Dakar
F5 Carrier Board
User Guide

Document Number 500-00354

Revision 2.11

October 1998

Part Number 500-00354
Revision 2.11

Customer Feedback

At Spectrum, we recognize that product documentation that is both accurate and easy to use is important in aiding you in your new product development. We appreciate hearing your comments on how our product's documentation could be improved.

If you wish to comment on any Spectrum documentation then please fax or e-mail a completed copy of this page to us.

Full Name of Document: _____

Document Number: _____ Version Number: _____

If you have found a technical inaccuracy please describe it here:

_____

_____

_____

_____

_____

If you particularly liked or disliked an aspect of the manual then please describe it here:

_____

_____

_____

_____

_____

It may be helpful for us to call you to discuss your comments. If this would be acceptable please provide the following details:

Name: _____ Telephone #: _____

Organization: _____

Thank you for your time,

Spectrum Signal Processing Documentation Group

Fax: (604) 421-1764
Email: documentation@spectrumsignal.com

Contacting Spectrum…

Spectrum's team of dedicated Applications Engineers are available to provide technical support to you for this product. Our office hours are Monday to Friday, 8:00 AM to 5:00 PM, Pacific Standard Time.

Telephone 1-800-663-8986 or (604) 421-5422

Fax (604) 421-1764

Email support@spectrumsignal.com

Internet http://www.spectrumsignal.com

When you contact us, please have the following information on hand:

- A concise description of the problem
- The name of all Spectrum hardware components
- The name and version number of all Spectrum software components
- The minimum amount of code that demonstrates the problem
- The version number of all software packages, including compilers and operating systems

Preface

Spectrum Signal Processing offers a complete line of DSP hardware, software and I/O products for the DSP Systems market based on the latest DSP microprocessors, bus interface standards, I/O standards and software development environments. By delivering quality products, and DSP expertise tailored to specific application requirements, Spectrum can consistently exceed the expectations of our customers. We pride ourselves in providing unrivaled pre and post sales support from our team of application engineers. Spectrum has excellent relationships with third party vendors which allows us to provide our customers with a more diverse and top quality product offering.

Spectrum achieved ISO 9001 quality certification in 1994.

As Spectrum's hardware products are static sensitive, please take precautions when handling and make sure they are protected against static discharge.

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1. Purpose of This Manual

This manual describes the F5 Software Development Kit (SDK) that's used to develop applications for the F5 Carrier Board. It describes how to install the SDK and how to use the F5 Host and C4x DSP libraries.

A second manual, the *Dakar F5 Carrier Board Technical Reference*, describes the features, architecture, and specifications of the F5 Carrier Board, and is the primary hardware reference.

1.2. Reference Documents

This guide is meant to be used in conjunction with the following documents:

- *Dakar F5 Carrier Board Technical Reference* from Spectrum
- *Toolbox Configuration Utilities User Guide* available from Spectrum
- *TMS320C4x User's Guide* available from Texas Instruments
- *TMS320C4x C Source Debugger User's Guide* available from Texas Instruments
- *TMS320 Floating-Point DSP Assembly Language Tools User's Guide* available from Texas Instruments
- *TMS320 Floating-Point DSP Optimizing C Compiler User's Guide* available from Texas Instruments

1.3. Conventions Used in This Manual

This manual uses the following conventions:

- *Italic* font is used to designate placeholder names, such as command parameter names, cross-references, and references to other documents. For example:

  The value passed to *phResc* must be a valid pointer to an F5 resource.

- **Bold** font is used to emphasize text, filenames, and command names within paragraphs. For example:

  Refer to the **readme.txt** file for the most current information.

- `This font` is used to designate program code, examples, text that appears on the screen, and commands that you must enter in an interactive display. For example:

  In the Run dialog box enter:

  ```
  a:\install.exe
  ```

- *<F5RootDirectory>* is used to indicate the location of your F5 SDK files on your hard drive. It specifies the directory on your hard drive from which your F5 SDK files branch, and is most likely the directory you chose during installation.

- "0x" before a number indicates that this is hexadecimal notation (base 16). For example:

  Set the Reload Configuration Registers Bit (bit 29) in the PCI9060/9080 EEPROM control register (PCI offset 0x6C, local bus offset 0xEC) to "0".

- An "h" after a number indicates that this is hexadecimal notation (base 16). For example:

  ```
  IRAM0:  origin = 002FF800h   length = 0400h  /* Internal RAM
  ```

- Wherever possible, the following side-by-side shaded boxes have been used to indicate information that is different for a Windows 95 or Windows NT environment. If you are using a Windows NT environment, for example, you can disregard the information in the **Windows 95** column throughout the manual. For example:

  The F5 Software Development Kit (SDK) consists of the following:

Windows 95:	**Windows NT:**
F5 Windows 95 Host Application Library (ALIB_W95)	F5 Windows NT Host Application Library (ALIB_NT)

2 Hardware Overview

The F5 Carrier Board is a TIM-40 carrier board for use within computers equipped with a PCI bus. The full-length PCI board features one embedded Texas Instruments TMS320C44 Digital Signal Processor (DSP) and three TIM-40 module sites. TIM-40 modules define a family of DSP modules, based on TMS320C4x processors, designed for use in multiple DSP systems. Spectrum offers a range of single and dual processor TIM-40 modules that can be used with the F5 Carrier Board.

2.1. Features of the F5 Carrier Board

The F5 Carrier Board offers the following features:

- Flexible architecture consisting of up to seven TMS320C4x-based processor nodes
- One embedded TMS320C44 (’C44) processor with associated memory devices to form a virtual TIM-40 site for Node A
- Three TIM-40 sites supporting single or double width TIM-40 modules
- Up to 420 MFLOPS performance, by using three dual-processor TIM-40 modules and the embedded ’C44 for a total of seven 50/60 MHz TMS320C4x processors
- Internal TMS320C4x communication port (COMM Port) connections between processor nodes
- Ten external COMM Port connectors from the processor nodes
- 132 MBytes/s peak transfer rates from 32-bit PCI (Master/Slave) Bus
- One on-board DSP~LINK3 module site
- One external DSP~LINK3 connector
- Non-intrusive multi-processor debugging in real-time using JTAG interface
- External connector on end-plate which provides access to TIM-40 Application Specific Pins for nodes B, C, and D
- Up to 512k x 32-bit 1 wait-state SRAM shared between all processor nodes
- Up to 1M x 32-bit 0 wait-state SRAM for the Node A embedded processor
- Programmable Erasable ROM (PEROM) for the Node A embedded processor which provides TIM-40 compliant code boot-strapping and board identification

2.1.1. Scaleable Modular Architecture

With its embedded ’C44 processor and its three module sites that are compatible with Texas Instruments’ TIM-40 specification, the F5 Carrier Board can be configured with one to seven 'C4x DSPs. This results in an architecture that can be scaled from 60 to 420 MFLOPS of processing power with varying memory capacity. A wide range of modules are available from Spectrum providing the flexibility of different memory and analog I/O combinations for each TIM-40 site. Single or double width TIM-40 modules can be accommodated.

2.1.2. PCI Interface

Any PCI master on the PCI bus can access the following devices on the F5 Carrier Board via the PCI slave interface:

- Test Bus Controller
- PLX PCI9060/9080 PCI interface chip registers
- Far Global SRAM (for shared memory communication)
- Interrupt registers
- Bus Control registers

The Node A embedded 'C44 can master the PCI bus through the DMA channels of the PLX PCI9060/9080 PCI interface chip. This allows the F5 Carrier Board to access the resources on the host computer through DMA transfers initiated by Node A.

Note: The Intel® 430FX chipset does not support DMA Bus Mastering. If your computer is equipped with this chipset, the F5 Carrier Board cannot initiate a DMA transfer to the PCI bus as a PCI master.

2.1.3. TMS320C4x Communication Port Interfaces

The TMS320C4x communication ports (COMM Ports) are used for inter-processor communication both internally and externally to the F5 Carrier Board. The COMM Ports provide bi-directional asynchronous communication between TMS320C4x processors. They are ideal for passing large data sets between processors without loading any shared resource, such as the Far Global SRAM. All four nodes are connected to each other internally and there are 10 external COMM Port connections shared amongst the nodes.

Intel is a registered trademark of Intel Corporation.

2.1.4. Shared Memory Resources

The shared bus architecture allows the on-board DSP, the TIM-40 module sites, and the PCI interface to access a shared bank of on-board SRAM. The F5 Carrier Board will have either 0.5 MB or 2 MB of SRAM.

2.1.5. Node A Memory Resources

Node A memory resources consist of a local bus PEROM, local bus SRAM and global bus SRAM in accordance to the TIM-40 specification. These resources are only accessible from the Node A 'C4x. The PEROM device is a 32K x 8-bit device used for TIM-40 IDROM and, optionally, for the loading of boot-code. The local and global busses of Node A contain either both 0.5 MB or both 2 MB zero wait-state 15 ns SRAM banks.

2.1.6. DSP~LINK3 Interface

DSP~LINK3 modules can be used with the F5 Carrier Board via the DSP~LINK3 interface. The DSP~LINK3 module site allows Spectrum DSP~LINK3 modules to be installed right on the F5 Carrier Board while the DSP~LINK3 ribbon cable connector allows external DSP~LINK3 modules to be connected to the F5 Carrier Board. The interface is directly accessible to all processors nodes.

Spectrum offers a range of modules with DSP~LINK3 interfaces that can be used with the F5 Carrier Board, including IndustryPack® modules.

The DSP~LINK3 interface is an open standard for Spectrum's I/O interface. It defines a 32-bit wide, 40 Mbyte-per-second I/O interface with low interrupt latency. The full DSP~LINK3 specification is available from Spectrum upon request. DSP~LINK3 is electrically compatible the DSP~LINK2 specification and may be used with DSP~LINK2 with the appropriate mechanical adapter available from Spectrum.

2.1.7. JTAG Interface Debug Support

An on-board JTAG Test Bus Controller (TBC) is mapped to the PCI Local Bus to provide multiprocessor, C source, debug capability in conjunction with support software applications. JTAG-based debugging uses the 'C4x's dedicated debug port to minimize the intrusiveness of the debugger on your application.

2.2. Memory Configurations Available

The F5 Carrier Board is available in the memory configurations shown in the following table. Each SRAM bank consists of four 8-bit SRAM memory devices.

