("The DSP calculated the first 20 primes to be:\n");
    for (indx = 0; indx < 20; indx++)
    {
        printf("%d\n", primesbuf[indx]);
    }

    // Close the processor.
```

```
    dsp21k_close(processor);
    return 0;
}
```

2.2.4 HIL Message Handler

The HIL provides a mechanism to route messages through a common message handler. The HIL function, *dsp21k_msg_func* can be used to override the default message handler with a new handler. When the function *dsp21k_msg* is called, the message will be routed to the new message handler.

2.2.5 HIL Error Messages

Most of the HIL functions return HIL error values. A return value of 0 (**`DSP21K_SUCCESS`**) indicates that the function completed successfully. A negative return value indicates some type of failure. When a function returns with a negative value, you can call *dsp21k_err_msgs* to get the error string associated with that error.

When a function does not return a HIL error value (such as *dsp21k_open*), you can check for an error using the function *dsp21k_get_last_error*.

Table 2–3 shows the values for the HIL error messages, and the example below shows how to test for failure.

Example 2–2 *Testing for Errors or Failure*

```
...
retval = dsp21k_ul_32s(processor, dsp_addr, count, buf_32s);
if(retval < 0)
{
    //error!
    printf("Error in dsp21k_ul_32s: %s\n",
        dsp21k_err_msgs(retval));
}
else
...
```

Table 2–3 HIL Error Message Values

Error Message*	Value	Error Message	Value
DSP21K_SUCCESS	0	DSP21K_FILE_NOT_FOUND	–20
DSP21K_ERROR	–1	DSP21K_ITEM_NOT_FOUND	–21
DSP21K_NOT_SUPPORTED	–2	DSP21K_SYMBOL_NOT_FOUND	–22
DSP21K_INVALID_DATA	–3	DSP21K_FILE_CREATE_ERROR	–24
DSP21K_OUT_OF_MEMORY	–4	DSP21K_FILE_READ_ERROR	–25
DSP21K_CANNOT_MAP_BAR	–5	DSP21K_ELF_LIB_ERROR	–26
DSP21K_NULL_PROCESSOR	–6	DSP21K_INTERRUPT_ERROR	–28
DSP21K_INVALID_ADDRESS	–7	DSP21K_CANNOT_CLOSE	–30
DSP21K_INVALID_COUNT	–8	DSP21K_OPEN_ERROR	–31
DSP21K_TIMEOUT	–10	DSP21K_DRIVER_NOT_FOUND	–40
DSP21K_CANCELED	–11	DSP21K_DRIVER VERSION	–41
DSP21K_EEPROM_CORRUPT	–15	DSP21K_DRIVER_KERNEL	–42
DSP21K_NEEDS_CFG_FILE	–16	DSP21K_DRIVER_ERROR	–43
DSP21K_DEVICE_NOT_FOUND	–19	DSP21K_NETWORK_ERROR	–50

* Error definitions are in `const21k.h`

2.3 Accessing DSP Memory

2.3.1 Processor Memory Addresses

The DSP memory addresses that the HIL functions use are the actual DSP addresses that the processor uses for both internal and external DSP memory access. The HIL has different functions for downloading to memory that is addressed with different data widths. For example, to download to a 16-bit SHARC address, the HIL function *dsp21k_dl_16s* is used. To upload from a 48-bit SHARC address, the HIL function *dsp21k_ul_48s* is used.

The following functions are exceptions to this rule because function addresses for these parameters are offsets into the register spaces rather than DSP addresses.

- dsp21k_rd_bdreg
- dsp21k_wr_bdreg

If a HIL function is given an invalid address or an address that does not match the data width of the function, a HIL error value will be returned.

2.3.2 Processor Resources Used

The Host Interface library uses certain resources of the DSPs and the boards carrying them in order to perform its task of transferring data. Therefore, any application on either the PC host or DSP must avoid using these resources. The following list briefly identifies all known resources that BittWare's SharcFIN boards use.

Table 2–4 DSP Resources Used

Processor Type	Resource Used
ADSP-21065L	SHARC DMA channel 1 for all host access to memory (internal/external)
ADSP-21060 and ADSP-21062	DMA channel 8 for 48-bit external memory host accesses
ADSP-21061	DMA channel 7 for 48-bit external memory host accesses
ADSP-21160	No resources used
ADSP-21161	DMA channel 13 for all host access to memory (internal/external)
ADSP-TS101S and ADSP-TS201S	TCBO and DMAR1 for SDRAM access TCB0/TCB1 and DMAR0 for DMA functions

2.3.3 Processor Memory-Mapped Registers

A processor's memory-mapped registers can be accessed two ways.

- The functions, *dsp21k_riop* and *dsp21k_wiop* allow access to the processor's IOP registers. The TigerSHARC (TS) family of processors does not technically have IOP registers, but they do have a large amount of memory-mapped registers, all of which can be accessed using *dsp21k_riop* and *dsp21k_wiop*.
- The functions *dsp21k_ul_32s* and *dsp21k_dl_32s* also allow access to memory-mapped processor registers; whether they are 21xxx IOP registers or TS memory mapped registers. The addresses used to access the memory-mapped registers are the same addresses that the processor itself uses.

Some processors require that quad-word registers or long-word registers be accessed in total from the host processor. In the case a part of one of these registers is requested using one of the two methods above, the host interface library will access the entire register but only return/modify the part corresponding to the specified address.

2.4 Accessing Host Physical Memory with the HIL

The Host Interface Library (HIL) provides access to the host computer's physical memory. The HIL can lock down a buffer of physical memory and map it into your application's virtual memory so that you can access it directly. The HIL can also access a buffer of physical memory that has already been locked down in another process or application. The latter includes the ability to access PCI addresses directly.

In both cases, access to physical memory is provided through the HIL function *dsp21k_alloc_phys_memory*. This function takes a processor and an address to a DSP21K_PHYS_MEMORY structure. The DSP21K_PHYS_MEMORY structure contains the following fields:

- **`phys_addr`** Host physical memory address, accessible by the PCI device.
- **`size`** Requested size in bytes of the memory buffer.
- **`mem_ptr`** Virtual memory pointer, accessible by the Host PC.

1. To allocate a new physical memory buffer, set the parameters as follows:
   - **`phys_addr`** Must be 0
   - **`size`** Size in bytes of buffer to allocate
   - **`mem_ptr`** N/A

2. To map physical memory that has been allocated in a different process or exists as a PCI address, set the parameters as follows:
   - **`phys_addr`** Physical memory or PCI address
   - **`size`** Size in bytes of memory to map
   - **`mem_ptr`** N/A

In the first case, if the function succeeds, **`phys_addr`** will be filled in with the physical memory address of the physical memory buffer that has been locked down. In the second case, it is very important that **`phys_addr`** is a valid physical memory address or PCI address. Setting this parameter incorrectly can cause system or hardware failure (see warning below).

Warning *Be careful when giving a host physical address to map. If an incorrect address is used, operating system memory or another PCI device's memory could be accessed. Accessing unknown host physical memory can possibly corrupt the operating system or affect the hardware in some way, resulting in a crash or a hardware failure. Permanent corruption of the operating system or hardware is also possible.*

In both cases, **`mem_ptr`** will be filled in with a virtual memory pointer that can only be used in the current process. Dereferencing **`mem_ptr`** as a pointer to a type (such as LPULONG or LPCHAR) will result in reads or writes to that physical memory. Physical memory buffer access functions are also provided to read from and write to allocated host physical memory. Using the functions, *dsp21k_rd_phys_memory*, *dsp21k_rd_phys_memory8*, *dsp21k_wr_phys_memory*, and *dsp21k_wr_phys_memory8* is the recommended method of accessing physical memory for portability.

Each time the function *dsp21k_alloc_phys_memory* is called, an element is added to an internal list keeping track of allocated and mapped memory within the HIL. You can access this internal list by using the functions *dsp21k_phys_memory_count* and *dsp21k_get_phys_memory*.

The HIL provides the function *dsp21k_free_phys_memory* to free or unmap host physical memory. All physical memory should be freed before the last processor is closed.

2.5 Handling PCI Interrupts with the HIL

There are two methods for handling PCI Interrupts using the Host Interface Library. The first method is to use the functions, *dsp21k_int_enable* and *dsp21k_int_disable*. The second method uses an additional function, *dsp21k_int_wait* to be used as an alternative to the user interrupt handler passed to *dsp21k_int_enable*. The first method is described below.

2.5.1 Using an Interrupt Handler for Handling PCI Interrupts

```
dsp21k_int_enable(PDSP21K processor,
   dsp21k_int_handler_fptr handler, void * param)
```

processor	Pointer to processor structure
handler	Address of function to call on interrupt
param	Pointer to user data to pass to the interrupt handler

The type **dsp21k_int_handler_fptr** has the following form:

```
void my_int_handler(ULONG user_int_status, UCHAR
   mailbox_status, void * param)
```

user_int_status	32-bit interrupt status register
mailbox_status	8-bit mailbox interrupt status register
param	Pointer to user data passed to dsp21k_int_enable

Table 2–5 shows which register contents are passed into each status parameter, depending on the type of BittWare device.

Table 2–5 Interrupt Status Parameters for BittWare PCI Devices

Device Type	user_int_status (32-bit offsets)	mailbox_status (byte offsets)
SFIN-160, SFIN-101, SFIN-201 and DataFIN	Flag Status and Interrupt Status registers (0x5E)	Incoming Mailbox Full status register (0x85)
SFIN-161	IRQ Status register (0xC)	unused
PLX	Local to PCI Doorbell register (0x19)	unused

The *dsp21k_int_enable* function inserts an interrupt service routine (ISR) into the operating system's interrupt table for your device's interrupt number. The function then starts a thread that waits for an event triggered by the ISR and opens up the interrupt mask(s) on the device. The interrupt handler parameter to *dsp21k_int_enable* is saved so that it can be called when the thread is woken up.

The thread has the same priority and runs in the same context as the thread that creates it. In advance of calling *dsp21k_int_enable*, the thread's priority can be set by calling the function *dsp21k_wr_bd_setting* with a parameter of DEV_INTERRUPT_PRIORITY and a value of the priority you want the thread to run at.

When an interrupt occurs, the ISR reads the interrupt status register(s) on the device, clears the interrupt, and saves the status (ANDed with the mask). When the waiting thread is woken up, it reads the saved status and passes it to the user interrupt handler. In most operating systems, there is a modest latency between the execution of the ISR and the call to the user interrupt handler.

The *dsp21k_int_disable* function handles any pending interrupts, masks the interrupt, and removes the ISR from the operating system's interrupt tabletable, and stops the thread that is waiting on interrupt events. Closing the last open processor on a device also disables the interrupts for that device.

2.5.2 Using the dsp21k_int_wait Function for Handling PCI Interrupts

```
dsp21k_int_enable(PDSP21K processor,
   dsp21k_int_handler_fptr handler, void * param)
```

processor	Pointer to processor structure
handler	null
param	null

```
dsp21k_int_wait(PDSP21K processor, ULONG
   user_int_status, UCHAR mailbox_status)
```

processor	pointer to processor structure
user_int_status	32-bit interrupt status register
mailbox_status	8-bit mailbox interrupt status register

Table 2–5 shows which register contents are passed into each status parameter, depending on the type of BittWare device.

The *dsp21k_int_enable* function inserts an interrupt service routine (ISR) into the operating system's interrupt table for your device's interrupt number and opens up the interrupt mask(s) on the device. The NULL handler parameter to *dsp21k_int_enable* informs the HIL that the *dsp21k_int_wait* function will be used to retrieve the interrupts.

When an interrupt occurs, the ISR reads the interrupt status register(s) on the device, clears the interrupt, and saves the status (ANDed with the mask). If the interrupt has occurred before calling *dsp21k_int_wait*, the values are stored until they are retrieved with a call to *dsp21k_int_wait*. Otherwise, *dsp21k_int_wait* will wait for interrupts until a call to *dsp21k_int_disable*. In either case, *dsp21k_int_wait* gets the values saved by the first interrupt whose values are pending retrieval. In most operating systems, there is a modest latency between the execution of the ISR and when the *dsp21k_int_wait* function will return with the status values.

The *dsp21k_int_disable* function handles any pending interrupts, masks the interrupt, and removes the ISR from the operating system's interrupt table. If *dsp21k_int_disable* is called from another thread, a thread that is waiting in *dsp21k_int_wait* will return. Closing the last open processor on a device also disables the interrupts for that device.

2.6 Host Interface Library Function Groups

The Host Interface Library functions fall into one of the following groups.

2.6.1 Board Control Functions

The functions below manage the carrier board.

dsp21k_dma_start[1]
dsp21k_int_disable
dsp21k_int_enable
dsp21k_int_dsp
dsp21k_int_wait
dsp21k_is_dma_capable[1]
dsp21k_is_dma_complete[1]
dsp21k_rd_bd_setting
dsp21k_rd_bdreg
dsp21k_reset_bd
dsp21k_wr_bd_setting
dsp21k_wr_bdreg

2.6.2 Data Transfer Functions

The functions below perform data transfers or conversions.

dsp21k_dbl_to_xflt
dsp21k_dl_8s
dsp21k_dl_16s
dsp21k_dl_32s
dsp21k_dl_48
dsp21k_dl_64s
dsp21k_dl_dbl
dsp21k_dl_flt
dsp21k_dl_sctn32
dsp21k_dl_sctn48
dsp21k_dl_xflt
dsp21k_dma_start[1]
dsp21k_is_dma_capable[1]
dsp21k_is_dma_complete[1]
dsp21k_fast_extmem_xfers

1. This function can serve as either a board control or data transfer function.

dsp21k_ul_8s
dsp21k_ul_16s
dsp21k_ul_32s
dsp21k_ul_48(s)
dsp21k_ul_64s
dsp21k_ul_dbl
dsp21k_ul_flt
dsp21k_ul_int
dsp21k_ul_sctn32
dsp21k_ul_sctn48
dsp21k_ul_xflt
dsp21k_xflt_to_dbl

2.6.3 Host Physical Memory Buffer Functions

The functions below control the host physical memory buffer:

dsp21k_alloc_phys_memory
dsp21k_free_phys_memory
dsp21k_get_phys_memory
dsp21k_phys_memory_count
dsp21k_rd_phys_memory
dsp21k_rd_phys_memory8
dsp21k_wr_phys_memory
dsp21k_wr_phys_memory8

2.6.4 DSP Information Functions

The functions below return information about the DSP hardware and tools.

dsp21k_bank_depth
dsp21k_bank_size
dsp21k_bank_width
dsp21k_board_name
dsp21k_board_type
dsp21k_build
dsp21k_device_num
dsp21k_dir
dsp21k_dsp_name
dsp21k_dsp_rev
dsp21k_dsp_type
dsp21k_get_board_name

dsp21k_get_device_info
dsp21k_get_dsp_name
dsp21k_get_pcirq
dsp21k_loaded_file
dsp21k_mem_width
dsp21k_mpid
dsp21k_num_dsps
dsp21k_prn_copyright
dsp21k_prn_version
dsp21k_proc_num
dsp21k_proc_running
dsp21k_serial_num
dsp21k_target
dsp21k_version

2.6.5 DSP Error and Message Functions

The functions below manage user and error messages.

dsp21k_err_msgs
dsp21k_get_last_error
dsp21k_msg
dsp21k_msg_func

2.6.6 Processor Control Functions

The functions below manage individual processors.

dsp21k_cfg_proc
dsp21k_close
dsp21k_open
dsp21k_open_by_id
dsp21k_open_by_file
dsp21k_reset_proc
dsp21k_riop[1]
dsp21k_wiop[1]

The functions below manage all available processors.

dsp21k_close_all
dsp21k_get_proc
dsp21k_open_all

1. The input parameters of this function have changed from earlier versions.

2.6.7 Program Control Functions

The functions below manage programs on the DSP.

dsp21k_load_exe
dsp21k_free_labels
dsp21k_get_addr
dsp21k_get_next_symbol
dsp21k_get_symbol
dsp21k_labels_defined
dsp21k_load_symbols[1]
dsp21k_start
dsp21k_symbol_count
dsp21k_symbol_size
dsp21k_symbol_width

2.6.8 DSP Broadcast Functions

The functions below broadcast to all the DSPs in the cluster.

dsp21k_bc_reset_proc
dsp21k_bc_cfg_proc
dsp21k_bc_start
dsp21k_bc_wiop
dsp21k_bc_dl_32s
dsp21k_is_bc_capable

1. The return value of this function has changed from previous versions.

Chapter 3

Host Interface Library Function Descriptions

This chapter describes each function in the DSP21k-SF Host Interface Library (HIL). The HIL provides a high-level interface to your BittWare DSP board for programs that run on the PC. This comprehensive set of functions provides routines for downloading code, transferring data, controlling the processor, and handling messages.

The following is a complete alphabetical list of the HIL functions. Use only the functions included in this reference manual.

Note *Do not directly access the elements of the PDSP21K processor structure because accessing those elements may create alignment problems, and the structure may change in future releases. If the HIL does not have a function to access the information you need, please contact BittWare so that we can add it in a future release.*

dsp21k_alloc_phys_memory

Prototype

```
int dsp21k_alloc_phys_memory( PDSP21K processor,
    DSP21K_PHYS_MEMORY *phys_mem)
```

Description

This function provides two different behaviors:

1. If **phys_addr** and **size** are filled in, then the function attempts to map the host PC's physical memory at that address (such as a physical memory buffer from a different process).

2. If **phys_addr** is 0 and **size** is filled in, the function attempts to allocate a new contiguous physical memory buffer on the host PC.

If a buffer was successfully mapped or allocated, the **phys_addr** parameter will contain the physical address of the buffer, the **size** parameter will contain the actual size of the buffer, and the **mem_ptr** parameter will be a pointer to virtual memory in the application space with which to access the memory. If the host PC does not have enough contiguous physical memory or cannot map the memory requested, the function will fail and return a negative HIL error value.

This function will perform differently on different operating systems. For example, on Windows 9x, the contiguous memory is allocated on a first come, first serve basis until it is used up; On most Linux systems, contiguous memory exists only in 128 KB chunks. The only limit to the number of physical buffers that can be allocated in any system is the amount of memory. The allocated buffer can be freed with a call to the function *dsp21k_free_phys_memory*.

Arguments

processor Pointer to processor structure

phys_mem Address of a DSP21K_PHYS_MEMORY structure whose members are:

- **phys_addr**: This parameter is required: 0 or host **physical memory address**
- **size:** Requested size in bytes of the memory buffer
- **mem_ptr:** Virtual memory pointer, accessible by the host PC

Returns The function returns 0 if it succeeds (DSP21K_SUCCESS). If it does not succeed, it returns a negative HIL error value.

Example

```
...
#define BUFFER_SIZE 0x1000
DSP21K_PHYS_MEMORY phys_mem;
ULONG wr_buffer[BUFFER_SIZE / 4];
ULONG rd_buffer[BUFFER_SIZE / 4];
int i;
  ...

//MUST set phys_addr to 0 if allocating new buffer
phys_mem.phys_addr = 0;
//allocate 4K bytes (1K 32-bit words)
phys_mem.size = BUFFER_SIZE;
dsp21k_alloc_phys_memory(processor, &phys_mem);

//write a count to the buffer
for(i = 0; i < phys_mem.size / 4; i++)
{
       wr_buffer[i] = i;
}

//write buffer to physical memory
dsp21k_wr_phys_memory(&phys_mem, 0,
phys_mem.size / 4, wr_buffer);
//read back the buffer
dsp21k_rd_phys_memory(&phys_mem, 0,
phys_mem.size / 4, rd_buffer);
```

```
if(memcmp((void *)wr_buffer, (void *)rd_buffer,
     phys_mem.size) != 0)
printf"failed\n");
else
printf("passed\n");.

//free the buffer
dsp21k_free_phys_memory(processor, &phys_mem);
  ...
```

See Also

- dsp21k_free_phys_memory
- dsp21k_get_phys_memory
- dsp21k_phys_memory_count
- dsp21k_rd_phys_memory
- dsp21k_rd_phys_memory8
- dsp21k_wr_phys_memory
- dsp21k_wr_phys_memory8

For further information on accessing host physical memory with the HIL, refer to section 2.4.

dsp21k_bank_depth

Prototype `int dsp21k_bank_depth (PDSP21K processor, int bank);`

Description This function returns the depth, in kilowords, of the external memory bank specified by **`bank`**.

Arguments

`processor`	Pointer to processor structure
`bank`	Number of the external memory bank

Returns The function returns the depth of the external memory bank in kilowords. If it encounters an error, it returns a negative HIL error value.

Example None

See Also
- dsp21k_bank_width
- dsp21k_bank_size

dsp21k_bank_size

Prototype `int dsp21k_bank_size (PDSP21K processor)`

Description This function returns the size, in words, reserved for the external memory banks as determined by the MSIZE value.

Arguments

`processor`	Pointer to processor structure

Returns This function returns the size, in words, reserved for the external memory banks as determined by the MSIZE value.

Example None

See Also
- dsp21k_bank_depth
- dsp21k_bank_width

dsp21k_bank_start

Prototype `int dsp21k_bank_start (PDSP21K processor, int bank)`

Description This function returns the DSP starting address of the given external memory bank.

Arguments

processor	Pointer to processor structure
bank	External memory bank

Returns This function returns the DSP starting address of the given external memory bank.

Example

```
printf("External bank 0 starts at 0x%08x\n",
     dsp21k_bank_start(processor, 0));
```

See Also

- dsp21k_bank_depth
- dsp21k_bank_width

dsp21k_bank_width

Prototype `int dsp21k_bank_width (PDSP21K processor, int bank);`

Description This function returns the width, in bits, of the external memory bank specified by **`bank`**.

Arguments

`processor`	Pointer to processor structure
`bank`	Number of the external memory bank

Returns The function returns the width of the external memory bank in bits. If it encounters an error, it returns a negative HIL error value.

Example None

See Also
- dsp21k_bank_depth
- dsp21k_bank_size
- dsp21k_mem_width

dsp21k_bc_cfg_proc

Prototype `int dsp21k_bc_cfg_proc(PDSP21K processor)`

Description Use this function to configure all of the processors in the cluster at once. Calling this function is equivalent to calling *dsp21k_cfg_proc* for each processor in the cluster at the same time.

Arguments

`processor`	Pointer to processor structure

Returns The function returns 0 if it succeeds (DSP21K_SUCCESS). If it does not succeed, it returns a negative HIL error value.

Example None

See Also

- dsp21k_cfg_proc
- dsp21k_is_bc_capable
- dsp21k_reset_bd

dsp21k_bc_dl_32s

Prototype

```
int dsp21k_bc_dl_32s(PDSP21K processor, ULONG
    dsp_addr,UINT count, LPULONG val)
```

Description From the host buffer that **val** points to, this function downloads **count** bit values to the memory of each of the DSPs in the cluster, starting at *dsp_addr*. Calling this function is equivalent to calling *dsp21k_dl_32s* for each processor in the cluster at the same time.

Arguments

processor	Pointer to processor structure
dsp_addr	DSP memory address
count	Number of 32-bit values to transfer
val	Address of host buffer

Returns The function returns 0 if it succeeds (DSP21K_SUCCESS). If it does not succeed, it returns a negative HIL error value.

Example None

See Also

- dsp21k_dl_32s
- dsp21k_is_bc_capable

dsp21k_bc_reset_proc

Prototype `int dsp21k_bc_reset_proc(PDSP21K processor)`

Description This function performs a processor (soft) reset on each processor in the cluster at once. Use this function to reset all processors in a cluster on a broadcast capable board. Calling this function is equivalent to calling *dsp21k_reset_proc* for each processor in the cluster at the same time.

Arguments

`processor`	Pointer to processor structure

Returns The function returns 0 if it succeeds (DSP21K_SUCCESS). If it does not succeed, it returns a negative HIL error value.

Example None

See Also

- dsp21k_reset_proc
- dsp21k_is_bc_capable

dsp21k_bc_start

Prototype `int dsp21k_bc_start(PDSP21K processor)`

Description This function releases all of the processors in the cluster from reset at once. The programs on all of the processors will begin executing at the reset vector. Calling this function is equivalent to calling *dsp21k_start* for each processor in the cluster at the same time.

Note *For TigerSHARC (ADSP_TSxxx) processors, this function will replace the instruction at the reset vector of each processor in the cluster with the instruction at the reset vector of the current processor. Reset vector instructions that differ from the current processor's reset vector instruction will be lost and will not be executed by the other processors. The other processors will instead execute the current processor's reset vector instruction.*

Arguments

`processor`	Pointer to processor structure

Returns The function returns 0 if it succeeds (DSP21K_SUCCESS). If it does not succeed, it returns a negative HIL error value.

Example None

See Also
- dsp21k_start
- dsp21k_is_bc_capable

dsp21k_bc_wiop

Prototype

```
void dsp21k_bc_wiop (PDSP21K processor, USHORT offset,
    ULONG value)
```

Description

This function writes **`value`** to the IOP register on all of the processors in the cluster that **`offset`** specifies at once. Calling this function is equivalent to calling *dsp21k_wiop* for each processor in the cluster at the same time.

Arguments

`processor`	Pointer to processor structure
`offset`	Address of IOP register
`value`	32-bit value to be written

Returns

None

Example

None

See Also

- dsp21k_wiop
- dsp21k_is_bc_capable

dsp21k_block_depth

Prototype `int dsp21k_block_depth (PDSP21K processor, int block)`

Description This function returns the depth of the given internal memory block in kilowords. To get the depth of PM (Program Memory), use block #0. To get the depth of DM (Data Memory), use block #1.

Arguments

`processor`	Pointer to processor structure
`block`	Zero-based number of the internal memory block

Returns This function returns the depth of the given internal memory block in kilowords.

Example

```
printf("Memory block 0 is %dk wordsln",
     dsp21k_block_depth(processor, 0));
```

See Also dsp21k_block_start

dsp21k_block_start

Prototype `int dsp21k_block_start (PDSP21K processor, int block)`

Description This function returns the DSP starting address of the given internal memory block. To get the starting address of PM (Program Memory), use block #0. To get the starting address of DM (Data Memory), use block #1.

Arguments

`processor`	Pointer to processor structure
`block`	Zero-based number of the internal memory block

Returns This function returns the DSP starting address of the given internal memory block.

Example

```
printf("Memory block 0 starts at 0x%06x\n",
     dsp21k_block_start(processor, 0));
```

See Also dsp21k_block_depth

dsp21k_board_name

Prototype `char *dsp21k_board_name (PDSP21K processor);`

Description This function returns a pointer to a null-terminated string containing the name of the board type for **`processor`**. See `const21.h` for a list of valid board types.

Arguments

`processor`	Pointer to processor structure

Returns The function returns a pointer to the first character of a null-terminated character string containing the board's name. If the board type is not known or the processor pointer is invalid, the function returns a pointer to a string containing "unknown."

Example None

See Also

- dsp21k_board_type
- dsp21k_dsp_name
- dsp21k_get_board_name
- dsp21k_get_dsp_name
- dsp21k_dsp_type

dsp21k_board_type

Prototype `int dsp21k_board_type(PDSP21K processor);`

Description This function returns an integer value identifying the board type for **`processor`**. See `const21.h` for a list of valid board types.

Arguments

`processor`	Pointer to processor structure

Returns The function returns an integer containing the board's type. If the board type is not known or the processor pointer is invalid, the function returns a value of "0."

Example None

See Also

- dsp21k_board_name
- dsp21k_dsp_type
- dsp21k_dsp_name
- dsp21k_get_board_name
- dsp21k_get_dsp_name

dsp21k_build

Prototype `char *dsp21k_build (void);`

Description This function returns a pointer to a null-terminated string that identifies the build of the library.

Arguments None

Returns The function returns a pointer to the first character of a null-terminated character string that identifies the library build.

Example None

See Also
- dsp21k_version
- dsp21k_os_target
- dsp21k_prn_version

dsp21k_cfg_proc

Prototype `int dsp21k_cfg_proc(PDSP21K processor);`

Description Use this function to configure the processor before accessing memory or loading a program. Its primary use is to program the processor's registers so that the program can access memory properly. The default register values are stored in the device's EEPROM or configuration file. Since the reset will return these registers to their default values, you should call this function after the processor or the board has been reset. The *dsp21k_load_exe()* function automatically calls this function. See *dsp21k_reset_bd* function page for the suggested method for configuring different types of processors.

Warning *Calling this function while the processor is running may have undesired consequences.*

Arguments

`processor` Pointer to processor structure

Returns The function returns 0 if it succeeds (DSP21K_SUCCESS). If it does not succeed, it returns a negative HIL error value. Refer to section 2.2.5 for more information on HIL error values.

Example

```
PDSP21K processor;
PDSP21K dsp;
int i;
int num_dsps;

//open processor 1 on device 0
processor = dsp21k_open_by_id(0, 1);

if(processor)
{
```

```
        num_dsps = dsp21k_num_dsps(processor);
        //reset board
        dsp21k_reset_bd(processor);

        //processors need to be configured after
        //a board reset for host access
        //configure all processors on the board
    for(i = 1; i <= num_dsps; i++)
        {
               dsp = dsp21k_open_by_id(0, i);
               if(dsp)
        {
               dsp21k_cfg_proc(dsp);
               dsp21k_close(dsp);
        }
     }
     dsp21k_close(processor);
}
```

See Also

- dsp21k_load_exe
- dsp21k_reset_bd
- dsp21k_reset_proc
- dsp21k_start
- dsp21k_proc_running
- dsp21k_bc_cfg_proc

dsp21k_close

Prototype `int dsp21k_close (PDSP21k processor);`

Description If **`processor`** has been successfully opened with *dsp21k_open, dsp21k_open_all, dsp21k_open_by_id,* or *dsp21k_open_by_file,* this function frees all memory allocated in the *open* call. This function should be called once for every time an open function was called for the processor. If the processor being closed is the last open processor on a board, this function will disable interrupts if they are enabled.

Arguments

`processor` Pointer to processor structure

Returns The function returns 0 if it succeeds (DSP21K_SUCCESS). If it does not succeed, it returns a negative HIL error value. Refer to section 2.2.5 for more information on HIL error values.

Example

```
#include <dsp21k.h>
void main(int argc, char *argv[])
{
  PDSP21K processor;
  //open first processor on device 0
  processor = dsp21k_open(0);
  if (processor)
  {
    dsp21k_load_exe(processor,argv[1]);
    dsp21k_start(processor);
    dsp21k_close(processor);
  }
}
```

See Also

- dsp21k_open
- dsp21k_open_by_id
- dsp21k_open_all
- dsp21k_close_all

dsp21k_close_all

Prototype `void dsp21k_close_all ( void );`

Description This function closes all open processors.

Arguments None

Returns None

Example None

See Also
- dsp21k_open_all
- dsp21k_get_proc
- dsp21k_close

dsp21k_dbl_to_xflt

Prototype