Table 1 Memory Device Sizes (in 32-bit words)

Total SRAM (Bytes)	Local SRAM Bank 1	Local SRAM Bank 0	Global SRAM Bank 1	Global SRAM Bank 0	Far Global SRAM
	15 ns	*15 ns*	*15 ns*	*15 ns*	*20 ns*
1.5 Mbytes	empty	128K	empty	128K	128K
6 Mbytes	empty	512K	empty	512K	512K
10 Mbytes	512K	512K	512K	512K	512K

2.3. Bus Architecture

Several different communication buses are used on the F5 Carrier Board to connect the embedded ’C44 processor, TIM-40 sites, memory devices, and interface circuitry as shown in the following diagram.

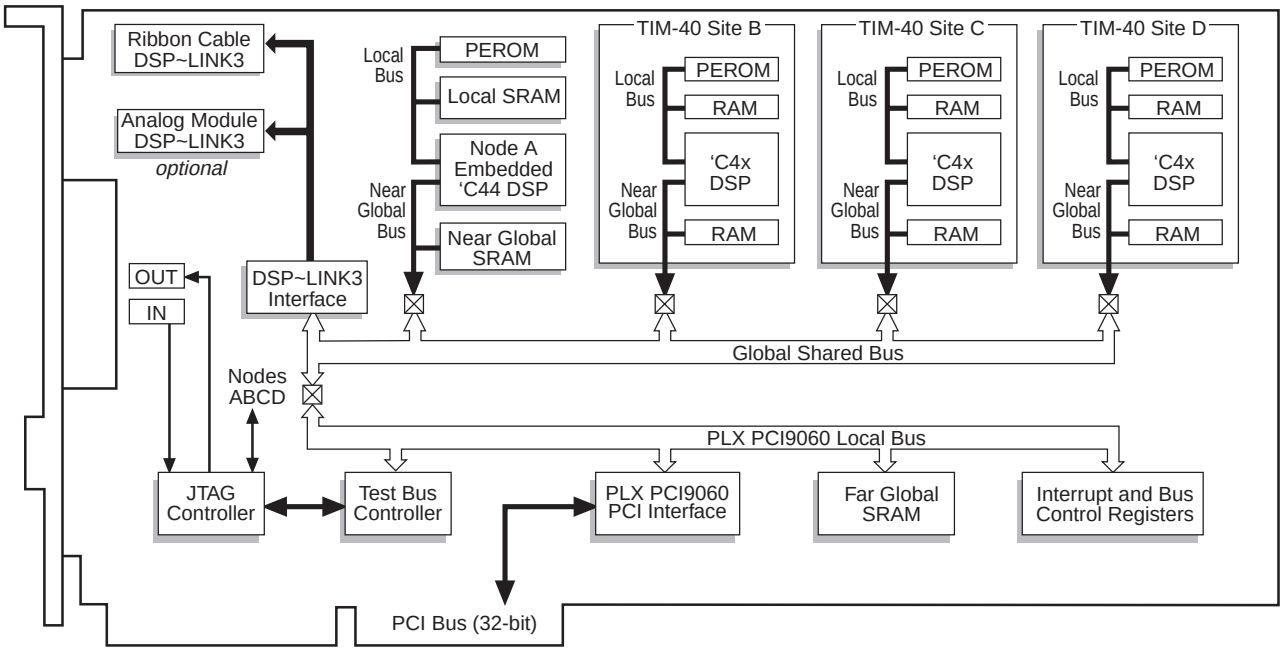


Figure 1 F5 Block Diagram

The internal busses of the F5 Carrier Board are described in the following passage.

Local Bus

The Local Bus address range is specific to a single ’C4x DSP, and is therefore not shared with other processors or nodes. It is a private memory bus of a particular’C4x.

Near Global Bus

The Near Global Bus refers to the Global Bus of each TIM-40 site and the embedded ’C4x node. The SRAM located on ’C44 Global Bus is zero wait state. The Near Global Bus SRAM of a node cannot be accessed by the DSPs of any other nodes.

Global Shared Bus

The Global Shared Bus interconnects the:

- Buffered Global Buses of each TIM-40 site via the Global Connectors
- Buffered Global Buses of the embedded ’C44 node A
- DSP~LINK3 Interface
- Interface between the PLX PCI9060/9080 Local Bus and the Global Shared Bus

32-bit buffers isolate the Global Shared Bus from the ’C4x node Global Buses. An analog quickswitch connects the data and address lines of the Interface between the PLX PCI9060/9080 Local Bus and the Global Shared Bus; the interface control lines are buffered.

PLX PCI9060/9080 Local Bus

The PLX PCI9060/9080 Local Bus of the PLX PCI9060/9080 chip is connected to the

- Global Shared Bus interface buffer
- Far Global SRAM
- Interrupt Controller
- Registers for the Interrupt Controller and Bus Arbitration
- JTAG Controller

2.4. ’C4x Communication Port Architecture

Routing of the ’C4x COMM ports to the 10 external COMM port connectors is shown in the following diagram. Arrows indicate the default port direction after the board is initialized.

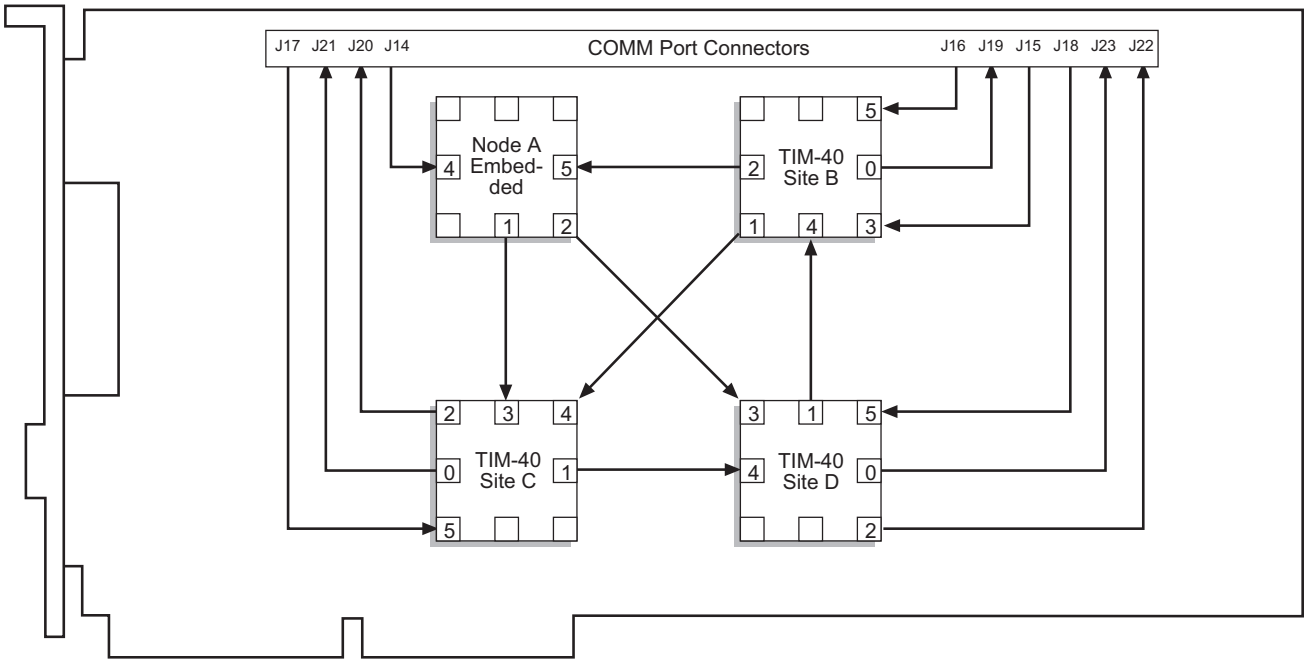


Figure 2 F5 COMM Port Architecture

2.5. Resetting the F5 Carrier Board

The F5 Carrier Board is reset upon system power up. Resets can also be performed from host software on the PCI bus and from JTAG. The following table shows which F5 Carrier Board hardware is initialized by each type of reset.

Table 2 Reset Summary

F5 Carrier Board Hardware Reset	Power Up	PCI Software	JTAG
All CPU Nodes	✓	✓	✓
DSP~LINK3 Slave Devices	✓	✓	
JTAG devices connected to the JTAG Out connector /GRESET pin (J11 pin 20)	✓	✓	✓
PCI bus Interface Logic	✓		
PCI9060/9080 Local Bus Logic	✓	✓	
Support Logic	✓	✓	✓

When the processor nodes are reset with any of the above methods, the ‘C4x processor IIOF lines are tristated as per the TIM-40 specification to allow the boot mode selection jumpers to be read by the C4X processor. To enable its IIOF signals, a ‘C4x must then execute an IACK cycle.

2.5.1. Power Up Reset

The PCI bus RST# signal resets all F5 Carrier Board hardware when it is asserted. It is asserted upon system power up and typically when the operating system is started.

The system must issue PCI bus configuration cycles after RST# to configure the F5 Carrier Board PCI bus interface. Usually the host BIOS or operating system will do this automatically.

The F5 Carrier Board remains in a reset state for approximately 30 milliseconds after power on for on board initialization. Once this is complete, the on board reset logic is released.

2.5.2. PCI Software Reset

Any PCI master (usually the host) can reset all F5 Carrier Board hardware *except* the PCI bus interface logic, using the following procedure on the PLX PCI9060/9080 PCI interface chip.

1. Set the PCI Adapter Software Reset Bit (bit 30) in the PCI9060/9080 EEPROM control register (PCI offset 0x6C, local bus offset 0xEC) to “1”. This places the F5 Carrier Board in the reset state.
2. Set this same bit to “0”. This releases the F5 Carrier Board from reset.
3. Set the Reload Configuration Registers Bit (bit 29) in the PCI9060/9080 EEPROM control register (PCI offset 0x6C, local bus offset 0xEC) to “0”.
4. Set this same bit to “1”. This reloads the PCI9060/9080 configuration registers from EEPROM. The transition from 0 to 1 causes the reload to occur.

PCI bus configuration registers in the PCI9060/9080 are *not* affected by this reset, therefore PCI bus configuration cycles do not need to be issued.

After reset the PCI9060/9080 EEPROM needs to be reloaded as described above. The PCI9060/9080 EEPROM enable jumper J25 must be installed, and the EEPROM loaded with the correct values.

Note: The PLX PCI9060/9080 can be reset and then initialized from its EEPROM by using the **F5_Control** function with the F5_CTL_RESET action parameter. Refer to the function description in this guide and to the source code examples for complete details.

2.5.3. JTAG Reset

Setting the /GRESET input of the JTAG IN connector (J10 pin 20) to 0 volts resets:

- All processor nodes
- JTAG Out
- Support logic

The /GRESET signal is a TRISTATE line that is shared across all F5 Carrier Board boards connected by a 20 pin JTAG connector. All F5 Carrier Boards that are connected together through a JTAG chain can be simultaneously reset by asserting this line. This ensures that all F5 Carrier Boards that are connected together via COMM PORT cables are reset simultaneously to avoid any damage due to contention on the COMM PORT direction settings.

To resume normal operation, the /GRESET signal must be tristated by an external driver. Pull up resistors on the board can then place the line at 5 volts.

2.6. Booting the F5 Board

Jumper J28 is used to select the boot mode of the embedded ‘C44. When installed, the ‘C44 boots from its PEROM; when removed, the COMM port boots the ‘C44. (See *section 4.1* for the location of J28.)

If PEROM is used to boot the ‘C44, Spectrum’s “Bootloader” will automatically be loaded from Node A's PEROM to the ‘C44 when the board is powered up or RESET.

3 Software Overview

The F5 Software Development Kit (SDK) consists of the following items:

Windows 95:	**Windows NT:**
• F5 Windows 95 Host Application Library (ALIB_W95) • F5 Windows 95 Device Driver • F5 C4x Application Library (ALIB_C4x) • Examples • Utilities • Toolbox Utility • Manuals	• F5 Windows NT Host Application Library (ALIB_NT) • F5 Windows NT Kernel Interface Library and Device Driver • F5 C4x Application Library (ALIB_C4x) • Examples • Utilities • Toolbox Utility • Manuals

Note: A complete set of source code and make files for the Windows device drivers and Windows host application libraries are **not** provided with the F5 SDK.

The Host SDK includes only certain source code files, such as ALIB_W95 and ALIB_NT; these are provided for reference only and do not form a complete set. You won't be able to successfully compile and link these programs.

3.1. Hardware and Software Requirements

The following table lists the F5 software installation and development requirements.

Table 3 F5 Hardware and Software Requirements

Hardware	Software
• A x486 (or higher) PC computer with a free PCI slot and at least 16MB of RAM, 2MB of free hard drive space • An F5 PCI Carrier Board	• Windows 95 or NT 4.0 operating system • Texas Instruments' TMS320C4x DSP C Compiler, Assembler, and Linker • A TMS320C4x debugging system (recommended) • **Windows NT:** Visual Basic, version 5.0 (to run the example programs). • A 32-bit Windows C compiler (for host code development). We recommend Microsoft Visual C/C++: **Windows 95:** version 5.0 **Windows NT:** version 5.0

3.2. F5 Host Program Structure

The main difference between the Windows 95 and the Windows NT Host Program Flow lies in the Device Driver layer; the Windows NT version of the F5 SDK has an additional Kernel Interface Library layer, as can be seen in the following diagram.

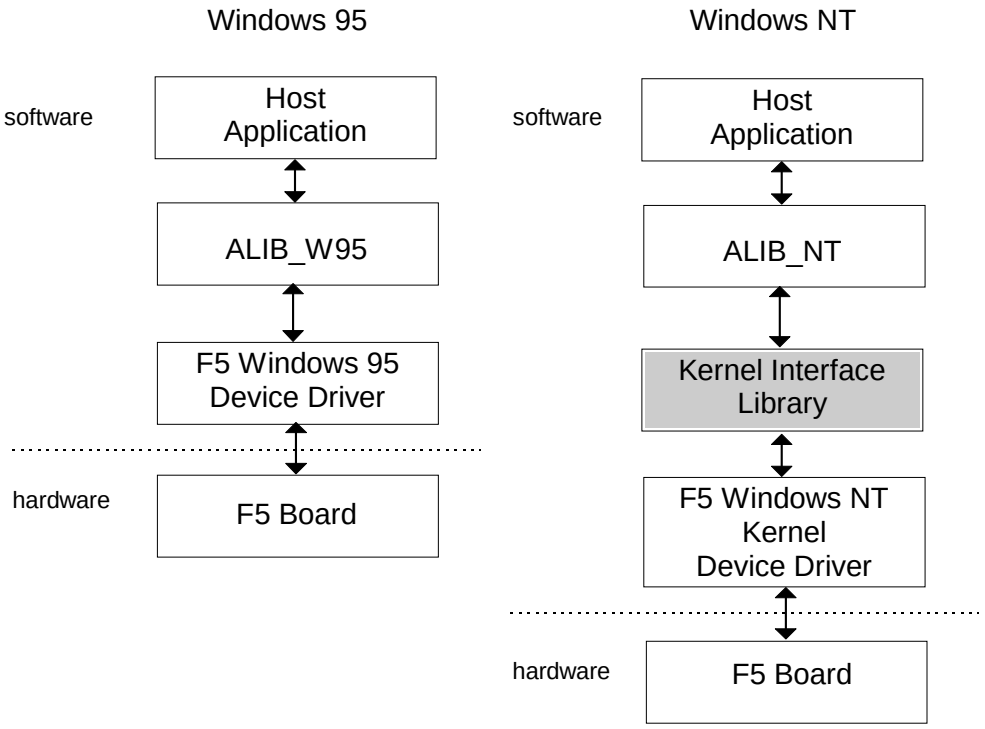


Figure 3 F5 Host Program Flow

3.2.1. F5 Windows Host Application Libraries

Both the ALIB_W95 and the ALIB_NT are high level 32-bit Dynamic Link Libraries (DLLs) that provide your host system with functions for initializing, downloading code to, and performing I/O operations on an F5 board. Both ALIBs support multiple processes and/or threads and can access multiple F5 boards concurrently.

ALIB_W95 runs under Windows 95, and ALIB_NT runs under Windows NT 4.0. ALIB_NT supports Visual Basic version 5.0.

Note: The application libraries are described further in *chapters 5* and *7*.

3.2.2. F5 Windows 95 and NT Device Drivers

Windows 95:

The F5 Windows 95 Device Driver (**vf5d.vxd**) provides (through ALIB_W95) the lowest level software interface to basic control and I/O routines. The device driver functions are actually hidden behind the provided F5 Host Application Library (ALIB_W95). That is, your application calls the ALIB_W95 functions, which in turn call the "lower-level" device driver functions, as illustrated in *Figure 3*.

Note: Although you can access the device driver functions directly, we recommend that you use the provided ALIB_W95 host functions to communicate with the F5 board.

Windows NT:

The F5 Windows NT Kernel Device Driver (**udrv.sys**) provides the lowest level software interface to basic control and I/O routines. The device driver functions are actually hidden behind the provided F5 Host Application Library (ALIB_NT). That is, your application calls the ALIB_NT functions, which in turn call the "lower-level" device driver functions, as illustrated in *Figure 3*.

The Kernel Interface Library (**kintssp.dll**) provides a standard interface for the ALIB_NT to access the Windows NT Kernel Device Driver.

3.3. F5 C4x Application Library

The F5 C4x Application Library (ALIB_C4x) provides your DSP application with routines to configure and transfer data to F5 resources (Shared SRAM and PCI). You can develop C4x DSP application code for the F5 on any platform which supports Texas Instruments' TMS320C4x development tools.

The ALIB_C4x is described further in *chapter 8 Development of DSP Applications*, and the functions are detailed in *chapter 9 DSP Software Functions*.

3.4. Data Type Definitions

Data types used in F5 Host and DSP code are defined in two header files:

- **f5user.h** defines data types used in F5 Host code
- **f5_c4x.h** defines data types used in F5 DSP code

The most common data type used in both host and DSP code is UINT32. UINT32 is defined as a 32-bit wide unsigned integer.

3.5. Calling Conventions for Windows 95 and Windows NT

Windows 95:	**Windows NT:**
uses C calling convention (-cdecl). For example: `F5API RESULT F5_Control (…)`	uses standard calling convention (-stdcall). For example: `F5API RESULT` **`CCC`** `F5_Control (…)` `CCC` - C calling convention macro - used for Application Library (ALIB) exported functions

Table 4 Calling Conventions for Windows 95 and Windows NT

Note: The syntax for the functions described in *chapter* 7 *Host Software Functions* and *chapter* 9 *DSP Software Functions* is shown using the C calling convention. For Windows NT, use the calling convention as shown in the above table.

3.6. Example Programs

The F5 SDK provides example programs that demonstrate how to use the various F5 application library functions. Most of the programs are provided in both C and Visual Basic versions.

The examples can be found in the *<F5RootDirectory>\examples* subdirectories. See *chapter 10* for a description of the programs and details on running and rebuilding the example programs.

Note: The Visual Basic (VB) examples are for the Windows NT environment only. To run the Visual Basic examples, the user must have Visual Basic version 5.0 and must possess a working knowledge of VB 5.0 development environment and programming.

3.7. Manuals Provided With the F5 SDK

The manuals provided with the F5 SDK are:

- *Dakar F5 Carrier Board Technical Reference*
- *Dakar F5 Carrier Board User Guide (this manual)*
- *Toolbox Configuration Utilities User Guide*

3.8. Utilities Provided With the F5 SDK

Utilities are provided with the F5 SDK to allow you to do the following:

To..	Use:	Filename
List all F5 boards on a host system, including their hardware configurations.	F5 List Inspector	f5list.exe
Test the F5 ALIB. Each function can be tested separately. (See *section 4.10* for more details.)	F5 SDK Tester	f5lb_t.exe
Windows 95 only: Display debug messages from both the Windows 95 Device Driver and the F5 Host ALIB functions in debug mode. (See *section 6.9 Viewing Debug Messages (Windows 95)* for more details.)	DBGMON	dbgmon.exe
Windows NT only: Read and Write directly to/from the board. (Use the **About** button for more details.)	F5 IO Tester	f5io.exe
Toolbox is used to generate System Definition Files (SDF). See appendix C for more information on SDF. Refer to the "Toolbox Configuration Utilities User Guide" for installation and usage of toolbox.	Toolbox	toolbox.exe
GNU utilities are used to rebuild DSP-based examples.	PC GNU Utilities	mkdepend.exe

All of the above utilities, except Toolbox and GNU, can be found in your *<F5RootDirectory>\bin* directory. Utilities can be run by double-clicking on the utility's filename (in Explorer) or by going to the **F5 SDK** Program Folder in the **Start** > **Programs** menu.

The Toolbox and GNU utilities are located on separate disks shipped with the **F5 SDK** package.

4 Hardware and Software Installation

This chapter describes how to configure and install the F5 Carrier Board into a computer. It also describes how to install the Software Development Kit (SDK) for the F5 Carrier Board. The F5 Carrier Board host and DSP libraries, device driver, example programs, and utilities are all part of the SDK.

Caution: Before handling the F5 Carrier Board or a module, ensure that you and the components are properly grounded to prevent damage from electrostatic discharge.

4.1. Configuring the F5 Carrier Board

Configure the on board jumpers to suit your application. Refer to the following figure and table for information on the jumper settings.