```
float dsp21k_dbl_to_xflt(double d, unsigned char *
    xbyte);
```

Description This function converts a 64-bit double to a 40-bit extended float. The function return value passes back the most significant 32 bits of the extended float and places the extended 8 bits in **xbyte**.

Arguments

d	64-bit double
xbyte	Byte where extended 8 bits are placed upon return

Returns The function returns a 32-bit floating point converted from the passed double.

Example None

See Also dsp21k_xflt_to_dbl

dsp21k_device_num

Prototype `int dsp21k_device_num(PDSP21K processor);`

Description This function returns the device number of the board that the currently selected processor is on. A board's device number can be modified by using the BittWare Configuration Manager.

Arguments

`processor`	Pointer to processor structure

Returns This function returns the device number of the board that the currently selected processor is on.

Example None

See Also
- dsp21k_board_type
- dsp21k_dsp_type

dsp21k_dir

Prototype `char * dsp21k_dir()`

Description This function returns the installation location of the DSP21k-SF Toolkit, if possible.

Arguments None.

Returns If the DSP21KSF environment variable is set, this function returns its value. Otherwise, this function routines the default installation location of the DSP21k-SF Toolkit.

Example None.

See Also None

dsp21k_dl_8s

Prototype

```
int dsp21k_dl_8s (PDSP21K processor, ULONG dsp_addr,
     ULONG count, LP UCHAR val)
```

Description From the host buffer pointed to by **val**, this function downloads **count** 8-bit values to the DSP memory address (**dsp_addr**). This function will only write to 8-bit wide memory such as flash.

Arguments

processor	Pointer to processor structure
dsp_addr	DSP memory address
count	Number of 8-bit values to write
val	Address of host buffer

Returns This function returns 0 if it succeeds (DSP21K_SUCCESS). If it does not succeed, it returns a negative HIL error value.

Example

```
UCHAR byte_value = 0xf0;

//write a single byte to flash memory
//NOTE - this will not alter flash memory, you
// muse use the flash library for that.
dsp21k_dl_8s(processor, 0x2800000, 1, &byte_value);
```

See Also dsp21k_dl_16s

dsp21k_dl_16s

Prototype

```
int dsp21k_dl_16s (PDSP21K processor, ULONG dsp_addr,
     ULONG count, LP USHORT val)
```

Description

From the host buffer pointed to by **val**, this function downloads **count** 16-bit values to the DSP memory address (**dsp_addr**). This function will only read from 16-bit wide memory such as the SHARC's short word memory.

Arguments

processor	Pointer to processor structure
dsp_addr	DSP memory address
count	Number of 16-bit values to write
val	Address of host buffer

Returns

This function returns 0 if it succeeds (DSP21K_SUCCESS). If it does not succeed, it returns a negative HIL error value.

Example

```
USHORT shrt_value = 0xbeef;

dsp21k_dl_16s(processor, 0x80000, 1, &shrt_value);

See Also
dsp21k_ul_16s
```

See Also

dsp21k_ul_16s

dsp21k_dl_32s

Prototype

```
int dsp21k_dl_32s (PDSP21K processor,
     unsigned long dsp_addr, unsigned int count, void *
     val);
```

Description

From the host buffer that **`val`** points to, this function downloads **`count`** 32-bit values to the DSP's memory, which starts at **`dsp_addr`**. You can determine global variable addresses with *dsp21k_get_addr*.

Arguments

`processor`	Pointer to processor structure
`dsp_addr`	DSP memory address
`count`	Number of 32-bit values to transfer
`val`	Address of host buffer

Returns

The function returns 0 if it succeeds (DSP21K_SUCCESS). If it does not succeed, it returns a negative HIL error value. Refer to section 2.2.5 for more information on HIL error values.

Example

```
PDSP21K processor;
ULONG   dsp_addr;
long    buffer[2];

processor = dsp21k_open( 0 );
buffer[0] = 0xdeadface;
buffer[1] = 0xcafebeef;
dsp_addr  = 0x30000;

dsp21k_dl_32s( processor, dsp_addr, 2, &buffer[0] );

dsp21k_close( processor );
```

See Also

- dsp21k_ul_32s

- dsp21k_get_addr
- dsp21k_dl_int
- dsp21k_ul_int

dsp21k_dl_48

Prototype

```
int dsp21k_dl_48 (PDSP21K processor, unsigned long
     dsp_addr, unsigned short * dh, unsigned short *
     dm, unsigned short * dl);
```

Description This function downloads a single 48-bit value, which is defined by **dh** : **dm** : **dl**, to the DSP's memory. The DSP memory starts at **dsp_addr**. Determine DSP global variable addresses with *dsp21k_get_addr*.

Arguments

processor	Pointer to processor structure
dsp_addr	DSP memory address
dh	Upper 16 bits of 48-bit word
dm	Middle 16 bits of 48-bit word
dl	Lower 16 bits of 48-bit word

Returns The function returns 0 if it succeeds (DSP21K_SUCCESS). If it does not succeed, it returns a negative HIL error value. Refer to section 2.2.5 for more information on HIL error values.

Example

```
PDSP21K processor;
ULONG   dsp_addr;
USHORT  buffer[3];

processor = dsp21k_open( 0 );
buffer[0] = 0xdead;
buffer[1] = 0xcafe;
buffer[2] = 0xbeef;
dsp_addr  = 0x21000;

// Download "0xdead:cafe:beef".
dsp21k_dl_48( processor, dsp_addr,
&buffer[0], &buffer[1], &buffer[2] );

dsp21k_close( processor );
```

See Also

- dsp21k_get_addr
- dsp21k_ul_48
- dsp21k_ul_48s
- dsp21k_dl_48s

dsp21k_dl_48s

Prototype

```
int dsp21k_dl_48s (PDSP21K processor, ULONG
    dsp_addr,UINT count, LPUSHORT val);
```

Description

This function downloads **count** 48-bit values, each of which are defined by three USHORTs contained in **val**, to the DSP's memory, starting at **dsp_addr**. Determine DSP global variable addresses with *dsp21k_get_addr*.

Arguments

processor	Pointer to processor structure
dsp_addr	DSP memory address
count	Number of 48-bit values
val	Pointer to a buffer containing **count** * 3 USHORTS

Returns

None

Example

```
PDSP21K processor;
ULONG dsp_addr;
USHORT buffer[3];
processor = dsp21k_open( 0 );
buffer[0] = 0xdead;
buffer[1] = 0xcafe;
buffer[2] = 0xbeef;
dsp_addr = 0x41000;
// Download "0xdead:cafe:beef".
dsp21k_dl_48s( processor, dsp_addr, 1, buffer );
dsp21k_close( processor );
```

See Also

- dsp21k_dl_48
- dsp21k_get_addr
- dsp21k_ul_48
- dsp21k_ul_48s

dsp21k_dl_64s

Prototype

```
int dsp21k_dl_64s (PDSP21K processor, ULONG dsp_addr,
     ULONG count, LP ULONG val)
```

Description

From the host buffer pointed to by **val**, this function downloads **count** 64-bit values to the DSP memory address (**dsp_addr**). This function will only read from 64-bit wide memory such as the ADSP-2116x's long word memory.

Arguments

processor	Pointer to processor structure
dsp_addr	DSP memory address
count	Number of 64-bit values to write
val	Address of host buffer

Returns

This function returns 0 if it succeeds (DSP21K_SUCCESS). If it does not succeed, it returns a negative HIL error value.

Example

```
ULONG val64[2] = {0x01234567, 0x89abcdef};

//write a 64-bit value at the start of long word memory
     on a 21160
dsp21k_dl_64s(processor, 0x20000, 1, val64);
```

See Also

dsp21k_ul_64s

dsp21k_dl_dbl

Prototype

```
void dsp21k_dl_dbl(PDSP21K processor, unsigned long
     dsp_addr, double val);
```

Description

This function writes a single 64-bit float (**`val`**) to the DSP's memory, which starts at **`dsp_addr`**. Determine global variable addresses with *dsp21k_get_addr*.

Arguments

`processor`	Pointer to processor structure
`dsp_addr`	DSP memory address
`val`	Value to write

Returns

None

Example

None

See Also

- dsp21k_get_addr
- dsp21k_ul_dbl
- dsp21k_dl_32s

dsp21k_dl_flt

Prototype

```
void dsp21k_dl_flt(PDSP21K processor, unsigned long
     dsp_addr, float val);
```

Description

This function writes a single 32-bit float (**`val`**) to the DSP's memory, which starts at **`dsp_addr`**. Determine global variable addresses with *dsp21k_get_addr*.

Arguments

`processor`	Pointer to processor structure
`dsp_addr`	DSP memory address
`val`	Value to write

Returns

None

Example

None

See Also

- dsp21k_get_addr
- dsp21k_ul_flt
- dsp21k_dl_32s

dsp21k_dl_int

Prototype

```
void dsp21k_dl_int (PDSP21K processor, unsigned long
     dsp_addr, int val);
```

Description This function writes a single 32-bit integer (**`val`**) to the DSP's memory, which starts at **`dsp_addr`**. Determine global variable addresses with *dsp21k_get_addr*.

Arguments

`processor`	Pointer to processor structure
`dsp_addr`	DSP memory address
`val`	Value to write

Returns None

Example None

See Also

- dsp21k_get_addr
- dsp21k_ul_int
- dsp21k_dl_32s

dsp21k_dl_sctn32

Prototype

```
int dsp21k_dl_sctn32 (PDSP21K processor, ULONG
     dsp_addr, UINT count, UINT size, LPUCHAR val)
```

Description From the host buffer pointed to by **`val`**, this function downloads **`count`** program section values, each of **`size`** bytes, to DSP data memory, starting at the DSP address (**`dsp_addr`**). The program sections must be in Extensible Linker Format (ELF), and the values must be byte-packed into **`val`**.

Arguments

`processor`	Pointer to processor structure
`dsp_addr`	DSP data memory address
`count`	Number of values to download
`size`	size of each value in bytes
`val`	Address of host buffer

Returns This function returns 0 if it succeeds (DSP21K_SUCCESS). If it does not succeed, it returns a negative HIL error value.

Example None

See Also

- dsp21k_dl_sctn48
- dsp21k_ul_sctn32
- dsp21k_ul_sctn48

dsp21k_dl_sctn48

Prototype

```
int dsp21k_dl_sctn48 (PDSP21K processor, ULONG
     dsp_addr, UINT count, UINT size, LPUCHAR val)
```

Description From the host buffer pointed to by **val**, this function downloads **count** program section values, each of **size** bytes, to DSP data memory, starting at the DSP address. The program sections must be in Extensible Linker Format (ELF), and the values must be byte-packed into **val**.

Arguments

processor	Pointer to processor structure
dsp_addr	DSP memory address
count	Number of values to download
size	size of each value in bytes
val	Address of host buffer

Returns This function returns 0 if it succeeds (DSP21K_SUCCESS). If it does not succeed, it returns a negative HIL error value.

Example

```
UCHAR ibuf40005[6] = { 0x07, 0x3e, 0x00, 0xff, 0xff,
     0xff };

// Write "jump (pc, -1)" at 40005, the 21160 reset
     vector
dsp21k_dl_sctn48(processor, 0x40005L, 1, 6, ibuf40005);
```

See Also

- dsp21k_ul_sctn48
- dsp21k_dl_sctn32
- dsp21k_ul_sctn32

dsp21k_dl_xflt

Prototype

```
void dsp21k_dl_xflt (PDSP21K processor, unsigned long
     dsp_addr, double val);
```

Description

This function writes a single 40-bit float (**`val`**) to the DSP's memory, which starts at **`dsp_addr`**. You can determine global variable addresses with the *dsp21k_get_addr* function.

Arguments

`processor`	Pointer to processor structure
`dsp_addr`	DSP memory address
`val`	Value to write

Returns

None

Example

None

See Also

- dsp21k_get_addr
- dsp21k_ul_xflt

dsp21k_dma_start

Prototype

```
int dsp21k_dma_start (PDSP21K processor,
    DSP21K_DMA_XFER * dma)
```

Description

This function sets up DMA registers in the SharcFIN or TigerFIN to transfer data using the parameters given in the dma structure. See the const21k.h header file for possible field definitions.

Arguments

processor Pointer to processor structure.

dma Address of DSP21K_DMA_XFER whose members are:

- **pci_addr:** Address of host physical memory buffer or any valid PCI memory address
- **dsp_addr:** DSP address (can be an SDRAM address)
- **count**: Number of 64-bit words to transfer
- **channel**: FIN_DMA_CHAN0 or FIN_DMA_CHAN1*
- **direction**: FIN_DMA_PC_TO_DSP or FIN_DMA_DSP_TO_PC
- **burst_size**: 8, 16, 32, or 64
- **burst_control**: FIN_DMA_BURST_ENABLE or FIN_DMA_BURST_DISABLE
- **interrupt_control**: FIN_DMA_INTRPT_ENABLE or FIN_DMA_INTRPT_DISABLE
- **bus_lock_control**: FIN_DMA_BUSLOCK_ENABLE or FIN_DMA_BUSLOCK_DISABLE†
- **stride**: address increment‡
- **dma_width**: FIN_DMA_WIDTH_64 or FIN_DMA_WIDTH_32†
- **wait_control**: FIN_DMA_WAIT or FIN_DMA_NO_WAIT

* On the SharcFIN, channel refers to the PCI master channel. On the SFIN-101 and SFIN-201, PCI master channel is always 0 and this field refers to the ADSP-TS101 and ADSP-TS201's DMA channel. For Tiger external memory transfers, this must always be channel 0.
† This parameter is only used for SharcFIN DMAs.
‡ Stride is 0-255 for SharcFIN and is a 16-bit value written into ADSP-TS101 or ADSP-TS201 XMODIFY DMA register for SFIN-101/SFIN-201 DMAs.

Returns The function returns 0 if it succeeds (DSP21K_SUCCESS). If it does not succeed, it returns a negative HIL error value.

Example

```
if(dsp21k_is_dma_capable(processor))
{
       DSP21K_DMA_XFER fin_dma;
       DSP21K_PHYS_MEMORY phys_mem;

       //allocate a 1024 bytes (128 64-bit words)
       phys_mem.phys_addr = 0;
       phys_mem.size = 0x1000;
       if(dsp21k_alloc_phys_memory(processor,
    &phys_mem) < 0)
       {
              printf("Could not allocate physical
                memory\n");
              return;
       }
       //see const21k.h for dma definitions
       fin_dma.burst_control =FIN_DMA_BURST_ENABLE;
       fin_dma.burst_size =FIN_DMA_BURST_32;
       fin_dma.bus_lock_control =
    FIN_DMA_BUSLOCK_DISABLE;
       fin_dma.channel =   FIN_DMA_CHAN0;
       fin_dma.count = phys_mem.size / 8;//in 64-bit
    words
       fin_dma.direction = FIN_DMA_DSP_TO_PC;
       fin_dma.dma_width = FIN_DMA_WIDTH_64;
       fin_dma.dsp_addr = 0x800000;//from SDRAM
       fin_dma.interrupt_control =
    FIN_DMA_INTRPT_DISABLE;
       fin_dma.pci_addr = phys_mem.phys_addr;//host
    physical memory address
```

```
        fin_dma.stride = 1;                    //
        fin_dma.wait_control = FIN_DMA_NO_WAIT;

        if(dsp21k_dma_start(processor, &fin_dma) < 0)
        {
               printf("DMA error\n");
               return;
        }

        while(!dsp21k_is_dma_complete(processor,
            &fin_dma))
        {
               Sleep(1); //or use another thread delay
                   function
        }
}
```

See Also

- dsp21k_is_dma_capable
- dsp21k_is_dma_complete
- dsp21k_alloc_phys_memory
- dsp21k_free_phys_memory

dsp21k_dsp_name

Prototype `char *dsp21k_dsp_name(PDSP21K processor);`

Description This function returns a pointer to a null-terminated string that contains the name of the processor type for **`processor`**. See `const21k.h` for a list of valid processor types.

Arguments

`processor`	Pointer to processor structure

Returns The function returns a pointer to the first character of a null-terminated character string containing the processor's name. If the processor type is not known or the processor pointer is invalid, the function returns a pointer to a string containing "unknown."

Example None

See Also

- dsp21k_dsp_type
- dsp21k_board_name
- dsp21k_board_type
- dsp21k_get_board_name
- dsp21k_get_dsp_name

dsp21k_dsp_rev

Prototype `int dsp21k_dsp_rev(PDSP21K processor);`

Description This function returns the DSP revision number of this processor. If the device was opened using an EEPROM Configuration File, this is the DSP revision number from the file. Otherwise, the number is from the board's EEPROM.

Arguments

`processor`	Pointer to processor structure

Returns This function returns the DSP revision number of this processor.

Example None

See Also
- dsp21k_dsp_type
- dsp21k_num_dsps

dsp21k_dsp_type

Prototype `int dsp21k_dsp_type(PDSP21K processor);`

Description This function returns an integer value identifying the processor type for **`processor`**. This value comes from the device's EEPROM or configuration file. See `const21k.h` for a list of valid processor types.

Arguments

`processor`	Pointer to processor structure

Returns The function returns an integer value identifying the processor type for the processor.

Example None

See Also

- dsp21k_dsp_name
- dsp21k_board_type
- dsp21k_board_name
- dsp21k_get_board_name
- dsp21k_get_dsp_name
- dsp21k_dsp_rev
- dsp21k_num_dsps

dsp21k_err_msgs

Prototype `char *dsp21k_err_msgs(int num);`

Description This function returns a pointer to a null-terminated string that contains a description of the HIL error message number (**num**).

Arguments

num	HIL error message number

Returns The function returns a pointer to the first character of a null-terminated character string containing a description of the error message number (**num**). If the error is invalid or unknown, the function returns a pointer to an empty string (“ ”).

Example None

See Also dsp21k_get_last_error

dsp21k_fast_extmem_xfers

Prototype

```
int dsp21k_fast_extmem_xfers (PDSP21K processor,
     unsigned long address_param, unsigned long
     size_param);
```

Description This function speeds up access to external memories that cannot be directly accessed from the host. By default, the DSP21k-SF Toolkit uses a single unused word in the SHARC interrupt table to provide the buffer needed to access these external memories. Providing a larger buffer can significantly improve the speed at which it is accessed.

Note *The buffer space needed for faster access to external memory must be dedicated to this purpose and can not conflict with data or instructions of the SHARC program. The DSP21k-SF Toolkit can not count on any location to be available except the location already used.*

This function provides a way to tell the DSP21k-SF Toolkit the location and size of the buffer it can use for external access. You can reserve the space in your architecture file or in your DSP program and get the location by calling *dsp21k_get_address*.

Note *Once this function has been called, the DSP21k-SF Toolkit will continue to use this location until it is told to use a different address.*

You can restore the default values, which are always safe to use, for the external memory buffer by calling this function with a value of zero (0) for the length of the buffer.

Note *This function is only useful if the host must use the DSP to access external memory. For example, this function does nothing when called with a processor on a Hammerhead-PCI board because the host accesses external memory directly.*

Arguments

processor	Pointer to processor structure
address_param	SHARC address of buffer
size_param	Size of buffer

Returns The function returns 0 if it succeeds (DSP21K_SUCCESS). If it does not succeed, it returns a negative HIL error value. Refer to section 2.2.5 for more information on HIL error values.

Example None

See Also None

dsp21k_free_labels

Prototype `int dsp21k_free_labels(PDSP21K processor);`

Description This function frees the memory used to store global addresses that are allocated when the executable file is downloaded (see *dsp21k_load_exe*), or its symbols are loaded (see *dsp21k_load_symbols*).

Arguments

`processor`	Pointer to processor structure

Returns If the function is successful, it will return zero. If unsuccessful, it will return non-zero.

Example None

See Also

- dsp21k_load_exe
- dsp21k_get_addr
- dsp21k_load_symbols
- dsp21k_labels_defined

dsp21k_free_phys_memory

Prototype

```
int dsp21k_free_phys_memory( PDSP21K processor,
    DSP21K_PHYS_MEMORY *phys_mem)
```

Description This function frees or unmaps a physical memory buffer on the host PC allocated or mapped by a previous call to *dsp21k_alloc_phys_memory*.

Arguments

processor Pointer to processor structure

phys_mem Address of a **DSP21K_PHYS_MEMORY** structure whose members are:

- **phys_addr**: Host physical address of allocated or mapped memory.
- **size**: Unused by this function
- **mem_ptr**: Unused by this function

Returns The function returns 0 if it succeeds (DSP21K_SUCCESS). If it does not succeed, it returns a negative HIL error value.

Example

```
  ...
DSP21K_PHYS_MEMORY phys_mem;
int i, failed;
  ...

//MUST set phys_addr to 0 if allocating new buffer
phys_mem.phys_addr = 0;
//allocate 4K bytes (1K 32-bit words)
phys_mem.size = 0x1000;
dsp21k_alloc_phys_memory(processor, &phys_mem);

//write a count to the buffer
for(i = 0; i < phys_mem.size / 4; i++)
{
```

```
//dereference as a 32-bit pointer
*((LPULONG)phys_mem.mem_ptr + i) = i;
}

//read back the count
for(i = 0; i < phys_mem.size / 4; i++)
{
if(i != *((LPULONG)phys_mem.mem_ptr + i))
{
       printf("failed\n");
       failed = TRUE;
       break;
}
}

if(!failed)
printf("passed\n");

//free the buffer
dsp21k_free_phys_memory(processor, &phys_mem);
  ...
```

See Also

- dsp21k_alloc_phys_memory
- dsp21k_phys_memory_count
- dsp21k_get_phys_memory

For further information on accessing host physical memory with the HIL, refer to section 2.4.

dsp21k_get_addr

Prototype

```
ULONG dsp21k_get_addr (PDSP21K processor, const char *
     name)
```

Description

This function searches for the symbol **name** in the symbol table of the currently loaded program. If the symbol is found, it will return the DSP address associated with **name**.

Arguments

processor	Pointer to processor structure
name	Global variable name (prepend underscores for C/C++ source)

Returns

The function returns the address of the variable if it is found. If it is not found, it returns the value 0xFFFFFFFF.

Example

```
//load primes program
if(dsp21k_load_exe(processor, "primes.dxe") ==
     DSP21K_SUCCESS)
{
       ULONG primes_addr;
       ULONG primes_buf[20];

       //start processor
       dsp21k_start(processor);

       //get address of primes array
       primes_addr = dsp21k_get_addr(processor,
     "_primes");

       if(primes_addr != 0xFFFFFFFF)
       {
              //upload first 20 primes
              dsp21k_ul_32s(processor, primes_addr, 20,
                     primes_buf);
       }
}
```

See Also

- dsp21k_free_labels
- dsp21k_get_next_symbol
- dsp21k_get_symbol
- dsp21k_labels_defined
- dsp21k_load_symbols
- dsp21k_symbol_count
- dsp21k_symbol_size
- dsp21k_symbol_width

dsp21k_get_board_name

Prototype `char *dsp21k_get_board_name (int board_type_number);`

Description This function returns a pointer to a null-terminated string containing the name of the board type for the associated **`board_type_number`**. See `const21k.h` for a list of valid board types.

Arguments

`board_type_number` Number of board

Returns This function returns a pointer to the first character of a null-terminated character string containing the board's name. If the board type is not known or is invalid, the function returns a pointer to a string containing "unknown."

Example None

See Also

- dsp21k_board_name
- dsp21k_get_dsp_name
- dsp21k_dsp_name
- dsp21k_dsp_type
- dsp21k_board_type

dsp21k_get_device_info

Prototype `int dsp21k_get_device_info (PDSP21K processor, int DSP21K_DEVICE_CFG * cfg)`

Description This function fills in a **DSP21K_DEVICE_CFG** structure containing PCI base address registers and their respective sizes, port addresses, and the interrupt number for the device that the processor is on.

Arguments

processor Pointer to processor structure

cfg Address of a **DSP21K_DEVICE_CFG** structure whose members are:

- **pci_badr[6]**: array of physical addresses of mapped memory regions
- **pci_port[6]**: array of physical port addresses
- **badr_length[6]**: array of lengths of mapped memory regions in bytes
- **port_length[6]**: lengths of port regions in bytes
- **interrupt_number**: interrupt number for this device

Returns The function returns 0 if it succeeds (DSP21K_SUCCESS). If it does not succeed, it returns a negative HIL error value.

Example

```
DSP21K_DEVICE_CFG cfg;

if(dsp21k_get_device_info(processor, &cfg) ==
     DSP21K_SUCCESS)
{
printf("BAR2's physical address is %08x",
     cfg.pci_badr[2]);
}
```

See Also None

dsp21k_get_dsp_name

Prototype `char *dsp21k_get_dsp_name (int dsp_type_number);`

Description This function returns a pointer to a null-terminated string that contains the name of the processor type for the **`dsp_type_number`**. See `const21k.h` for a list of valid processor types.

Argument

`dsp_type_number` Code identifying the processor type:

2 = ADSP-21020
3 = ADSP-21060
4 = ADSP-21062
5 = ADSP-21061
6 = ADSP-21065
7 = ADSP-21160
9 = ADSP-21161
10 = ADSP-TS101
11 = FPGA
12 = ADSP-TS201

Returns This function returns a pointer to the first character of a null-terminated character string containing the processor's name. If the processor type is not known or is invalid, the function returns a pointer to a string containing "unknown."

Example None

See Also

- dsp21k_dsp_name
- dsp21k_dsp_type
- dsp21k_get_board_name
- dsp21k_board_name

- dsp21k_board_type

dsp21k_get_last_error

Prototype `int dsp21k_get_last_error();`

Description Functions without specific HIL error return values (such as *dsp21k_open*) instead set a global error value. This function will return the most recent global error value.

Arguments None

Returns This function returns the last global error value. If none, it returns 0 (DSP21K_SUCCESS).

Example None

See Also dsp21k_err_msgs

dsp21k_get_next_symbol

Prototype `LPCHAR dsp21k_get_next_symbol(PDSP21K processor, int symbol_index);`

Description This function returns the global symbol at index **`symbol_index`** in the symbol table. Index **`symbol_index`** is zero-based. To get the number of symbols in the symbol table, call *dsp21k_symbol_count*.

Arguments

`processor`	Pointer to processor structure
`symbol_index`	Zero-based symbol table index of symbol

Returns This function returns the global symbol at index **`symbol_index`** in the symbol table. If the index is greater than or equal to the number of symbols, the function returns NULL.

Example

```
#include <dsp21k.h>
void main(int argc, char *argv[])
{
PDSP21K processor;
int symbol_index, symbol_count;
       //open first processor on device 0
       processor = dsp21k_open(0);
if (processor)
{
       dsp21k_load_exe(processor, argv[2]);

       printf("DSP program %s symbol list\n", argv[2]);
       symbol_count = dsp21k_symbol_count(processor);
       for(symbol_index = 0; symbol_index <
          symbol_count; symbol_index++)
       {
              printf("\t%s\n",
                     dsp21k_get_next_symbol(processor,
                      symbol_index);
       }
       dsp21k_close(processor);
}
}
```

See Also

- dsp21k_load_exe
- dsp21k_free_labels
- dsp21k_labels_defined
- dsp21k_load_symbols
- dsp21k_symbol_count

dsp21k_get_pcirq

Prototype `int dsp21k_get_pcirq(PDSP21K processor);`

Description This function returns the current DSP-to-PC interrupt number.

Arguments

`processor`	Pointer to processor structure

Returns The function returns the interrupt number. It will return 0 if unused.

Example None

See Also dsp21k_get_device_info

dsp21k_get_phys_memory

Prototype

```
int dsp21k_get_phys_memory (int index,
    DSP21K)_PHYS_MEMORY *phys_mem)
```

Description This function returns the number of Host PC physical memory buffers allocated with previous calls to *dsp21k_alloc_phys_memory*.

Arguments

index	Gives the physical memory buffer number (0 to count –1). Use *dsp21k_phys_memory_count* to determine count.
phys_addr	Gives the host physical memory address accessible by the PCI device.
size	Gives the size of the physical memory buffer in bytes.
mem_ptr	A virtual memory pointer accessible by the Host PC.

Returns The function returns 0 if it succeeds (DSP21K_SUCCESS). If it does not succeed, it returns a negative HIL error value.

Example

```
//free all previously allocated
//physical memory buffers
for(i = 0; i < dsp21k_phys_memory_count(); i++)
{
       dsp21k_get_phys_memory(i, &phys_mem);
       dsp21k_free_phys_memory(processor, &phys_mem);
}
```

See Also

- dsp21k_alloc_phys_memory
- dsp21k_free_phys_memory
- dsp21k_phys_memory_count

For further information on accessing host physical memory with the HIL, refer to section 2.4.

dsp21k_get_proc

Prototype `PDSP21K dsp21k_get_proc ( long proc_index);`

Description This function allows access to DSPs that were opened by a previous call to *dsp21k_open_all*.

Arguments

`proc_index`	Index from 0 to (number returned by *dsp21k_open_all*) –1

Returns The function returns a pointer to the DSP in position **`proc_index`** opened by a previous call to *dsp21k_open_all*. The function returns NULL if **`proc_index`** is invalid.

Example None

See Also
- dsp21k_open_all
- dsp21k_close_all

dsp21k_get_symbol

Prototype

```
LPCHAR dsp21k_get_symbol(PDSP21K processor, ULONG
    address);
```

Description This function returns the symbol associated with the DSP address **address**. If a symbol does not exist at that address, the function returns NULL.

Arguments

processor	Pointer to processor structure
address	DSP address

Returns The function returns the name of the variable if one exists at the address given; otherwise, it returns NULL.

Example None

See Also

- dsp21k_load_exe
- dsp21k_free_labels
- dsp21k_load_symbols
- dsp21k_labels_defined

dsp21k_int_disable

Prototype `int dsp21k_int_disable(PDSP21K processor)`

Description This function destroys the interrupt thread, if it exists, and disables the interrupt. For further information on handling PCI interrupts, refer to section 2.5.

Arguments

`processor`	Pointer to processor structure

Returns The function returns 0 if it succeeds (DSP21K_SUCCESS). If it does not succeed, it returns a negative HIL error value.

Example

```
void isr_func( ULONG user_int_status,
       UCHAR mailbox_status, void * param )
{
       int * interrupt_count = (int *)param;

       *interrupt_count = *interrupt_count + 1;

       printf("PC received interrupt #%d from DSP",
     *interrupt_count);
}

int main(int argc, char * argv[])
{
       int interrupt_count;
       ULONG user_int_status;
       UCHAR mailbox_status;
  ...