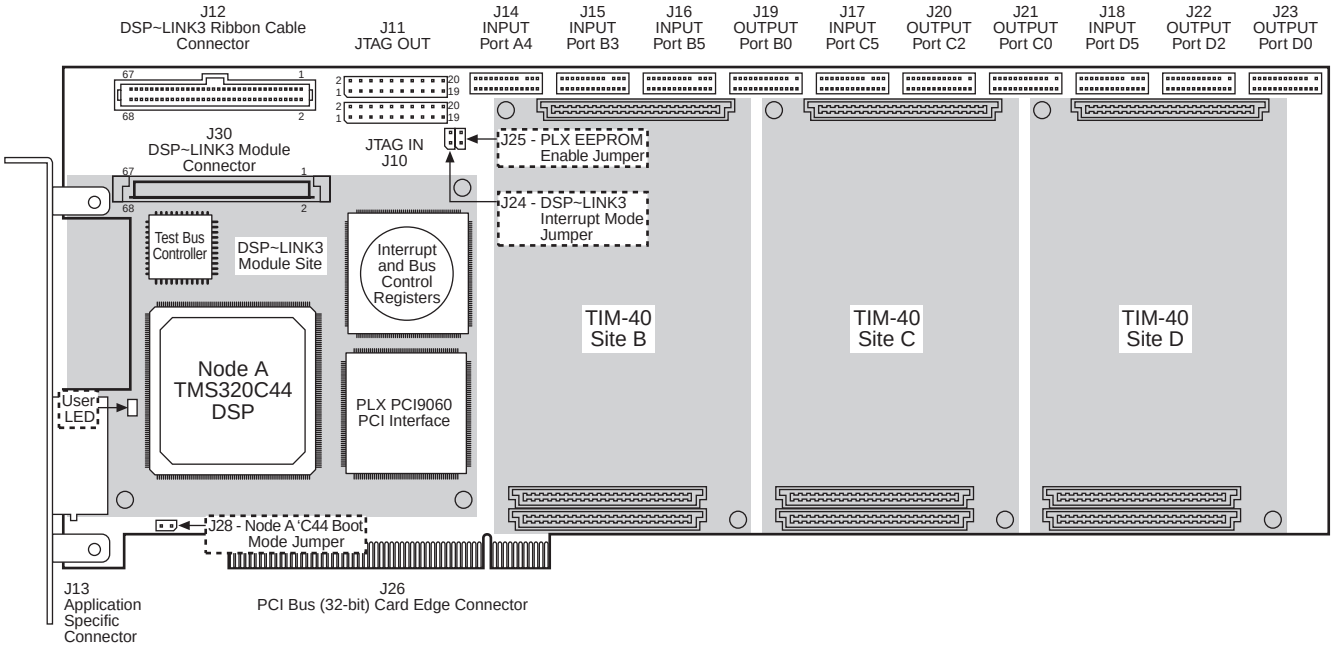


Figure 4 F5 Connector and Jumper Locations

Table 5 F5 Jumper Summary

Jumper	Description	Installed	Not Installed
J24	DSP~LINK3 Interrupt Mode	All DSP~LINK3 interrupts to Node A IIOF0	DSP~LINK3 interrupt IRQ0 to Node A IIOF0*
J25	PLX PCI9060/9080 EEPROM Enable	Enabled*	Disabled
J28	Node A 'C44 Boot Mode	Boot from PEROM*	Boot from COMM Port

*Default

4.2. Module Installation

Your F5 Carrier Board may already contain DSP~LINK3 or TIM-40 modules. If it doesn't, or you are adding modules to it, follow these instructions to install them on the F5 Carrier Board.

Caution: Before handling the F5 Carrier Board or module, ensure that you and the components are properly grounded to prevent damage from electrostatic discharge.

The following figure shows the location of the modules on the F5 Carrier Board and the mounting hardware for the sites.

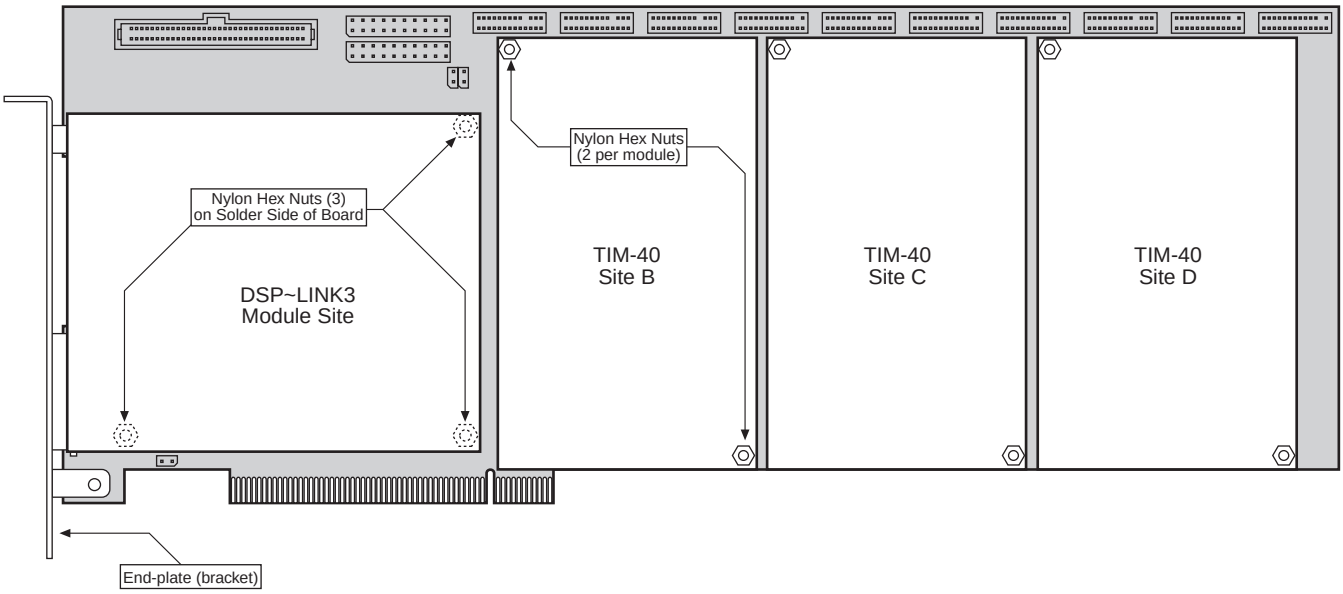


Figure 5 F5 Module Locations

4.2.1. Installing DSP~LINK3 Modules

To install a DSP~LINK3 module onto the F5 Carrier Board:

1. If the module that you are installing has its own end plate bracket, remove the end plate bracket that is attached to the F5 Carrier Board.
2. Place the DSP~LINK3 module on the F5 Carrier Board. Align the three nylon mounting posts on the DSP~LINK3 module with their corresponding holes on the F5 Carrier Board.
3. Hand tighten the three 3mm nylon hex nuts to the mounting posts on the solder side of the F5 Carrier Board to secure the module in place.
4. Screw the end plate of the module to the F5 Carrier Board.

4.2.2. Installing TIM-40 Modules

To install a TIM-40 module onto the F5 Carrier Board:

1. Use the nylon mounting posts on the TIM-40 sites to position the TIM-40 module.
2. Press down on the module to mate the top, bottom, and global connectors.
3. Secure the module on to the F5 Carrier Board by hand-tightening the nylon hex nuts that came with the module on to the mounting posts.

4.3. Installation into PCI Slot and Cabling

After the jumpers on the F5 Carrier Board have been configured, it can be installed in a PC and any other connections can be made.

Caution: Use the keyed COMM Port cables from Spectrum to prevent damage to the F5 Carrier Board when making COMM Port connections. These cables are keyed to prevent accidental connection of outputs to outputs, or inputs to inputs. If you use other cables, be careful to avoid connecting a default output to a default input COMM Port.

---

1. Ensure that all hardware is powered off.
2. Attach the JTAG, DSP~LINK3 module, and/or DSP~LINK3 cables to the board if you are using them.
3. Install the F5 Carrier Board into an empty PCI slot of the PC computer.
4. Attach the COMM Port and Application Specific Connector cables to the board if you are using them.
5. Do **not** power on the system at this time. Go to the next section..

4.4. Windows NT Installation Notes

> **Note:** This section is only applicable if you are installing the F5 SDK onto a Windows NT platform.

Before installing the F5 SDK make sure the Plug and Play BIOS is disabled.

4.5. Windows 95 Installation Notes

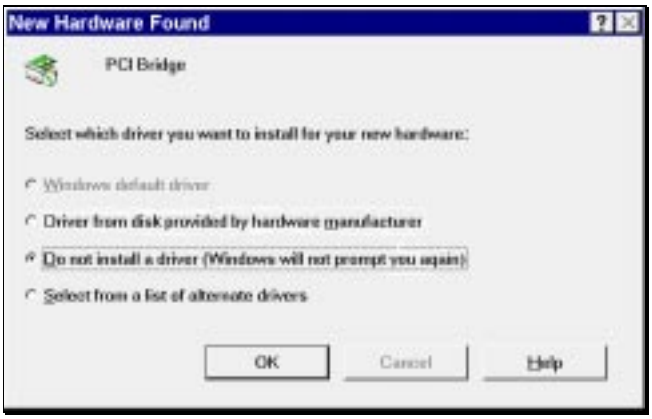
> **Note:** This section is only applicable if you are installing the F5 SDK onto a Windows 95 platform. If you are not installing onto a Windows 95 platform, go to *section 4.6*.

The first time you plug an F5 Carrier Board into a computer running the Windows 95 operating system, Windows 95 will prompt you to set up the new hardware. This section describes how to correctly perform this initialization procedure. If not done properly, the Plug and Play component of Windows 95 will not initialize the F5 Carrier Board correctly.

> **Note:** The initialization procedure will vary slightly depending upon which particular build of Windows 95 you have running on your machine.

1. If the F5 Carrier Board is not installed in the computer, turn off the computer and install the F5 Carrier Board. For information on how to do this, see the previous section.
2. Turn on the power. Windows 95 should automatically launch.
3. Depending upon which build of Windows 95 your computer is running, one of two dialog boxes will appear:
   - New Hardware Found dialog box (for build 4.00.950a)
   - Install Device Driver Wizard dialog box (for build 4.00.950b)
4. Perform the steps corresponding to the dialog box that is displayed on your system.