       //METHOD 1: use isr_func for handling interrupts
       // pass interrupt count to handler
       if(dsp21k_int_enable( processor, isr_func,
    &interrupt_count) == DSP21K_SUCCESS)
              printf("interrupts enabled\n");
```

```
  ...

       if(dsp21k_int_disable(processor) ==
    DSP21K_SUCCESS)
              printf("interrupts disabled\n");
  ...

       //METHOD 2: do not pass a handler
       if(dsp21k_int_enable( processor, NULL, NULL) ==
    DSP21K_SUCCESS)
              printf("interrupts enabled\n");

       //disable after 10 interrupts
       while(interrupt_count < 10)
{
       if(dsp21k_int_wait(processor, &user_int_status,
    &mailbox_status) == DSP21K_SUCCESS)
              printf("PC received interrupt #%d from
    DSP", *interrupt_count);

}

  ...

if(dsp21k_int_disable(processor) == DSP21K_SUCCESS)
       printf("interrupts disabled\n");
  ...
}

}
```

See Also

- dsp21k_int_enable
- dsp21k_int_wait
- dsp21k_get_pcirq

dsp21k_int_dsp

Prototype `int dsp21k_int_dsp(PDSP21K processor, int irq_num)`

Description This function will cause the PC to generate the specified DSP interrupt. This function interrupts the processor according to the following table.

Interface Chip	irq_num	Interrupt type
SharcFIN, TigerFIN, T2FIN	mailbox number	mailbox
FINLite	not used	DSP interrupt
PLX	not used	doorbell

Arguments

processor	Pointer to processor structure
irq_num	DSP interrupt number

Returns The function returns 0 if it succeeds (DSP21K_SUCCESS). If it does not succeed, it returns a negative HIL error value.

Example

```
// Generate interrupt on DSP
dsp21k_int_dsp(processor, 0);
```

See Also None

dsp21k_int_enable

Prototype

```
int dsp21k_int_enable(PDSP21K processor,
     dsp21k_int_handler_fptr handler, void * param)
```

Description

This function enables the interrupt for the device that processor is on and then provides one of two different behaviors:

- If **handler** is not NULL, this function sets up the interrupt thread to call handler when the board receives an interrupt. The **param** parameter will get passed on to the **handler** function.
- If **handler** is NULL, this function returns and the *dsp21k_int_wait* function should be used to wait for interrupts.

For further information on handling PCI interrupts, refer to section 2.5

Arguments

processor	Pointer to processor structure
handler	Address of function to call on interrupt or NULL. Function must be in the form: **void my_int_handler(ULONG user_int_status, UCHAR mailbox_status, void * param);**
param	Pointer to user data to pass to the interrupt handler

Returns

The function returns 0 if it succeeds (DSP21K_SUCCESS). If it does not succeed, it returns a negative HIL error value.

Example

```
void isr_func( ULONG user_int_status, UCHAR
     mailbox_status, void * param )
{
       int * interrupt_count = (int *)param;

       *interrupt_count = *interrupt_count + 1;
```

```
        printf("PC received interrupt #%d from DSP",
      *interrupt_count);
}

int main(int argc, char * argv[])
{
        int interrupt_count;
        ULONG user_int_status;
        UCHAR mailbox_status;
...

        //METHOD 1: use isr_func for handling interrupts
        // pass interrupt count to handler
        if(dsp21k_int_enable( processor, isr_func,
      &interrupt_count) == DSP21K_SUCCESS)
                printf("interrupts enabled\n");
  ...

        if(dsp21k_int_disable(processor) ==
      DSP21K_SUCCESS)
                printf("interrupts disabled\n");
  ...

/       /METHOD 2: do not pass a handler
        if(dsp21k_int_enable( processor, NULL, NULL) ==
      DSP21K_SUCCESS)
                printf("interrupts enabled\n");

        //disable after 10 interrupts
        while(interrupt_count < 10)
{
        if(dsp21k_int_wait(processor, &user_int_status,
      &mailbox_status) == DSP21K_SUCCESS)
                printf("PC received interrupt #%d from
      DSP", *interrupt_count);

}

  ...

        if(dsp21k_int_disable(processor) ==
      DSP21K_SUCCESS)
                printf("interrupts disabled\n");
  ...
}
```

See Also

- dsp21k_int_disable
- dsp21k_int_wait
- dsp21k_get_pcirq

dsp21k_int_wait

Prototype

```
int dsp21k_int_wait(PDSP21K processor, LPULONG
    user_int_status, LPUCHAR mailbox_status)
```

Description

This function waits for the host to be interrupted by the device that processor is on. The contents of the interrupt status registers are returned in *user_int_status* and *mailbox_status*. The function will return only if an interrupt occured or the interrupt was disabled with a call to *dsp21k_int_disable*.

For further information on handling PCI interrupts, refer to section 2.4

Arguments

processor	Pointer to processor structure
user_int_status	Interrupt status - refer to section 2.5
mailbox_status	Interrupt status - refer to section 2.5

Returns

The function returns 0 if it succeeds (DSP21K_SUCCESS). If it does not succeed, it returns a negative HIL error value.

Example

```
int main(int argc, char * argv[])
{
       int interrupt_count;
       ULONG user_int_status;
       UCHAR mailbox_status;
  ...

       // do not pass a handler
       if(dsp21k_int_enable( processor, NULL, NULL) ==
    DSP21K_SUCCESS)
              printf("interrupts enabled\n");

  ...
       //disable after 10 interrupts
```

```
        while(interrupt_count < 10)
{
        if(dsp21k_int_wait(processor, &user_int_status,
      &mailbox_status) == DSP21K_SUCCESS)
                printf("PC received interrupt #%d from
      DSP", *interrupt_count);

}

  ...

        if(dsp21k_int_disable(processor) ==
      DSP21K_SUCCESS)
                printf("interrupts disabled\n");
...
}
```

See Also

- dsp21k_int_disable
- dsp21k_int_enable
- dsp21k_get_pcirq

dsp21k_is_bc_capable

Prototype `int dsp21k_is_bc_capable(PDSP21K processor)`

Description This function checks to make sure the broadcast functions will work for the cluster that the processor is on.

Arguments

`processor`	Pointer to processor structure

Returns This function returns TRUE if the processor is on a cluster that can be broadcast to; if not, it returns FALSE.

Example None

See Also

- dsp21k_bc_cfg_proc
- dsp21k_bc_dl_32s
- dsp21k_bc_reset_proc
- dsp21k_bc_start
- dsp21k_bc_wiop

dsp21k_is_dma_capable

Prototype `int dsp21k_is_dma_capable (PDSP21K processor)`

Description This function returns TRUE if the HIL DMA routines have been implemented for the PCI interface chip behind this processor, otherwise it returns FALSE.

Arguments

`processor`	Pointer to processor structure

Returns This function returns TRUE if the HIL DMA routines have been implemented for the PCI interface chip for this processor, otherwise it returns FALSE.

Example See *dsp21k_dma_start* for a complete example.

See Also

- dsp21k_dma_start
- dsp21k_is_dma_complete
- dsp21k_alloc_phys_memory
- dsp21k_free_phys_memory

dsp21k_is_dma_complete

Prototype

```
int dsp21k_is_dma_complete(PDSP21K processor,
     DSP21K_DMA_XFER * dma)
```

Description This function returns TRUE if the DMA started with dsp21k_dma_start is complete. Otherwise it returns FALSE.

Arguments

processor	Pointer to processor structure
dma	address of DSP21K_DMA_XFER that has been passed to dsp21k_dma_start function. The following members are used by this function: • **channel:** FIN_DMA_CHAN0 or FIN_DMA_CHAN1* • **direction:** FIN_DMA_PC_TO_DSP or FIN_DMA_DSP_TO_PC

* On the SharcFIN, channel refers to the PCI master channel. On the SFIN-101 and SFIN-201, PCI master channel is always 0 and this field refers to the ADSP-TS101 and ADSP-TS201's DMA channel. For Tiger external memory transfers, this must always be channel 0.

Returns This function returns TRUE if the DMA started with dsp21k_dma_start is complete. Otherwise it returns FALSE.

Example *See dsp21k_dma_start for a complete example.*

See Also

- dsp21k_is_dma_capable
- dsp21k_dma_start
- dsp21k_alloc_phys_memory
- dsp21k_free_phys_memory

dsp21k_labels_defined

Prototype `int dsp21k_labels_defined ( PDSP21K processor );`

Description This function determines if any labels are defined for **`processor`** and returns TRUE or FALSE to indicate whether labels are defined.

Note *This function does not check to see if a specific label is defined.*

Arguments

`processor` Pointer to processor structure

Returns This function returns non-zero (TRUE) if labels are defined for the processor. It returns zero (FALSE) if no labels are defined.

Example None

See Also

- dsp21k_free_labels
- dsp21k_load_symbols
- dsp21k_loaded_file
- dsp21k_get_addr
- dsp21k_load_exe

dsp21k_load_exe

Prototype

```
int dsp21k_load_exe(PDSP21K processor, const char *
    file)
```

Description

If the file exists, this function will download a file, with or without a path, that must be in the executable format produced directly by the Analog Devices linker. The function will first reset and configure the processor before downloading if the DEV_RESET_ON_LOAD setting is set for the processor (see *dsp21k_wr_bd_setting* and *dsp21k_rd_bd_setting*). By default, the DEV_RESET_ON_LOAD flag is set to TRUE. After calling this function, you can access global program symbols with the *dsp21k_get_addr* function. To start the program, call *dsp21k_start*.

Arguments

processor	Pointer to processor structure
file	Name of file to download

Returns

The function returns 0 if it succeeds (DSP21K_SUCCESS). If it does not succeed, it returns a negative HIL error value. Refer to section 2.2.5 for more information on HIL error values.

Example

```
#include <dsp21k.h>
void main(int argc, char *argv[])
{
PDSP21K processor;

//open first processor on device 0
processor = dsp21k_open(0);
if(processor)
{

       if(dsp21k_load_exe(processor, argv[1]) < 0)
       {
              printf("Error downloading executable
     file, \"%s\"\n", argv[1]);
```

```
        }
        else
                dsp21k_start(processor);

        dsp21k_close(processor);
}
}
```

Note *Starting with Release 6.30, the SYSCON register's IMDW bits are no longer automatically set in this function for 2106x boards. If your DSP program contains 40-bit data, set these bits yourself after calling dsp21k_start to start the DSP program.*

See Also

- dsp21k_cfg_proc
- dsp21k_free_labels
- dsp21k_get_addr
- dsp21k_get_next_symbol
- dsp21k_labels_defined
- dsp21k_load_symbols
- dsp21k_proc_running
- dsp21k_rd_bd_setting
- dsp21k_start
- dsp21k_symbol_count
- dsp21k_wr_bd_setting

dsp21k_load_symbols

Prototype

```
int dsp21k_load_symbols(PDSP21K processor, char * file-
    name);
```

Description This function loads the symbol table from the executable **`filename`** into the PC memory structures to allow functions that use symbol table information to access it. This function operates similarly to *dsp21k_load_exe*, but it does not download anything to the processor. It only loads data in the PC memory structures. This function is useful for accessing global variables in a DSP for which the original processor structure, used to download the processor, is not available.

Arguments

`processor`	Pointer to processor structure
`filename`	Pointer to a null-terminated string containing the name of the executable file

Returns The function returns 0 if it succeeds (DSP21K_SUCCESS). If it does not succeed, it returns a negative HIL error value. Refer to section 2.1.5 for more information on HIL error values.

Example None

See Also

- dsp21k_get_addr
- dsp21k_load_exe
- dsp21k_free_labels
- dsp21k_labels_defined

Note *The return value of this function has been modified from earlier versions.*

dsp21k_loaded_file

Prototype `char *dsp21k_loaded_file ( PDSP21K processor );`

Description This function returns a pointer to a null-terminated string containing the name of the file last loaded onto **`processor`**.

Arguments

`processor`	Pointer to processor structure

Returns The function returns a pointer to the first character of a null-terminated character string containing the loaded file's name. If no file is loaded or the processor pointer is invalid, the function returns NULL.

Example None

See Also dsp21k_load_exe

dsp21k_mem_width

Prototype `int dsp21k_mem_width(PDSP21K processor, ULONG dsp_addr)`

Description This function attempts to determine the width in bits of a DSP memory address.

Arguments

`processor`	Pointer to processor structure
`dsp_addr`	DSP memory address

Returns The function returns the width in bits of a DSP memory address. If the DSP memory address is unknown, this function will return 0.

Example None

See Also
- dsp21k_bank_width
- dsp21k_symbol_width

dsp21k_mpid

Prototype `int dsp21k_mpid (PDSP21K processor)`

Description This function returns the processor's multiprocessor ID.

Arguments

`processor`	Pointer to processor structure

Returns This function returns the processor's multiprocessor ID.

Example None

See Also dsp21k_num_dsps

dsp21k_msg

Prototype

```
int dsp21k_msg(PDSP21K processor,
     char * format [,argument] ...);
```

Description This is the function used to manage all formatted output messages from the library functions. Use this function for all message output to ensure that the messages are properly directed to the default or specified output function. This function will take no other action than to pass control to the message handler function. The default message handler function is *vprintf()*.

Arguments

processor	Pointer to processor structure
format	Format-control string
[,argument] ...	Optional arguments, variables of the type specified in the format-control string

Returns The function returns the number of bytes output.

Example

```
dsp21k_msg (processor, "This is a test of %s\n",
                "dsp21k_msg");
```

See Also dsp21k_msg_func

dsp21k_msg_func

Prototype

```
int dsp21k_msg(PDSP21K processor, char * format [,argu-
     ment] ...);
```

Description This is the function used to manage all formatted output messages from the library functions. Use this function for all message output to ensure that the messages are properly directed to the default or specified output function. This function will take no action other than to pass control to the message handler function. The default message handler function is *vprintf()*.

Arguments

processor	Pointer to processor structure
format	Format-control string
[,argument]...	Optional arguments, variables of the type specified in the format-control string

Returns None

Example

```
#include <stdarg.h>

int my_msg_handler(const char * format, va_list
     arglist)
{
     printf("MY MESSAGE HANDLER: ");
     vprintf(format, arglist);
}

void main(void)
{
     dsp21k_msg_func(my_msg_handler);
     dsp21k_msg("Hey! This is my message!\n");
}
```

See Also dsp21k_msg_func

dsp21k_num_dsps

Prototype `int dsp21k_num_dsps(PDSP21K processor);`

Description This function returns the number of processors on the board that this processor is on (including this processor). If the device was opened using an EEPROM Configuration File, this is the number of DSPs from the file. Otherwise, it is the number from the board's EEPROM.

Arguments

processor	Pointer to processor structure

Returns This function returns the number of processors on the board that this processor is on.

Example

```
PDSP21K processor;
PDSP21K dsp;
int i;
int num_dsps;

//open processor 1 on device 0
processor = dsp21k_open_by_id(0, 1);

if(processor)
{
              num_dsps = dsp21k_num_dsps(processor);
       //reset board
       dsp21k_reset_bd(processor);

       //processors need to be configured after
       //a board reset for host access
       //configure all processors on the board
       for(i = 1; i <= num_dsps; i++)
       {
              dsp = dsp21k_open_by_id(0, i);
       if(dsp)
       {
```

```
                dsp21k_cfg_proc(dsp);
                    dsp21k_close(dsp);
            }
        }
        dsp21k_close(processor);
}
```

See Also

- dsp21k_mpid
- dsp21k_dsp_type
- dsp21k_dsp_rev

dsp21k_open

Prototype `PDSP21K dsp21k_open(int device_num)`

Description This function attempts to open the first processor on the BittWare device **`device_num`**. If the device exists and has at least one processor, the function opens the device driver and reads the hardware configuration information from its configuration file or its EEPROM. It creates a DSP21K structure, initializes it, and returns a pointer to it (which is needed to pass to other library routines). This function allocates memory, which is freed when you call *dsp21k_close*. It does not change DSP memory, and it does not disturb the processor. The processor keeps running if previously loaded.

Arguments

`device_num` Device number

Returns The function returns a pointer to a newly created processor structure. It returns NULL if it encounters an error.

Example

```
#include <dsp21k.h>
void main(int argc, char *argv[])
{
PDSP21K processor;

//open first processor on device 0
processor = dsp21k_open(0);
if(processor)
{
       if(dsp21k_load_exe(processor, argv[1]) < 0)
       {
              printf("Error downloading executable
     file, \"%s\"\n", argv[1]);
       }
       else
              dsp21k_start(processor);

       dsp21k_close(processor);
```

```
}
}
```

See Also

- dsp21k_close
- dsp21k_open_all
- dsp21k_open_by_id

dsp21k_open_all

Prototype `long dsp21k_open_all ( void );`

Description This function opens all processors in your system.

Note *The dsp21k_open_all function will also count processors that have already been opened.*

Arguments None

Returns This function returns the number of open DSPs.

Example None

See Also
- dsp21k_close_all
- dsp21k_get_proc
- dsp21k_open
- dsp21k_open_by_id
- dsp21k_open_by_title

dsp21k_open_by_id

Prototype `PDSP21K dsp21k_open_by_id(int device_num, int dsp_id)`

Description This function attempts to open the processor with ID number **dsp_id** on the BittWare device **device_num**. If the device exists and has at least one processor, the function opens the device driver and reads the hardware configuration information from its configuration file or its EEPROM. It creates a DSP21K structure, initializes it, and returns a pointer to it (which is needed to pass to other library routines). This function allocates memory, which is freed when you call *dsp21k_close*. It does not change DSP memory, and it does not disturb the processor. The processor keeps running if previously loaded.

Arguments

device_num	Device number
dsp_id	Processor ID number

Returns The function returns a pointer to a newly created processor structure. It returns NULL if it encounters an error.

Example

```
#include <dsp21k.h>
void main(int argc, char *argv[])
{
PDSP21K processor;

//open processor ID=1 on device 0
processor = dsp21k_open_by_id(0, 1);
if(processor)
{
       if(dsp21k_load_exe(processor, argv[1]) < 0)
       {
              printf("Error downloading executable
     file, \"%s\"\n", argv[1]);
       }
       else
```

```
                dsp21k_start(processor);

            dsp21k_close(processor);
    }
    }
```

See Also

- dsp21k_close
- dsp21k_open_all
- dsp21k_open_by_id

dsp21k_phys_memory_count

Prototype `int dsp21k_phys_memory_count()`

Description This function returns the number of host PC physical memory buffers allocated with previous calls to *dsp21k_alloc_phys_memory*.

Arguments None

Returns This function returns the number of physical memory buffers that have been allocated.

Example

```
//free all previously allocated
//physical memory buffers
for(i = 0; i < dsp21k_phys_memory_count(); i++)
{
       dsp21k_get_phys_memory(i, &phys_mem);
       dsp21k_free_phys_memory(processor, &phys_mem);
}
```

See Also

- dsp21k_alloc_phys_memory
- dsp21k_free_phys_memory
- dsp21k_get_phys_memory

For further information on accessing host physical memory with the HIL, refer to section 2.4.

dsp21k_prn_copyright

Prototype

```
char *dsp21k_prn_copyright(int firstyear, int current-
    year);
```

Description This function returns a pointer to a null-terminated string containing the BittWare copyright text for the given years.

Arguments

firstyear	First year for the copyright notice
currentyear	Current year for the copyright notice

Returns The function returns a pointer to the first character of a null-terminated character string containing the BittWare copyright notice.

Example

```
char * pchar;
pchar = dsp21k_prn_copyright (1992, 2002);
printf( "%s\n", pchar );
```

produces:

```
Copyright © 1992-2002 BittWare, Inc. All rights
    reserved.
```

See Also None

dsp21k_prn_version

Prototype `char *dsp21k_prn_version(float app_ver);`

Description This function returns a pointer to a null-terminated string containing the BittWare DSP21k-SF Toolkit release, library build, and version (**app_ver**) text.

Arguments

app_ver	Version number of the application

Returns The function returns a pointer to the first character of a null-terminated character string containing the BittWare version text.

Example

```
char * pchar;
pchar = dsp21k_prn_version (3.40);
printf( "%s\n", pchar );
```

produces:

```
Release 7.10, [ DSP21k-SF, Jun 26 2000 ],
Version 3.40
```

when linked with release 7.10 of the DSP21k-SF Toolkit.

See Also

- dsp21k_build
- dsp21k_os_name
- dsp21k_version
- dsp21k_prn_info

dsp21k_proc_num

Prototype `int dsp21k_proc_num ( PDSP21K processor );`

Description This function returns a unique number identifying the processor for backward capability. This number is generated using the following formula:

```
proc_num = (device_num * 10) + dsp_id
```

Arguments

processor	Pointer to processor structure

Returns The function returns a unique number identifying the processor.

Example None

See Also
- dsp21k_device_num
- dsp21k_mpid

dsp21k_proc_running

Prototype `int dsp21k_proc_running( PDSP21K processor );`

Description This function returns TRUE or FALSE to indicate whether **`processor`** is running a program or not.

Arguments

`processor`	Pointer to processor structure

Returns If the processor has been loaded and started, the function returns TRUE. If it has not been started, has been reset since being started, or has been closed and reopened since being started, it returns FALSE.

Example None

See Also

- dsp21k_start
- dsp21k_reset_proc
- dsp21k_reset_bd
- dsp21k_load_exe
- dsp21k_cfg_proc

dsp21k_rd_bd_setting

Prototype `int dsp21k_rd_bd_setting( PDSP21K processor, int setting_id, int * value)`

Description This function reads the specified HIL setting and places it in the **value** parameter. The list of valid settings are located in the `const21k.h` header file.

Arguments

processor	Pointer to processor structure
setting_id	See `const21k.h` for valid settings
value	Pointer to an integer to contain the setting value (1 or 0 for bit values)

Returns The function returns 0 if it succeeds (DSP21K_SUCCESS). If it does not succeed, it returns a negative HIL error value.

Example

```
int value;

//toggle burst enable
if(dsp21k_rd_bd_setting(processor,
     FIN_SDRAM_BURST_ENABLE, &value)
{
value = !value;
dsp21k_wr_bd_setting(processor,
     FIN_SDRAM_BURST_ENABLE, &value);
}
```

See Also

- dsp21k_rd_bdreg
- dsp21k_wr_bd_setting

dsp21k_rd_bdreg

Prototype

```
unsigned long dsp21k_rd_bdreg(PDSP21K processor,
     unsigned long offset);
```

Description This function reads the SharcFIN register at the 32-bit offset given by **offset**.

Arguments

processor	Pointer to processor structure
offset	Offset to read from

Returns The value read is an unsigned long.

Example

```
ULONG data_register_offset = 4;
ULONG data;
data = dsp21k_rd_bdreg( processor, data_register_offset
     );
```

See Also

- dsp21k_wr_bdreg
- dsp21k_rd_bd_setting

dsp21k_rd_phys_memory

Prototype

```
int dsp21k_rd_phys_memory( DSP21K_PHYS_MEMORY *
     phys_mem, ULONG offset, ULONG count, LPULONG buf)
```

Description This function reads a buffer of values from a physical memory buffer on the host PC allocated or mapped by a previous call to *dsp21k_alloc_phys_memory*.

Arguments

phys_mem	address of a DSP21K_PHYS_MEMORY structure whose members are: • **phys_addr:** Host physical memory address, accessible by the PCI device • **size:** Size in bytes of physical memory buffer • **mem_ptr:** Virtual memory pointer, accessible by the Host PC
offset	32-bit offset from start of physical memory buffer
count	Number of 32-bit words to transfer
buf	Pointer to buffer to read memory to

Returns The function returns 0 if it succeeds (DSP21K_SUCCESS). If it does not succeed, it returns a negative HIL error value.

Example See *dsp21k_alloc_phys_memory* example.

dsp21k_rd_phys_memory8

Prototype

```
int dsp21k_rd_phys_memory8( DSP21K_PHYS_MEMORY *
     phys_mem, ULONG offset, ULONG count, LPUCHAR buf)
```

Description

This function reads a buffer of values from a physical memory buffer on the host PC allocated or mapped by a previous call to *dsp21k_alloc_phys_memory*.

Arguments

phys_mem	address of a DSP21K_PHYS_MEMORY structure whose members are: • **phys_addr:** Host physical memory address, accessible by the PCI device • **size:** Size in bytes of physical memory buffer • **mem_ptr:** Virtual memory pointer, accessible by the Host PC
offset	8-bit offset from start of physical memory buffer
count	Number of 8-bit words to transfer
buf	Pointer to buffer to read memory to

Returns

The function returns 0 if it succeeds (DSP21K_SUCCESS). If it does not succeed, it returns a negative HIL error value.

Example

See *dsp21k_alloc_phys_memory* example.

dsp21k_reset_bd

Prototype `int dsp21k_reset_bd(PDSP21K processor);`

Description This function performs a board-level (hard) reset on the selected processor. The reset affects all processors on the board. Do not access the affected processors until they are configured. The suggested method of configuring processors depends upon the type of processor, according to the following table. Refer to the hardware documentation for any other effects of a board reset.

Processor Type	HIL Function(s)
ADSP-21xxx	**dsp21k_cfg_proc** or **dsp21k_bc_cfg_proc** if supported
ADSP-TS101	**dsp21k_bc_cfg_proc**
ADSP-TS201	**dsp21k_bc_reset_proc** and **dsp21k_bc_cfg_proc**

Arguments

processor Pointer to processor structure

Returns The function returns 0 if it succeeds (DSP21K_SUCCESS). If it does not succeed, it returns a negative HIL error value. Refer to section 2.2.5 for more information on HIL error values.

Example

```
PDSP21K processor;
PDSP21K dsp;
int i;
int num_dsps;

//open processor 1 on device 0
processor = dsp21k_open_by_id(0, 1);

if(processor)
{
                num_dsps = dsp21k_num_dsps(processor);
        //reset board
```

```
            dsp21k_reset_bd(processor);

            //processors need to be configured after
            //a board reset for host access
            //configure all processors on the board
            for(i = 1; i <= num_dsps; i++)
            {
                   dsp = dsp21k_open_by_id(0, i);
            if(dsp)
            {
                    dsp21k_cfg_proc(dsp);
                           dsp21k_close(dsp);
            }
          }
          dsp21k_close(processor);
}
```

See Also

- dsp21k_reset_proc
- dsp21k_start
- dsp21k_load_exe
- dsp21k_cfg_proc
- dsp21k_proc_running
- dsp21k_bc_cfg_proc
- dsp21k_bc_reset_proc

dsp21k_reset_proc

Prototype `int dsp21k_reset_proc(PDSP21K processor);`

Description This function performs a processor (soft) reset. Use this function to reset a single processor on a multi-processor board. After resetting, the processor must be configured with a call to *dsp21k_cfg_proc* before accessing the processor again with a HIL function.

Arguments

`processor`	Pointer to processor structure

Returns The function returns 0 if it succeeds (DSP21K_SUCCESS). If it does not succeed, it returns a negative HIL error value. Refer to section 2.2.5 for more information on HIL error values.

Example None

See Also

- dsp21k_reset_bd
- dsp21k_start
- dsp21k_load_exe
- dsp21k_cfg_proc
- dsp21k_proc_running

dsp21k_riop

Prototype

```
ULONG dsp21k_riop(PDSP21K processor,
     ULONG offset);
```

Description This function reads the IOP register specified by **`offset`** and returns the unsigned 32-bit value.

Arguments

`processor`	Pointer to processor structure
`offset`	Offset of IOP register

Returns The function returns a value read from the specified IOP register.

Example None

See Also

- dsp21k_wiop

dsp21k_serial_num

Prototype `int dsp21k_serial_num(PDSP21K processor);`

Description This function returns the serial number of the board that this processor is on. If the device was opened using an EEPROM Configuration File, the serial number is from the file. Otherwise, the serial number is from the board's EEPROM.

Arguments

`processor`	Pointer to processor structure

Returns This function returns the serial number of the board that the selected processor is on.

Example None

See Also None

dsp21k_sleep

Prototype

```
void dsp21k_sleep( PDSP21K processor, int milliseconds
    );
```

Description This function delays the calling thread for the specified number of milliseconds. The thread will yield and allow other threads to run while it is sleeping if the operating system supports this.

Arguments

processor	Pointer to processor structure
milliseconds	Number of milliseconds to sleep

Returns None.

Example

```
//pause for 1 second
      dsp21k_sleep(dsp, 1000);
```

dsp21k_start

Prototype `int dsp21k_start(PDSP21K processor);`

Description This function releases the processor from reset. The program will begin executing at the reset vector.

Arguments

`processor`	Pointer to processor structure

Returns The function returns 0 if it succeeds (DSP21K_SUCCESS). If it does not succeed, it returns a negative HIL error value. Refer to section 2.2.5 for more information on HIL error values.

Example

```
#include <dsp21k.h>

void main(int argc, char *argv[])
{
  PDSP21K processor;
  //open first processor on device 0
  processor = dsp21k_open(0);
  if (processor)
  {
    dsp21k_load_exe(processor,argv[1]);
    dsp21k_start(processor);
    dsp21k_close(processor);
  }
}
```

See Also

- dsp21k_reset_bd
- dsp21k_reset_proc
- dsp21k_load_exe
- dsp21k_cfg_proc
- dsp21k_proc_running

dsp21k_symbol_count

Prototype `long dsp21k_symbol_count(PDSP21K processor);`

Description This function returns the number of symbols in the symbol table. If a program is not loaded, the number of symbols is 0. Call *dsp21k_get_next_symbol* in a loop to iterate through the list of symbols from 0 to the number returned from this function.

Arguments

`processor`	Pointer to processor structure

Returns This function returns the number of symbols in the symbol table.

Example See *dsp21k_get_next_symbol* for example.

See Also

- dsp21k_load_exe
- dsp21k_free_labels
- dsp21k_get_next_symbol
- dsp21k_labels_defined
- dsp21k_load_symbols

dsp21k_symbol_size

Prototype `int dsp21k_symbol_size (PDSP21K processor, LPCHAR symbol)`

Description This function returns the size in words of the symbol. To get the word width of a symbol, call the function *dsp21k_symbol_width*.

Arguments

processor	Pointer to processor structure
symbol	Name of global variable

Returns This function returns the size in words of the symbol. If the symbol is not found, this function returns 0.

Example

```
#include <dsp21k.h>
void main(int argc, char *argv[])
{
       PDSP21K processor;
       LPCHAR symbol;
       int symbol_index, symbol_count, symbol_size,
     symbol_width;