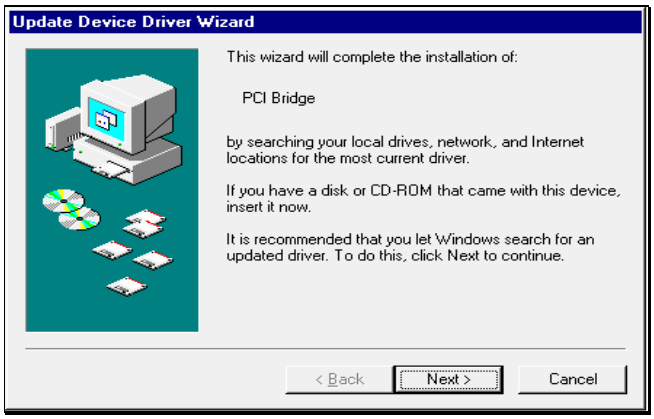
If the New Hardware Found dialog box appears…



Windows 95 Build 4.00.950a

1. Click **Do not install a device driver…**
2. Click **OK**.

If the Install Device Driver Wizard dialog box appears…



Windows 95 Build 4.00.950b

1. Ensure that drive A: and the CD-ROM drive are empty.
2. Click **Next.**
3. Windows 95 will search drive A: and the CD-ROM drive for a driver file. When Windows 95 is finished you will be prompted for Other Locations.
4. Click **Finish.**

Windows 95 is now configured for the F5 Carrier Board to work on the PCI bus and will not ask you to install drivers again for the F5 Carrier Board on this computer.

Note: You can also determine the build of Windows 95 that you have as follows. From the Explorer window in Windows 95, right-mouse click on the My Computer icon. Select Properties. Select the General tab. The System list will indicate the Windows 95 build version.

4.6. Installing Multiple F5 Carrier Boards

If multiple F5 Carrier Boards are installed on one PCI bus, then each board must have a unique identification (ID). This ID is stored in the PEROM of Node A of each F5 Carrier Board. By default, all F5 boards are shipped with an ID of 0x1.

Refer to *section 6.5 Changing your System's Hardware Configuration* for information on setting the F5 Carrier Board ID value.

4.7. Installing the Software Development Kit (Windows NT)

> **Note:** If you already have a version of the F5 Software Development Kit (SDK) installed on your system, ensure that you first uninstall it. See *section 4.12 Uninstalling the Software Development Kit (Windows NT)* for instructions on uninstalling the SDK.

When installing the SDK, you must install both the Host SDK and the DSP SDK We recommend you install the Host SDK first, then the DSP SDK. The steps are identical for each installation.

To install the Software Development Kit (SDK) for the F5 Carrier Board onto a Windows NT platform, follow these steps:

1. Ensure that you have administrative privileges.
2. Insert the installation disk into the 3.5 inch floppy disk drive of your development PC.

To install the Host SDK, use:	To install the DSP SDK, use:
"DAKAR WinNT Host SDK" disk	"DAKAR WinNT C4x DSP SDK" disk

3. From the **Start** menu, select **Run**, and, in the Run dialog box, type:

   `a:\setup.exe`

   where `a:\` is the floppy disk drive containing the installation disk.

4. Select **OK** and wait for the computer to display instructions on the screen. The InstallShield® Wizard will guide you through the setup process. Follow the instructions.

> **Note:** We recommend you accept the default installation directory.

5. When the screen titled

   `"Dakar (F5) Host Software Development Kit Setup Complete"`
   or
   `"Dakar (F5) DSP Software Development Kit Setup Complete"`

   appears, select **Yes, I want to restart my computer now.**

6. Click **Finish**. The computer will reboot.

7. Repeat steps 2 to 6, until you have installed first the Host SDK and second the C4x DSP SDK.

8. Insert the PC GNU Utilities floppy into the 3.5" floppy drive. Repeat steps 2 to 4 to install the PC GNU Utilities. Once complete remove the floppy from the disk drive.

During the installation process, the F5 SDK files are copied into the specified directory on your hard drive (the default directory is **C:\F5SDK**). In addition:

- The F5 Host Application Library file (**F5alib.dll**) and the F5 Windows NT Kernel Interface Library file (**kintssp.dll**) are copied to your *c:\windows* directory.
- The F5 Windows NT Kernel Device Driver file (**udrv.sys**) is copied to your *c:\windows\system32\drivers* directory (this is where Windows NT looks for this file).
- The installation program makes the appropriate changes in the Windows NT Registry so that **udrv.sys** can be loaded automatically when Windows NT reboots. After a successful installation, Windows NT is configured for the F5 board to work on the PCI bus.
- The following Environment Variable is automatically set in the Windows NT Registry:

  ```
  SET SSP_PATH=<F5RootDirectory>\BIN;<F5RootDirectory>\EXAMPLES\DSPLOAD
  ```

4.8. Installing the Software Development Kit (Windows 95)

Note: If you already have a version of the F5 Software Development Kit (SDK) installed on your system, ensure that you first uninstall it. See *section 4.13 Uninstalling the Software Development Kit* for instructions on uninstalling the SDK.

To install the Software Development Kit (SDK) for the F5 Carrier Board onto a Windows 95 platform, follow these steps:

1. Insert the "Dakar Win95 SDK" installation disk into the 3.5 inch floppy disk drive of your development PC.
2. From the **Start** menu, select **Run**, and, in the Run dialog box, type:

   `a:\setup.exe`

   where `a:\` is the floppy disk drive containing the installation disk.
3. Select **OK** and wait for the computer to display instructions on the screen. The InstallShield® Wizard will guide you through the setup process. Follow the instructions.

   Note: We recommend you accept the default installation directory.
4. When the "`Setup Complete`" screen appears, select **Yes, I want to restart my computer now**.
5. Click **Finish** and remove the installation disk from the floppy drive. The computer will reboot.

6. Insert the PC GNU Utilities floppy into the 3.5” floppy drive. Repeat steps 2 and 3 to install the PC GNU Utilities. Once complete remove the floppy from the disk drive.

During the installation process, the F5 SDK files are copied into the specified directory on your hard drive (the default directory is **C:\F5SDK**). In addition:

- The F5 Host Application Library file (**f5alib.dll**) is copied to your *c:\windows* directory.
- The F5 Windows 95 Device Driver file (**vf5d.vxd**) is copied to your *c:\windows\system* directory (this is where Windows looks for this file).
- The following DOS command is automatically inserted into your **autoexec.bat** file:

  ```
  SET SSP_PATH=<F5RootDirectory>\EXAMPLES\DSPLOAD
  ```

4.9. Directory Contents After Installation of the SDK

After you’ve installed the Software Development Kit for the F5 Carrier Board, the destination root directory that you specified during installation will contain the directories listed in the following table. Refer to any installed **readme.txt** files for the most current information on the F5 Carrier Board SDK.

Table 6 Directory Contents After Installing the SDK

Windows NT - Contents		Windows 95 - Contents	
Directory	**Description**	**Directory**	**Description**
<F5 Root>		<F5 Root>	
bin	Executable files for the DSP and Host (supplied by Spectrum)	bin	Executable files for the DSP and Host (supplied by Spectrum)
dev		alib	Some folders under *alib\host* are empty until code is built, See *section 10.6* for details
		shared	
		include	Contains header files
dsp		host	
bin	Output files - DSP binary files after you compile and link your DSP programs	bin	
		debug release	Contains f5alib.dll
include	DSP Include files	build	Contains make file f5alib.mak
lib	DSP application library file	include	Host include files
src	DSP source code *dev\dsp\src* contains different subdirectories for different modules	src	F5Alib source code
		lib	Contains cofflib.lib, sdflib.lib
		debug release	Contains f5alib.lib
drv	F5 Windows NT Kernel Device Driver	driver	F5 Windows 95 Device Driver
examples	Source code example programs. Each example is in a subdirectory of *examples*. For example, **dspload** program is in *\examples\dspload* subdirectory.	examples	Source code example programs. Each example is in a subdirectory of *examples*. For example, **dspload** program is in *\examples\dspload* subdirectory.
include	DSP and Host Include files. SDF files.	include	DSP and Host Include files. SDF files.
lib	DSP and Host application library files (supplied by Spectrum)	lib	DSP and Host application library files (supplied by Spectrum)
src	Host source code for the application library. See note at the beginning of *chapter 3.*		

4.10. Installing Toolbox

Refer to the Toolbox Configuration Utilities User Guide for complete information on installing Toolbox.

4.11. Verifying the Software Development Kit Installation

There are a number of programs that can be run to ensure that the F5 SDK is functioning properly. If you are just becoming familiar with the F5 SDK, we recommend to run the **dspload** example program.

Make sure your system configuration corresponds to the SDF used by the examples. All examples use the **F5.sdf** System Definition File (SDF) that's in *<F5RootDirectory>\include*. This SDF defines the following default system configuration:

- a single board system
- board ID = 0x1
- one embedded processor, no TIM modules

To run the **dspload** example program:

1. Go to the command prompt.
2. Change the directory to *<F5RootDirectory>\examples\dspload*

   where *<F5RootDirectory>* is the location of the F5 SDK files on your hard drive. It specifies the directory on your hard drive from which your F5 SDK files branch, and is most likely the directory you chose during installation.
3. Type `dspload`

   Your screen should resemble the following:

```
c:\f5sdk\Examples\Dspload\dspload

Opening system. . .    [OK]
Resetting DSPs. . .    [OK]
Loading DSP code . . . [OK]

on-board LED should now be blinking.

Closing system. . .    [OK]

c:\f5sdk\Examples\Dspload>
```

If the test is successful, the on-board LED will be blinking.

For information on other example programs, see *chapter 10 Example Programs.*

If you wish to do more detailed testing, see *section 10.4 Verifying the Device Driver and Host Library Installation*.

For troubleshooting information, see *section 10.5 Tips and Troubleshooting*.

4.12. Uninstalling the Software Development Kit (Windows NT)

To uninstall the F5 SDK:

1. Click the **Start** button, then point to **Settings**.
2. Click **Control Panel**.
3. Double-click **Add/Remove Programs**.
4. With the **Install/Uninstall** tab selected, a list of programs will be displayed.
5. Click **Dakar (F5) DSP Software Development Kit**, then click **Add/Remove**.
6. Click **Dakar (F5) Host Software Development Kit**, then click **Add/Remove**.

Note that if you use the "Un-install F5 SDK" option (**Start > Programs > F5 SDK Program Folder**), only the Host SDK will be uninstalled.

If the SDK has been successfully uninstalled, all components of the F5 SDK (except the files **udrv.sys** and **kintssp.dll**) will be removed from your computer. If you are using other Spectrum products on your computer system, do not remove these files. If you are not using other Spectrum products, you may manually delete these two files.