       //open first processor on device 0
       processor = dsp21k_open(0);
       if(processor)
       {

              dsp21k_load_exe(processor, argv[1]);

              printf("DSP program %s symbol list\n",
     argv[1]);
              symbol_count =
     dsp21k_symbol_count(processor);
              for(symbol_index = 0; symbol_index <
     symbol_count;
```

```
symbol_index++)
       {
              //get the name of this symbol
              symbol =
     dsp21k_get_next_symbol(processor, symbol_index);

              //get number of words this symbol takes up
              symbol_size =
     dsp21k_symbol_size(processor, symbol);
              //get width in bits of a symbol word
              symbol_width =
     dsp21k_symbol_width(processor,
symbol);

              //print symbol
              printf("\t%s (%d %d-bit words)\n",
     symbol,
symbol_size, symbol_width);
              }
              dsp21k_close(processor);
       }
}
```

See Also

- dsp21k_load_exe
- dsp21k_free_labels
- dsp21k_get_next_symbol
- dsp[21k_labels_defined
- dsp21k_load_symbols
- dsp21k_symbol_count
- dsp21k_symbol_width

dsp21k_symbol_width

Prototype `int dsp21k_symbol_width (PDSP21K processor, LPCHAR symbol)`

Description This function returns the word width in bits of the symbol. To get the word width of a symbol, call the function *dsp21k_symbol_width*.

Arguments

processor	Pointer to processor structure
symbol	Name of global variable

Returns This function returns the word width in bits of the symbol. If the symbol is not found, this function returns 0.

Example

```
#include <dsp21k.h>
void main(int argc, char *argv[])
{
       PDSP21K processor;
       LPCHAR symbol;
       int symbol_index, symbol_count, symbol_size,
     symbol_width;

//open first processor on device 0
processor = dsp21k_open(0);
if(processor)
{

       dsp21k_load_exe(processor, argv[1]);

       printf("DSP program %s symbol list\n", argv[1]);
       symbol_count = dsp21k_symbol_count(processor);
       for(symbol_index = 0; symbol_index <
     symbol_count;
symbol_index++)
       {
```

```
                //get the name of this symbol
                symbol =
     dsp21k_get_next_symbol(processor,
symbol_index);

                //get number of words this symbol takes up
                symbol_size =
     dsp21k_symbol_size(processor,
symbol);
                //get width in bits of a symbol word
                symbol_width =
     dsp21k_symbol_width(processor,
symbol);

                //print symbol
                printf("\t%s (%d %d-bit words)\n",
     symbol,
symbol_size, symbol_width);
       }
       dsp21k_close(processor);
}
}
```

See Also

- dsp21k_load_exe
- dsp21k_free_labels
- dsp21k_get_next_symbol
- dsp21k_labels_defined
- dsp21k_load_symbols
- dsp21k_symbol_count
- dsp21k_symbol_size

dsp21k_target

Prototype `char *dsp21k_target( void );`

Description This function returns a pointer to a null-terminated string containing the target environment for the library and tools.

Arguments None

Returns The function returns a pointer to the first character of a null-terminated character string describing the target environment for the library and tools.

Example None

See Also
- dsp21k_build
- dsp21k_prn_info
- dsp21k_prn_version
- dsp21k_version

dsp21k_ul_8s

Prototype

```
int dsp21k_ul_8s(PDSP21K processor, ULONG dsp_addr,
     ULONG count, LP UCHAR val);
```

Description

From the DSP memory address, *dsp_addr*, this function uploads **count** 8-bit values to the host buffer pointed to by **val**. This function will only read from 8-bit wide memory such as flash.

Arguments

processor	Pointer to processor structure
dsp_addr	DSP memory address
count	Number of 8-bit values to transfer
val	Address of host buffer

Returns

The function returns 0 if it succeeds (DSP21K_SUCCESS). If it does not succeed, it returns a negative HIL error value.

Example

```
UCHAR byte_value;

//read a single byte from flash memory
dsp21k_ul_8s(processor, 0x2800000, 1, &byte_value);
```

See Also

dsp21k_dl_8s

dsp21k_ul_16s

Prototype

```
int dsp21k_ul_16s(PDSP21K processor, ULONG dsp_addr,
     ULONG count, LP USHORT val);
```

Description From the DSP memory address, *dsp_addr*, this function uploads **count** 16-bit values to the host buffer pointed to by **val**. This function will only read from 16-bit wide memory such as the SHARC's short word memory.

Arguments

processor	Pointer to processor structure
dsp_addr	DSP memory address
count	Number of 16-bit values to transfer
val	Address of host buffer

Returns The function returns 0 if it succeeds (DSP21K_SUCCESS). If it does not succeed, it returns a negative HIL error value.

Example

```
USHORT shrt_value;

//read a 16-bit value at the start of short word memory
     on a 21160
dsp21k_ul_16s(processor, 0x80000, 1, &shrt_value);
```

See Also dsp21k_dl_16s

dsp21k_ul_32s

Prototype

```
int dsp21k_ul_32s(PDSP21K processor, unsigned long
     dsp_addr, unsigned int count, void * val);
```

Description

From the DSP's memory, which starts at **dsp_addr**, this function uploads **count** 32-bit values to the host buffer pointed to by **val**. Determine the DSP global variable addresses with *dsp21k_get_addr*.

Arguments

processor	Pointer to processor structure
dsp_addr	DSP memory address
count	Number of 32-bit values to transfer
val	Address of host buffer

Returns

The function returns 0 if it succeeds (DSP21K_SUCCESS). If it does not succeed, it returns a negative HIL error value. Refer to section 2.2.5 for more information on HIL error values.

Example

```
PDSP21K processor;
ULONG dsp addr;
long buffer[2];

processor = dsp21k_open(0);
dsp_addr = 0x30000;

dsp21k_ul_32s(processor, dsp_addr, 2, &buffer[0]);
dsp21k_close(processor);
```

See Also

- dsp21k_dl_32s
- dsp21k_get_addr
- dsp21k_ul_int
- dsp21k_dl_int

dsp21k_ul_48

Prototype

```
int dsp21k_ul_48(PDSP21K processor, unsigned long
     dsp_addr, unsigned short dh, unsigned short dm,
     unsigned short dl);
```

Description

This function uploads a single 48-bit value to **dl**, **dm**, and **dh** from the DSP's memory, starting at **dsp_addr**. Determine DSP global variable addresses with *dsp21k_get_addr*.

Arguments

processor	Pointer to processor structure
dsp_addr	DSP memory address
dh	Upper 16 bits of the 48-bit word
dm	Middle 16 bits of the 48-bit word
dl	Lower 16 bits of the 48-bit word

Returns

The function returns 0 if it succeeds (DSP21K_SUCCESS). If it does not succeed, it returns a negative HIL error value. Refer to section 2.2.5 for more information on HIL error values.

Example

```
PDSP21K processor;
ULONG  dsp_addr;
USHORT buffer[3];

processor = dsp21k_open(0);
dsp_addr = 0x21000;

//upload from 0x21000.
dsp21k_ul_48 (processor, dsp_addr, &buffer[0],
     &buffer[1], &buffer[2]);

dsp21k_close(processor);
```

See Also

- dsp21k_dl_48s
- dsp21k_get_addr
- dsp21k_ul_48s
- dsp21k_dl_48

dsp21k_ul_48s

Prototype

```
void dsp21k_ul_48s (PDSP21K processor, ULONG
    dsp_addr,UINT count, LPUSHORT val);
```

Description

This function uploads **count** 48-bit values, each of which are defined by three USHORTs contained in **val**, from the DSP's memory, starting at **dsp_addr**. Determine DSP global variable addresses with *dsp21k_get_addr*.

Arguments

processor	Pointer to processor structure
dsp_addr	DSP memory address
count	Number of 48-bit values
val	Pointer to a buffer containing **count** * 3 USHORTS

Returns

None

Example

```
PDSP21K processor;
ULONG dsp_addr;
USHORT buffer[3];
processor = dsp21k_open(0);
dsp_addr = 0x41000;
//Upload from 0x41000.
dsp21k_ul_48 (processor, dsp_addr, 1, buffer);
dsp21k_close(processor);
```

See Also

- dsp21k_dl_48
- dsp21k_dl_48s
- dsp21k_get_addr
- dsp21k_ul_48

dsp21k_ul_64s

Prototype

```
int dsp21k_ul_64s(PDSP21K processor, ULONG dsp_addr,
     ULONG count, LP ULONG val);
```

Description From the DSP memory address, *dsp_addr*, this function uploads **count** 64-bit values to the host buffer pointed to by **val**. This function will only read from 64-bit wide memory such as the ADSP-2116x's long word memory.

Arguments

processor	Pointer to processor structure
dsp_addr	DSP memory address
count	Number of 16-bit values to transfer
val	Address of host buffer

Returns The function returns 0 if it succeeds (DSP21K_SUCCESS). If it does not succeed, it returns a negative HIL error value.

Example

```
ULONG val64[2];

//read a 64-bit value at the start of long word memory
     on a 21160
dsp21k_ul_64s(processor, 0x20000, 1, val64);
```

See Also dsp21k_dl_64s

dsp21k_ul_dbl

Prototype

```
double dsp21k_ul_dbl(PDSP21K processor,
     unsigned long dsp_addr);
```

Description This function uploads a single 64-bit float from the DSP's memory, which starts at **dsp_addr**. Determine global variable addresses with *dsp21k_get_addr*.

Arguments

processor	Pointer to processor structure
dsp_addr	DSP memory address

Returns The function returns an uploaded value.

Example None

See Also

- dsp21k_dl_dbl
- dsp21k_get_addr
- dsp21k_ul_32s

dsp21k_ul_flt

Prototype

```
float dsp21k_ul_flt(PDSP21K processor,
     unsigned long dsp_addr);
```

Description This function uploads a single 32-bit float from the DSP's memory, which starts at **dsp_addr**. Determine global variable addresses with *dsp21k_get_addr*.

Arguments

processor	Pointer to processor structure
dsp_addr	DSP memory address

Returns This function returns the uploaded value from the DSP's memory.

Example None

See Also

- dsp21k_dl_flt
- dsp21k_get_addr
- dsp21k_ul_32s

dsp21k_ul_int

Prototype

```
int dsp21k_ul_int(PDSP21K processor, unsigned long
     dsp_addr);
```

Description This function uploads a single 32-bit integer from the DSP's memory, which starts at **`dsp_addr`**. Determine global variable addresses with *dsp21k_get_addr*.

Arguments

`processor`	Pointer to processor structure
`dsp_addr`	DSP memory address

Returns The function returns the uploaded value.

Example None

See Also

- dsp21k_dl_int
- dsp21k_get_addr
- dsp21k_ul_32s

dsp21k_ul_sctn32

Prototype

```
int dsp21k_ul_sctn32 (PDSP21K processor, ULONG
     dsp_addr, UINT count, UINT size, LPUCHAR val)
```

Description From DSP data memory, starting at the DSP address, this function uploads **`count`** program section values, each of **`size`** bytes, to the host buffer pointed to by **`val`**. The program sections must be in Extensible Linker Format (ELF), and the values are byte-packed into **`val`**.

Arguments

`processor`	Pointer to processor structure
`dsp_addr`	DSP data memory address
`count`	Number of values to upload
`size`	Size of each value in bytes
`val`	Address of host buffer

Returns This function returns 0 if it succeeds (DSP21K_SUCCESS). If it does not succeed, it returns a negative HIL error value.

Example None

See Also

- dsp21k_dl_sctn32
- dsp21k_dl_sctn48
- dsp21k_ul_sctn48

dsp21k_ul_sctn48

Prototype

```
int dsp21k_ul_sctn32 (PDSP21K processor, ULONG
     dsp_addr, UINT count, UINT size, LPUCHAR val)
```

Description

From DSP data memory, starting at the DSP address, this function uploads **count** program section values, each of **size** bytes, to the host buffer pointed to by **val**. The program sections must be in Extensible Linker Format (ELF), and the values are byte-packed into **val**.

Arguments

processor	Pointer to processor structure
dsp_addr	DSP program memory address
count	Number of values to upload
size	Size of each value in bytes
val	Address of host buffer

Returns

This function returns 0 if it succeeds (DSP21K_SUCCESS). If it does not succeed, it returns a negative HIL error value.

Example

```
UCHAR instr[6];

//upload instruction at 40005, the 21160 reset vector
dsp21k_ul_sctn48(processor, 0x40005, 1, 6, instr);

//turn instruction into nop;
instr[0] = 0;
instr[1] = 0;

//download nop; instruction to 0x40005
dsp21k_dl_sctn48(processor, 0x40005, 1, 6, instr);
```

See Also

- dsp21k_dl_sctn32
- dsp21k_dl_sctn48
- dsp21k_ul_sctn32

dsp21k_ul_xflt

Prototype

```
double dsp21k_ul_xflt(PDSP21K processor,
     unsigned long dsp_addr);
```

Description This function uploads a single 40-bit float from the DSP's memory, which is given by **`dsp_addr`**. Determine global variable addresses with *dsp21k_get_addr*.

Arguments

`processor`	Pointer to processor structure
`dsp_addr`	DSP memory address

Returns The function returns the uploaded 40-bit float.

Example None

See Also

- dsp21k_dl_xflt
- dsp21k_get_addr

dsp21k_usleep

Prototype

```
void dsp21k_usleep( PDSP21K processor, int microseconds
    );
```

Description

This function delays the calling thread for the specified number of microseconds, or the minimum number of microseconds that the operating system can delay for, if it is larger. The thread will yield and allow other threads to run while it is sleeping if the operating system supports this.

Arguments

processor	Pointer to processor structure
milliseconds	DSP memory address

Returns

None.

Example

```
//pause for 6 microseconds
       dsp21k_usleep(dsp, 6);
```

dsp21k_version

Prototype `float dsp21k_version(void);`

Description This function returns the version (release) number of the library as a float, with two digits to the right of the decimal point. The format is as follows: "MM.NB" where "MM" is the major version number, "N" is the minor version number, and "B" is the bug-fix number.

Arguments None

Returns The function returns the DSP21k library version (release) number.

Example
```
printf("Release number = %4.2F\n", dsp21k_version() );
```

See Also
- dsp21k_build
- dsp21k_prn_info
- dsp21k_prn_version

dsp21k_wiop

Prototype

```
void dsp21k_wiop(PDSP21K processor, ULONG offset, ULONG
     value);
```

Description This function writes **value** to the IOP register that **offset** specifies. The **offset** argument must be a valid IOP register offset.

Arguments

processor	Pointer to processor structure
offset	Address of IOP register
value	32-bit value to be written

Returns None

Example None

See Also dsp21k_riop

dsp21k_wr_bd_setting

Prototype `int dsp21k_wr_bd_setting( PDSP21K processor, int setting_id, int * value)`

Description This function writes the specified HIL setting with the value parameter. The list of valid settings are located in the `const21k.h` header file.

Arguments

`processor`	Pointer to processor structure
`setting_id`	See `const21k.h` for valid settings
`value`	Pointer to an integer containing the desired setting value (1 or 0 for bit values)

Returns The function returns 0 if it succeeds (DSP21K_SUCCESS). If it does not succeed, it returns a negative HIL error value.

Example

```
int value;

//toggle burst enable
if(dsp21k_rd_bd_setting(processor,
     FIN_SDRAM_BURST_ENABLE, &value)
{
       value = !value;
       dsp21k_wr_bd_setting(processor,
     FIN_SDRAM_BURST_ENABLE, &value);
}
```

See Also

- dsp21k_wr_bdreg
- dsp21k_rd_bd_setting

dsp21k_wr_bdreg

Prototype