4.13. Uninstalling the Software Development Kit (Windows 95)

To uninstall the F5 SDK:

1. Click **Start**, then point to **Programs**.
2. Point to **F5 SDK**, then click **F5 Uninstall**.

Another way to uninstall the F5 SDK is through the Windows Control Panel (see your Windows documentation for more information).

The Uninstall program does not delete changes made by the installation program to **autoexec.bat**: PATH and SET SSP_PATH. You will have to manually delete these changes.

5 JTAG Debugging Software

The F5 Carrier Board supports JTAG emulation using the on-board Test Bus Controller (TBC), otherwise known as back-plane debugging, or using an external emulation board from White Mountain (the Mountain-510 Universal Emulator). The debugging application supported for both back-plane and external emulation is GO DSP's Code Composer™.

The following diagram illustrates the two types of debugging that can be performed.

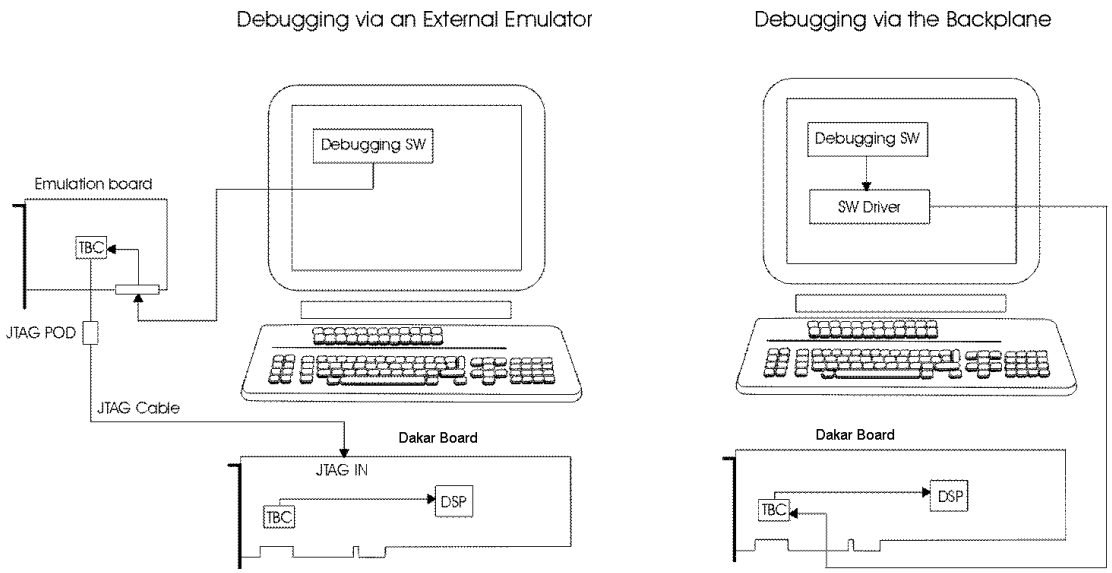


Figure 6 JTAG Debugging Configurations

Note: If a JTAG IN connection with a clock signal is present, the Test Bus Controller (TBC) will be automatically disconnected.

Refer to the JTAG Debugging section of the *F5 Carrier Board Technical Reference* manual for further information about JTAG debugging and setting up a JTAG chain for multiple board debugging.

5.1. Using Code Composer

This section describes how to set up Code Composer for use with the F5 Carrier Board and how to load and run files using Code Composer. For further details, refer to the *Code Composer User's Guide* from GO DSP.

5.1.1. Setup for Back-plane Debugging

Follow this procedure to set up Code Composer for back-plane debugging.

1. Ensure that you've installed the Dakar Win NT Host SDK (Dakar Win 95 Host SDK, for Windows 95) and the C4x DSP SDK.

2. Install Code Composer.

3. Insert the Spectrum *Code Composer C4x MSTR* disk into the 3.5 inch floppy disk drive of your PC. This disk contains a sample GEL file and the DLLs that are required by Code Composer to access Dakar devices. Read the `readme` file provided on this disk for the most current information about the disk.

4. Copy the `ssf5_4xnt.dll (ssf5_win95.dll for Windows 95)`and the `f5_cpu_a.gel` files that are on the Spectrum *Code Composer C4x MSTR* disk to the *<CodeComposerRoot>* directory, where *<CodeComposerRoot>* is the base directory containing your Code Composer files.

5. Ensure that the `f5_cpu_a.gel` file properly reflects the memory maps of your system's processor. See *Section 5.1.3* for information about editing GEL files and the *Code Composer User's Guide* for more information about GEL files.

6. Run the Code Composer Setup program by choosing Start→Programs→Code Composer→Setup Code Composer.

7. Within the General Setup tab, click on Add Driver and from the list of files, select the `ssf5_4xntnt.dll (SSF5_4xwin95.dll for Windows 95)` file in the *<CodeComposerRoot>* directory.

8. Enter the Board ID of the Dakar board in the I/O Port text box. For a single board system, enter `0x1` as the I/O Port address.

   For systems with more than one Dakar board, enter the Board ID of the first F5 Carrier Board that is in the JTAG scan chain. You should only have one instance of Code Composer running for all boards in your system.

   Note: Refer to Section 6.5 *Changing your System's Hardware Configuration* for information on setting the F5 Carrier Board ID value.

9. Click on the MultiProcessor tab and enter a processor name for each DSP in your Dakar system.

For example, type `cpu_a` and click Insert.

10. Click on OK. Setup creates a `board.cfg` file that lists the names and types of processors in the JTAG scan path and also creates a `board.dat` file which is a binary file containing JTAG scan path information.

5.1.2. Setup for Debugging with the Mountain-510 Emulator

Follow this procedure to set up for debugging with the Mountain-510 emulator:

1. Ensure that you've installed the Dakar Win NT Host SDK (Dakar Win 95 Host SDK for Windows 95) and the C4x DSP SDK.

2. Install Code Composer.

3. Insert the Spectrum *Code Composer C4x MSTR* disk into the 3.5 inch floppy disk drive of your PC.

4. Copy the `f5_cpu_a.gel` file that's on the Spectrum *Code Composer C4x MSTR* disk to the *<CodeComposerRoot>* directory, where *<CodeComposerRoot>* is the base directory containing your Code Composer files.

5. Install the following:

   - The Mountain-510 emulator board
   - The White Mountain DSP Code Composer Support Disk

   See White Mountain DSP's documentation for installation instructions.

6. Run the Code Composer Setup program by choosing Start→Programs→Code Composer→Setup Code Composer.

7. Within the General Setup tab, click on Add Driver and from the list of files, select the `wm510c4x32.dll` file that's in the *<CodeComposerRoot>* directory.

8. Enter the base address of the Mountain-510 emulator in the I/O Port text box. Refer to the documentation provided by White Mountain DSP for the correct address (normally, the default address is `0x240`).

9. Click on the MultiProcessor tab and enter a processor name for each DSP in your Dakar system.

   For example, type `cpu_a` and click Insert.

10. Click on OK. Setup creates a `board.cfg` file that lists the names and types of processors in the JTAG scan path and also creates a `board.dat` file which is a binary file containing JTAG scan path information.

5.1.3. Editing the Sample GEL File

A GEL file is used to define memory mapped resources for the C4x processor. These registers have to be correctly set in the GEL file in order for your program to load and for the C4x to access its memory correctly. Refer to the F5 Carrier Board *Technical Reference* manual for information on memory resources.

> **Note:** The GEL file will have to be modified if a TIM-40 module is added to the board.

Use a text editor to edit Spectrum's sample GEL file (`f5_cpu_a.gel`)file. Note that you can also edit GEL files from within Code Composer (see the *Code Composer User's Guide* for more details). Refer to the *Code Composer User's Guide* for further information about GEL files.

5.1.4. Loading and Running Files

This section describes how to load and run files, such as DSP code, on the Dakar board using GO DSP's Code Composer.

1. If you are loading and running files via the back-plane of the Dakar board, ensure that there is **no** external cable connected to the front-panel JTAG IN of the Dakar board.

   Note: You cannot use the Dakar's TBC (back-plane debugging) at the same time that you are using an external emulator. In addition, the F5 Carrier Board must be powered off before switching between back-plane and external emulation, and before attaching or removing an external emulator.

2. If you are using the Mountain-510 emulator, power down your system and connect the emulator board to the JTAG IN connector of the Dakar board. Follow the installation instructions provided by White Mountain DSP.

3. Ensure that the `f5_cpu_a.gel` file found in the *<CodeComposerRoot>* directory properly reflects the memory map of the Dakar processor. See *Section 5.1.3* for more information about editing GEL files.

4. Ensure that the `board.cfg` file, created during Code Composer Setup, properly reflects the JTAG scan path of your system. These files can be found in the Code Composer root directory.

5. Start Code Composer by selecting Start→Programs→Code Composer→C4x Code Composer.

6. In the Processor Window, select File→Load GEL and from the list of files, select the `f5_cpu_a.gel` file from the Code Composer root directory.

7. Load the DSP application file by selecting File→Load Program and from the list of files, select the file that you want to load.

   For example, to load `blink.out`, select the `blink.out` file from the `<F5 Carrier Board`*`Root`*`>\examples\dspload` directory.

8. To run the DSP application file, select Debug→Run Free from the Code Composer Main Menu.

   If you loaded `blink.out`, the front panel LEDs should flash on and off.

9. Quit the debugger.

6 Development of Host Applications

The F5 ALIB_W95 and ALIB_NT provide the functions that you need to initialize, download code to, control, and perform I/O operations on an F5 board. This chapter discusses some of the functions that are used to perform these tasks and some important concepts that you should understand before using the Application Library (ALIB) functions.

6.1. Architectural Overview

The architecture of HOST_ALIB is grounded in the concept of resource management and manipulation. Figure 7 is a graphical representation of the architecture of HOST_ALIB, and illustrates how your application code may integrate easily with the library.

The essential elements of the architecture to take note of are:

- the representation of the hardware components in a System Definition File
- generation of the SDF with Spectrum's Toolbox utility
- specification of downloadable DSP code in a Load Definition File
- common sharing of prototypes and definitions (.h files) between ALIB_HOST and your application code

In any application there are four unique components:

- your application code
- a system handle and a collection of resource handles which your code acquires and uses for communication with the hardware
- the HOST_ALIB functions which translate your requests into specific actions on the hardware
- the system configuration and resources which HOST_ALIB manages on your behalf

Note that Toolbox is not a part of your application. It is a tool to be used when you initially install, or make a hardware modification to your Dakar system. The prototype and definition files are only required to build your application. The run-time system consists of your application, the Load and System Definition Files, and the Dakar hardware.

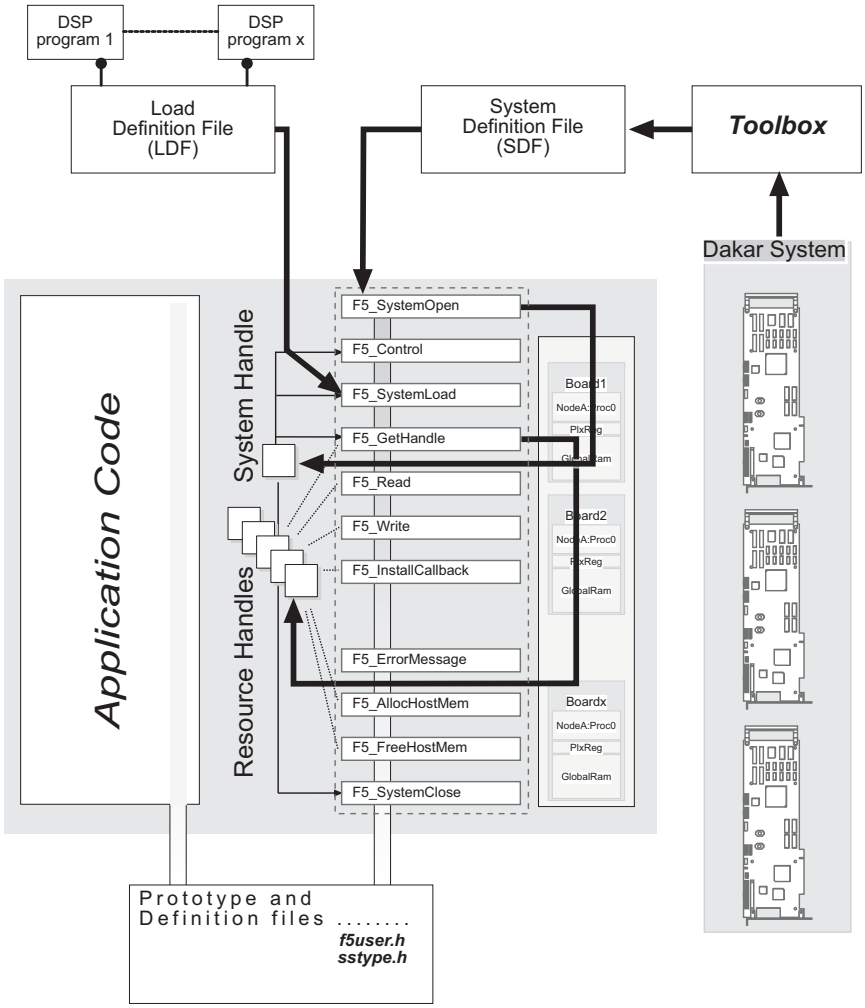


Figure 7 Dakar Host Software Architecture

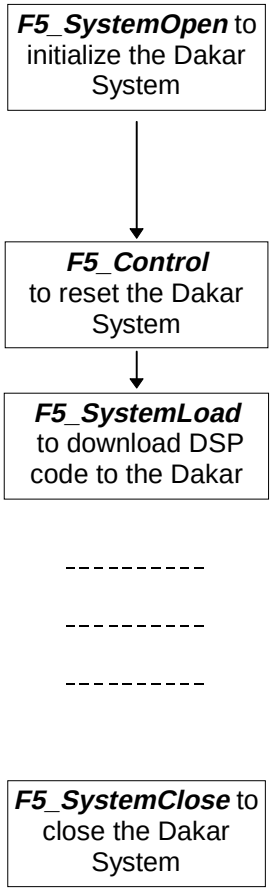
6.2. Calling Host Functions

HOST_ALIB requires that you follow a simple rule for successful applications:

Important! always **acquire** a resource **before** you call functions that **use** that resource.

Resources are managed and manipulated using handles. HOST_ALIB uses two types of handles; a system handle, and a resource handle. An application will always acquire one system handle. The number of resource handles acquired by an application depends on the function of the application.

Applications must call certain functions in a strict order, as shown in *Figure 8*. This ensures that resources are available when needed by calls to other functions.



The first function called in your host code must be F5_SystemOpen. F5_SystemOpen initializes and opens your Dakar system. This function returns a *system handle*.

To download DSP code to the Dakar board, first call F5_Control to reset the board, followed by F5_SystemLoad to download the code. These functions require a *system handle*.

Within the main body of your code you acquire handles to resources, and use them to perform the application specific functions. You can also use F5_Control and F5_SystemLoad repeatedly within the application to reset and reload the Dakar board.

Finally, your host code must call F5_SystemClose to close the system.

Figure 8 Host Program Flow

6.2.1. Obtaining Error Information

All ALIB_HOST functions can return an error value (that is, a non-zero result). If you wish to display a corresponding message for the error value, you can call the function F5_ErrorMessage. The F5_ErrorMessage function takes an error value as an argument, and returns a text description of the error. Error code values and their descriptions are listed in *Appendix A*.

6.3. Including F5 ALIB Functionality

In order to use the F5 Application Library (ALIB) functions, make sure you:

1. Include the **f5user.h** file in your code. This file contains all the required F5 function prototypes and defines and can be found in the *<F5RootDirectory>\include* directory.
2. Link your code with the **f5alib.lib** file that's found in your *<F5RootDirectory>\lib* directory. This will link your code with the F5 ALIB functions stored in the **f5alib.dll** file.

   The f5alib.dll file must be in your *c:\windows* or current directory. The installation program automatically installs this file into your *c:\windows* directory.
3. Set the Environment Variable SSP_PATH to include the location of the following files:

   - The **sspboot.out** file. This file is loaded into each processor in the loading path, except the boot processor (embedded processor).
   - Load Definition Files (LDFs) containing the loading path for the DSP code (see *section 6.6* for more information on LDFs).
   - DSP application code files to load onto the system's processors.

   Your application first looks for these files in the current directory and, if they are not found, they are then searched for in the path specified by SSP_PATH. For example, if the environment variable is set as:

   ```
   SSP_PATH=C:\F5\BIN;D:\F5USER
   ```

   your application first checks the current directory for the files, and if the files are not found, your application then checks the `C:\F5\BIN` directory, and then the `D:\F5USER` directory.

   SSP_PATH should include *<F5RootDirectory>\bin* and *<F5RootDirectory>\examples\dspload*

Note: To temporarily set the SSP_PATH Environment Variable, you must set it in the same Command Prompt session that your console program is running.

Windows 95:	**Windows NT:**
SSP_PATH can be set via the AUTOEXEC.BAT file.	SSP_PATH can be set or changed via the Control Panel, System applet, Environment tab, User Variables list box.

6.4. Defining your System's Hardware Configuration

A System Definition File (SDF) is used by the **F5_SystemOpen** function to define your system's hardware configuration and to compute the loading path for all processors in the system. The SDF describes the hardware configurations of all the F5 boards in a system, such as memory types and sizes, TIM modules, COMM Port connections, interrupt levels and vectors, board base addresses, boot processors, and other parameters.

Note: There should be one SDF for each of your F5 systems. Multiple F5 boards can be treated as different systems only if there are no COMM port connections between them.

Sample SDF files have been included with the F5 SDK and can be found in your *<F5RootDirectory>\include* directory:

- **f5.sdf** and **f5_1.sdf** define a single board system with one embedded processor only.
- **f5_2.sdf** defines a single board system with one embedded processor and one TIM module.
- **f5_3.sdf** defines a single board system with one embedded processor and two TIM modules.
- **f5_4.sdf** defines a single board system with one embedded processor and three TIM modules.

Note: The SDF will have to be edited every time you add an F5 board, TIM module or memory to your system. (See *section 6.5 Changing your System's Hardware Configuration* for more details.)

For a detailed description and example of a System Definition File, see *Appendix C: System Definition File: Description and Example*.

6.5. Changing your System's Hardware Configuration

Each time you add an F5 board, TIM module or memory to your system, you must edit the System Definition File (SDF) used in your application to reflect your new hardware configuration. For example, if you add an F5 board to your system, the board will have to be defined in the SDF with its unique board ID (base address).

> **Note:** You can generate a new SDF using Spectrum's Toolbox Utility. When generating a new SDF, Toolbox also creates a corresponding LDF. Be sure to use this new LDF file in the place of your old LDF.

A board's ID is contained in PEROM of its embedded C44. By default, all boards are shipped with a board ID of 0x1.

6.5.1. Changing an F5 Board ID

> **Note:** If your system contains more than one F5 board, you must ensure that each board has a unique board ID.

To change a board's ID, for example to 0x2:

1. Remove all other F5 boards from your system.

2. Remove jumper J28 on the F5 Board to be programmed.

3. Using your debug monitor, load the boot kernel file, **xxxx_a2.out**, that's provided in your *<F5RootDirectory>\bin* directory onto the embedded node of the target board. xxxx, in the filename xxxx_a2.out, corresponds to the board's type; either M1S1 or M2S2 (see *section 2.2*).

   Note that **xxxx_a3.out** corresponds to board ID 0x3, and so on.

4. Run the **xxxx_a2.out** file and quit the debug session.

> **Note:** The new board ID is effective only after the board has been reset. See *section 2.5 Resetting the F5 Carrier Board*.

6.6. Defining the DSP Code to Download

A Load Definition File (LDF) is used by the **F5_SystemLoad** function to determine which DSP applications to load to which F5 processors. The LDF is a text-based file containing the paths and filenames of the DSP executable files to load. Sample LDF files can be found in the *<F5RootDirectory>\examples* subdirectories. Use a text editor to edit one of these file.

6.6.1. Load Definition File (LDF) Example

```
[Files]
      Board1:SiteA:Proc0 = intrupts.out  ;  // Embedded TIM A
      Board1:SiteB:Proc0 = intrupts.out  ;  // TIM B
      Board1:SiteC:Proc0 = intrupts.out  ;  // TIM C
      //Board1:SiteD:Proc0 = loop.out  ;    // TIM D

[end]
```

6.7. Defining your System's Configuration for a Standalone Application

A Resource Definition File (RDF) is used in a standalone application to define your system's configuration. An RDF is generated from a System Definition File (SDF), described in *section 6.4*, and consists of the following:

- Resource Table – an array of system resources
- Processor File List – an array of DSP applications to load to F5 processors

---

Note: There should be one RDF for each F5 system.

To use the RDF in a standalone application:

- Include the RDF, which is in "C" code format, in your host code.
- Pass the name of the Resource Table to **F5_SystemOpen**. The Resource Table is used by F5_SystemOpen to define your system's hardware configuration.
- Pass the name of the Processor File List to **F5_SystemLoad** when loading DSP code. The File List identifies which DSP applications to load to which processors.