```
void dsp21k_wr_bdreg(PDSP21K processor, unsigned long
     offset, unsigned long value);
```

Description This function writes a SharcFIN register with **`value`** at the 32-bit offset given by **`offset`**.

Arguments

`processor`	Pointer to processor structure
`offset`	Offset to read from
`value`	Data to be written

Returns None

Example None

See Also

- dsp21k_rd_bdreg
- dsp21k_wr_bd_setting

dsp21k_wr_phys_memory

Prototype

```
int dsp21k_wr_phys_memory( DSP21K_PHYS_MEMORY *
     phys_mem, ULONG offset, ULONG count, LPULONG buf)
```

Description This function writes a buffer of values to a physical memory buffer on the host PC allocated or mapped by a previous call to *dsp21k_alloc_phys_memory*.

Arguments

phys_mem	address of a DSP21K_PHYS_MEMORY structure whose members are: • **phys_addr:** Host physical memory address, accessible by the PCI device • **size:** Size in bytes of physical memory buffer • **mem_ptr:** Virtual memory pointer, accessible by the Host PC
offset	32-bit offset from start of physical memory buffer
count	Number of 32-bit words to transfer
buf	Pointer to buffer to read memory to

Returns The function returns 0 if it succeeds (DSP21K_SUCCESS). If it does not succeed, it returns a negative HIL error value.

Example See *dsp21k_alloc_phys_memory* example.

dsp21k_wr_phys_memory8

Prototype

```
int dsp21k_wr_phys_memory8( DSP21K_PHYS_MEMORY *
     phys_mem, ULONG offset, ULONG count, LPUCHAR buf)
```

Description This function writes a buffer of values to a physical memory buffer on the host PC allocated or mapped by a previous call to *dsp21k_alloc_phys_memory*.

Arguments

phys_mem	address of a DSP21K_PHYS_MEMORY structure whose members are: • **phys_addr:** Host physical memory address, accessible by the PCI device • **size:** Size in bytes of physical memory buffer • **mem_ptr:** Virtual memory pointer, accessible by the Host PC
offset	8-bit offset from start of physical memory buffer
count	Number of 8-bit words to transfer
buf	Pointer to buffer to read memory to

Returns The function returns 0 if it succeeds (DSP21K_SUCCESS). If it does not succeed, it returns a negative HIL error value.

Example See *dsp21k_alloc_phys_memory* example.

dsp21k_xflt_to_dbl

Prototype

```
double dsp21k_xflt_to_dbl(float f, unsigned char
    xbyte);
```

Description This function returns a 64-bit double converted from a 40-bit extended floating point number.

Arguments

f	Most significant 32 bits of extended floating point number
xbyte	Least significant 8 bits of extended float point number

Returns The function returns a 64-bit double converted from an extended 40-bit float.

Example See also *dsp21k_dbl_to_xflt*

Chapter 4
Redistributing/Installing a HIL-based Application

This chapter gives an overview of the steps involved in installing and redistributing a Host Interface Library (HIL) application on both Linux and Windows operating systems.

4.1 Redistribution/Installation in Linux

4.1.1 Linux System Requirements

- To redistribute or install the HIL on a Linux machine, the requirements are:
- the kernel version of target machine must match that of the source machine.
- the target machine must have all dependent libraries installed.

4.1.2 Installing the HIL in Linux

Copy system files

To install the HIL in a Linux environment, copy these files:

- `/etc/rc.d/init.d/bittware`
- `/lib/modules/misc/windrvr6.o`
- `/lib/modules/misc/bwfin_module.o`
- `/usr/lib/libadlibelf.so`
- `/usr/lib/libhil.so.7.4.0`
- `/usr/lib/libhil.so`
- `/usr/local/dsp21ksf/bin/bwcfgm`
- `/usr/local/dsp21ksf/bin/wdreg`

Starting the HIL

After copying the above files, perform the following:

```
# install shared libs
cd /usr/lib;ldconfig

# install BittWare service
chmod 755 /etc/rc.d/init.d/bittware
/sbin/chkconfig --add bittware

# start service - this will get started at boot time
/etc/rc.d/init.d/bittware start
```

4.2 Redistribution/Installation in Windows

4.2.1 Windows System Requirements

To redistribute or install the HIL on a Windows machine, the target must have MFC 6.2 Runtime libraries installed, and the version must be Windows 98 or greater.

4.2.2 Installing the HIL in Windows

Copy system files

To install the HIL in a Windows environment, copy these files:

- `<WINDIR>\system32\drivers\bwfin.sys`
- `<WINDIR>\system32\drivers\windrvr6.sys`
- `<WINDIR>\inf\sharcfin.inf`
- `<WINDIR>\inf\windrvr6.inf`
- `c:\dsp21ksf\bin\bwcfg.exe`
- `c:\dsp21ksf\bin\adlibelf.dll`
- `c:\dsp21ksf\bin\hil.dll`
- `c:\dsp21ksf\drivers\wdreg16.exe`
- `c:\dsp21ksf\drivers\wdreg_gui.exe`
- `c:\dsp21ksf\drivers\wdreg.exe`

Starting the HIL:

1. Delete any existing BittWare hardware registry keys (or remove existing BittWare hardware using the device manager):

   Windows9x:
   `HKEY_LOCAL_MACHINE\ENUM\PCI\VEN_12BA*`
   or
   WindowsNT:
   `HKEY_LOCAL_MACHINE\CURRENTCONTROLSET\ENUM\PCI\VEN_12BA*`

2. Delete `c:\dsp21ksf\bin\bittware.dif` if it exists
3. Add "`c:\dsp21ksf\bin`" to PATH
4. Set DSP21KSF environment variable to `c:\dsp21ksf`

5. Set registry key as follows for WinNT-based:

   ```
   HKEY_LOCAL_MACHINE\System\CurrentControlSet\
    Control\Session Manager\Memory Management\
    SystemPages
   ```
   to 65536

6. Register the driver as follows:

   • **If Win98/ME-based:**

   `c:\dsp21ksf\drivers\wdreg16.exe` -silent -name bwfin uninstall

   `c:\dsp21ksf\drivers\wdreg16.exe` -silent -inf <WINDIR>/inf/windrvr6.inf install

   `c:\dsp21ksf\drivers\wdreg16.exe` -silent -name bwfin install

   • **If WinNT 4-based:**

   `c:\dsp21ksf\drivers\wdreg.exe` -silent -name bwfin uninstall

   `c:\dsp21ksf\drivers\wdreg.exe` -silent -install

   `c:\dsp21ksf\drivers\wdreg.exe` -silent -name bwfin install

   • **If Win2000/XP-based:**

   `c:\dsp21ksf\drivers\wdreg_gui.exe` -silent -name bwfin uninstall

   `c:\dsp21ksf\drivers\wdreg_gui.exe` -silent -inf <WINDIR>/inf/ windrvr6.inf install

   `c:\dsp21ksf\drivers\wdreg_gui.exe` -silent -name bwfin install

7. Add "`c:\dsp21ksf\bin\bwcfg.exe -build`" shortcut to Startup folder in Start Menu. Note that this shortcut must run before any other application that uses the Host Interface Library.

Index

A

arguments, HIL
- address_param 84
- app_ver 130
- bank 41, 44
- count 62—64, 69, 73, 74, 160, 161
- currentyear 129
- d 59
- dh 66, 153
- dl 66, 153
- dm 66, 153
- dsp_addr 62—64, 69, 73, 74, 160, 161
- firstyear 129
- irq_num 103
- num 82
- processor 20, 57—167
- size_param 84
- value 165, 167
- xbyte 59, 170

B

board name, returning pointer to 52
board type
- integer value of, returning 53
- name of, returning 90

board-level reset 137

C

configuring processor 55
copyright text, pointer to 129

D

Delphi 19
depth of external memory bank 41
device number, returning the 60
directories
- HIL
  - example program location 20
  - header file location 19
  - library location 19
- Windows DLL location 19

DLL, Windows 19
double, 64-bit
- converting to 40-bit extended float 59
- returning from 40-bit extended float 170

downloading
- 16-bit values to DSP memory 63
- 32-bit integer to DSP memory 72
- 32-bit values to DSP memory 46, 64
- 48-bit values to DSP memory 66, 68
- 64-bit values to DSP memory 69
- 8-bit values to DSP memory 62

DSP memory
- uploading
  - floats from 157, 158, 162
  - integers from 159
  - values from 152, 153
- writing
  - floats to 70, 71, 75

DSP. See *Processor*
dsp21k 43, 62, 63, 69, 73, 74, 156, 160, 161
DSP21k Toolkit
- additional documentation for 12, 14
- version number of, finding 164

E

error messages, HIL 82
error value, global 94
example program for HIL 20
executing program with HIL 143
external memory bank. see *memory*
external memory. See *memory*

F

file, last loaded onto processor 116

floats
converting from double 59
uploading from DSP memory
32-bit 158
40-bit 162
64-bit 157
writing to DSP memory 75
32-bit 71
64-bit 70
functions
Host Interface Library 37—170
board control functions 32
data transfer functions 32
DSP errors and messages functions 34
DSP information functions 33
processor control functions 34
program control functions 35

G

global variables
freeing memory for storage of 85

H

hard reset 137
header file (dsp21k.h) 19
HIL. see *Host Interface Library*
Host Interface Library 19
board types supported by 19
example program 20—21
functions 37—170
dsp21k_alloc_phys_memory 38
dsp21k_bank_depth 41
dsp21k_bank_size 42
dsp21k_bank_width 44
dsp21k_bc_cfg_proc 45
dsp21k_bc_dl_32s 46
dsp21k_bc_reset_proc 47
dsp21k_bc_start 48
dsp21k_bc_wiop 49
dsp21k_board_name 52
dsp21k_board_type 53
dsp21k_build 54
dsp21k_cfg_proc 20, 55
dsp21k_close 20, 57
dsp21k_close_all 58
dsp21k_dbl_to_xflt 59
dsp21k_device_num 60
dsp21k_dl_32s 64
dsp21k_dl_48 66
dsp21k_dl_48s 68
dsp21k_dl_dbl 69
dsp21k_dl_exe 20
dsp21k_dl_flt 71
dsp21k_dsp_name 79
dsp21k_dsp_rev 80
dsp21k_dsp_type 81
dsp21k_err_msgs 82
dsp21k_fast_extmem_xfers 83
dsp21k_free_labels 85
dsp21k_free_phys_memory 86
dsp21k_get_addr 20, 88
dsp21k_get_board_name 90
dsp21k_get_device_info 91
dsp21k_get_dsp_name 92
dsp21k_get_last_error 94
dsp21k_get_local_ptrs 95
dsp21k_get_next_symbol 95
dsp21k_get_pcirq 97
dsp21k_get_phys_memory 98
dsp21k_get_proc 99
dsp21k_get_symbol 100
dsp21k_int_disable 101
dsp21k_int_dsp 103
dsp21k_int_enable 104
dsp21k_is_bc_capable 107
dsp21k_is_dma_capable 110
dsp21k_is_dma_complete 111
dsp21k_labels_defined 112
dsp21k_load_exe 113
dsp21k_load_symbols 115
dsp21k_loaded_file 116
dsp21k_mem_width 117
dsp21k_mpid 118
dsp21k_msg 119
dsp21k_msg_func 120
dsp21k_num_dsps 121
dsp21k_open 123
dsp21k_open_all 125

dsp21k_open_by_id 126
dsp21k_phys_memory_count 128
dsp21k_prn_copyright 129
dsp21k_prn_version 130
dsp21k_proc_num 131
dsp21k_proc_running 132
dsp21k_rd_bd_setting 133
dsp21k_rd_bdreg 134
dsp21k_reset_bd 20, 135
dsp21k_reset_proc 139
dsp21k_riop 140
dsp21k_serial_num 141
dsp21k_sleep 142
dsp21k_start 20, 143
dsp21k_symbol_count 113, 144
dsp21k_symbol_size 145
dsp21k_symbol_width 147
dsp21k_target 149
dsp21k_ul_16s 151
dsp21k_ul_32s 152
dsp21k_ul_48 153
dsp21k_ul_48s 155
dsp21k_ul_8s 150
dsp21k_ul_dbl 157
dsp21k_ul_flt 158
dsp21k_ul_int 20, 21, 159
dsp21k_ul_ints 21
dsp21k_ul_xflt 162
dsp21k_usleep 163
dsp21k_version 164
dsp21k_wiop 165
dsp21k_wr_bd_setting 166
dsp21k_wr_bd_settings 166
dsp21k_wr_bdreg 167
dsp21k_wr_phys_memory 168
dsp21k_wr_phys_memory8 169
dsp21k_xflt_to_dbl 170
functions, groups of 32
board control functions 32
data transfer functions 32
DSP errors and messages functions 34
DSP information functions 33
processor control functions 34
program control functions 35
header file, location of 19
libraries, location of 19
message handler
output messages, managing 119
host physical memory, accessing 27—28

I

installation location, returning the 61
integers
downloading to DSP memory 72
uploading from DSP memory 159
internal memory. See *memory*
interrupt thread
destroying 101
interrupts
disabling 101
enabling 104
generating 103
PCI, handling 29—31
returning current DSP-to-PC IRQ 97
waiting 107
IOP registers
reading the 140
writing values to 49, 165

L

labels, defined 112
Labview 19
library, HIL
identifying build of 54
location of files 19
Linux
installation of HIL application 172
loading symbols into PC memory 115

M

memory
DSP. See *DSP memory*
external
access to, speeding up 83
depth of 41
size of 42
starting address of 43
width of 44

internal
depth of 50
starting address of 51
physical
accessing host physical memory 27—28
getting number of allocated buffers 98
messages
output (HIL), managing 119
multi-processor ID 118

P

PCI base address registers
getting info on 91
PDSP21K 20
pointer
PDSP21K 20
to file last loaded 116
to HIL target environment 149
to name of board type 52
processor
accessing a 99
closing a 57
closing all open 58
configuring 55
DSP revision number, getting the 80
identifying board name for 52
identifying board type for 53
multiprocessor ID of 118
processor reset. see *soft reset*
processor type
integer value, returning 81
name of, returning 79, 92
reset, releasing from 48, 143
reset, soft 47
resetting the 139
processor argument. see *arguments, HIL*
programs
32-bit
Delphi 19
Labview 19
Visual Basic 19
example program
HIL 20

R

reading
IOP registers 140
reset, releasing processor from 48, 143
resetting
selected board 137, 139
runprime.c 20—21

S

size
of external memory banks 42
soft reset 47, 139
symbol
getting a 100
returning DSP address of 88
returning next 95
symbol table
loading into PC memory 115

T

target environment for HIL, pointer to 149

U

uploading
floats from DSP memory
32-bit 158
40-bit 162
64-bit 157
integers from DSP memory
16-bit 159
values from DSP memory
32-bit 152
48-bit 153

V

values
downloading to DSP memory
16-bit 63
32-bit 46, 64
48-bit 66, 68
64-bit 69
8-bit 62
uploading from DSP memory
32-bit 152

48-bit 153
writing to IOP register 165
version number, DSP21k Toolkit 164
version text 130
Visual Basic 19

W

width
of external memory banks 44
Windows
installation of HIL application 173
Windows DLL 19
writing
32-bit integer to DSP memory 72
floats to DSP memory 70, 75
32-bit 71
program section values to DSP memory 73—74
values to IOP register 165

This page intentionally left blank.