See the following section for instructions on generating an RDF.

Note: Refer to Toolbox Configuration Utilities User Guide for Generating SDF and RDF files.

6.7.1. Generating a Resource Definition File (RDF)

Sample RDFs are provided with the F5 SDK and can be found in your *<F5RootDirectory>\include* directory. Using a text editor, you can edit one of these files to properly reflect your system's configuration, or you can generate an RDF using the **Toolbox Utility**.

To generate an RDF using Toolbox:

run c:\spectrum\toolbox\toolbox.exe (Win32)

Generally the sequence of operations is as follows:

1) Select Define a system of boards from the Main Menu
2) Choose a System name
3) Enter from menus the hardware in your System
4) Generate a System Definition File (SDF)
5) Generate a Resource Definition File (RDF), if required
6) Exit

The files generated by ToolBox (SDF, LDF, RDF, etc.) are placed in the directory:

c:\spectrum\toolbox\system\ (Win32)

To use these files with the Host application libraries, copy the required files to the directory where you are developing your host code.

For some host ALIBs which use SDF files, the ssp_path environment variable can be set to search the system directory (as well as the toolbox directory) to automatically find the generated SDF/LDF files. By setting the ssp_path environment variable it is not necessary to move the files after they have been generated.

6.7.2. Resource Definition File (RDF) Example

```
/*=========================================================================
 Spectrum Signal Processing Inc.

    RESOURCE FILE:   E:\F5-10\DEV\HOST\BIN\f5.rdf
    SOURCE SDF FILE: E:\F5-10\DEV\HOST\BIN\f5.sdf
    PURPOSE:         Resource File to be used by user application.

    COMMENTS:        This file was from the user SDF file.

    TIMESTAMP:       Fri Jul 18 15:56:03

  =========================================================================*/

#ifndef SS_RESRCE_H
#define SS_RESRCE_H

#ifdef  __cplusplus
extern C {
#endif


F5_RESOURCE F5_System[] =
{
   { /* BEGIN RESOURCE: */
      0x40, /* RescSize */
      "Board1:SiteA:Proc0", /* Full Resource Name */
      "", /* UserCode */
      {255, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, /* Boot Path */
      "Board1:SiteA:Proc0", /* Root Resource Name */
      0x20, /* TypeID */
      0x0, /* hNextResc */
      0x123, /*Board ID*/
      0x0, /*Board Ref*/
      0x0, /*Mem Offset*/
      0x0, /*Mem Length*/
      0x3d840000, /*GMCR*/
      0x3d840000 /*LMCR*/
   }, /* END OF RESOURCE */

...

   { /* BEGIN RESOURCE: */
      0x0, /* RescSize */
      "", /* Full Resource Name */
      "", /* UserCode */
      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, /* Boot Path */
      "", /* Root Resource Name */
      0x0, /* TypeID */
      0x0, /* hNextResc */
      0x0, /*Board ID*/
      0x0, /*Board Ref*/
      0x0, /*Mem Offset*/
      0x0, /*Mem Length*/
      0x0, /*GMCR*/
      0x0 /*LMCR*/
   }/* END OF RESOURCE */
```

```
};   /*== End Resource Table ==*/

S_PROC_FILE_LIST F5_SystemFiles[] =
{ /* BEGIN FILES: */
   {"Board1:SiteA:Proc0",""},
   {"Board1:SiteB:Proc0",""},
   {"",""}  /* End of List Indicator */

}; /* END FILES */

#ifdef  __cplusplus
}
#endif

#endif /* SS_RESRCE_H */

/* End of file f5.h */
```

6.8. Using Handles to Access Systems and Resources

Handles are used in F5 ALIB functions to identify particular systems or system resources. All of the information needed to perform operations on an F5 system or resource is stored in its handle. Once a handle has been obtained for an object, it can be used as a unique identifier for that object in subsequent ALIB function calls. These handles remove you from the low-level details of an F5 system and simplify access to the system and its resources. Two types of handles are used in ALIB:

- **System Handles** – used to uniquely identify particular F5 systems. You can obtain a system handle by calling **F5_SystemOpen**. This system handle can then be used in subsequent calls to F5_GetHandle, F5_InstCallback, F5_SystemClose, and F5_SystemLoad.

- **Resource Handles** – uniquely identify hardware resources on an F5 system that can be written to and read from by your PC. You can obtain a handle to a resource by calling **F5_GetHandle**. This resource handle can then be used in subsequent calls to F5_Control, F5_InterruptProc, F5_Read, and F5_Write.

  Windows NT: Resource handles can also uniquely identify a block of host memory allocated by the **F5_AllocHostMem** function. This resource handle can then be used in subsequent calls to **F5_Read** and **F5_Write** functions.

6.9. Viewing Debug Messages *(Windows 95)*

Note: This section applies only to the Windows 95 platform.

The installation program adds entries to the registry which setup the operation mode of both the F5 Windows 95 Device Driver and the ALIB_W95. Under the key:

HKEY_LOCAL_MACHINE

SOFTWARE

SSP F5

are the values:

DrvMode	“Debug” or “Release” (default)
LibMode	“Debug” or “Release” (default)
Kernellocation	“<F5RootDirectory>\bin”

When either DrvMode or LibMode is set to “Debug”, debugging messages will be sent to a debugging window such as the supplied DBGMON utility. Open DBGMON in order to view the messages sent. You can open DBGMON by double-clicking on the **dbgmon.exe** file that’s in *<F5RootDirectory>\bin*.

The above values can be changed by running the Windows utility “regedit” from the Run command under the Start Menu.

Caution: Be very careful when editing the Registry.

7 Host Software Functions

This chapter describes the host ALIB C language functions. The functions are listed in alphabetical order and are summarized in the following table.

Table 7 F5 ALIB Host Functions

Function Name	Description
F5_AllocHostMem	Allocates a contiguous block of memory on the host.
F5_Control	Performs a control operation (for example, RESET) on an F5 system, board, or resource.
F5_ErrorMessage	Returns a text description for an error code.
F5_FreeHostMem	Frees a previously allocated contiguous block of memory on the host.
F5_GetHandle	Returns a handle to an available resource on an F5 system.
F5_InstCallback	Installs or removes an Interrupt Callback function for a specified system.
F5_InterruptProc	Interrupts a processor on an F5 board.
F5_Read	Reads from an F5 memory resource.
F5_SystemClose	De-allocates all resources currently allocated to an F5 board and closes the board.
F5_SystemLoad	Loads DSP code to all processors in a system.
F5_SystemOpen	Initializes and opens an F5 system.
F5_Write	Writes to an F5 memory resource.

F5_AllocHostMem (Windows 95)

Function Allocates a contiguous block of memory on the host.

Include File F5user.h

Syntax
```
F5API RESULT F5_AllocHostMem (UINT32 dwSize,
                              UINT32* pdwLinAddr,
                              UINT32* pdwPhysAddr);
```

Parameters

dwSize	The size (in bytes) of the memory block to allocate.
PdwLinAddr	The Linear (virtual) address to the allocated memory block.
PdwPhysAddr	The Physical address of the allocated memory block. This address should be directly accessible by the DSP code.

Returned Values

F5_RES_OK	The function completed successfully.
F5_RES_OS_NOT_ENOUGH_MEM	Your PC does not have sufficient memory to allocate the required space.
F5_RES_DRV_CANT_ALLOC_MEM	Not enough memory to allocate for the request.
F5_RES_DRV_NO_CONTIG_MEM	Not enough contiguous memory to allocate memory for the request.
F5_RES_DRV_CANT_LOC_MEM	The Driver cannot lock the memory requested.

Remarks This function can be used to allocate a block of PCI memory on the host that can be accessed by the **C4X_Read** and **C4X_Write** ALIB_C4x functions (see the **busmastr** example). *PCI Physical address must be referenced in bytes and not in 32 bit words.*

Example Code
```
..

static UINT32     hostArraySize = 0x0;
static UINT32     arrayLinAddr = 0x0;
static UINT32     arrayPhysAddr = 0x0;
..

/*--------------------------------------------------------------------*/
int main(int argc, char *argv[])
{
..

hostArraySize = strtoul(choice, NULL, 16);

   printf("\nAllocating host memory...     ");
   result = F5_AllocHostMem(hostArraySize*4, &arrayLinAddr,
                            &arrayPhysAddr);
..

}
```

F5_AllocHostMem (Windows NT)

Function Allocates a contiguous block of memory on the host.

Include File F5user.h

Syntax

```
F5API RESULT F5_AllocHostMem (UINT32 dwSize,
                              HF5_RESOURCE *phMemResc,
                              UINT32* pdwPhysAddr);
```

Parameters

dwSize	The size (in bytes) of the memory block to allocate.
phMemResc	The allocated host memory resource handle.
pdwPhysAddr	The physical address of the allocated memory block. This address should be directly accessible by the DSP code.

Returned Values

F5_RES_OK	The function completed successfully.
F5_RES_OS_NOT_ENOUGH_MEM	Your PC does not have sufficient memory to allocate the required space.
F5_RES_DRV_CANT_ALLOC_MEM	Not enough memory to allocate for the request.
F5_RES_DRV_NO_CONTIG_MEM	Not enough contiguous memory to allocate memory for the request.
F5_RES_DRV_CANT_LOC_MEM	The Driver cannot lock the memory requested.

Remarks This function can be used to allocate a block of PCI memory on the host that can be accessed by the **C4X_Read** and **C4X_Write** ALIB_C4x functions (see the **busmastr** example). *PCI Physical address must be referenced in bytes and not in 32 bit words.*

Example Code

```
..

static UINT32     hostArraySize = 0x0;
static UINT32     arrayLinAddr = 0x0;
static UINT32     arrayPhysAddr = 0x0;
..

/*--------------------------------------------------------------------*/
int main(int argc, char *argv[])
{
HF5_RESOURCE hMemResc;
..

hostArraySize = strtoul(choice, NULL, 16);

   printf("\nAllocating host memory...     ");
   result = F5_AllocHostMem(hostArraySize*4, &hMemResc, &ArrayPhysAddr);
..
}
```

F5_Control

Function Performs a control operation on an F5 system, board, or resource.

Include File F5user.h

Syntax

```
F5API RESULT F5_Control(HF5_RESOURCE hResc,
                        UINT32 Action,
                        UINT32 Flags,
                        void* pvValue);
```

Parameters

hResc Handle to the F5 resource (returned from **F5_GetHandle**) to control.

Action Action to be performed, specify either:

F5_CTL_RESET = to reset the specified resource.

F5_CTL_INTPROC = to cause the INT_NODE to interrupt a processor.

F5_CTL_GETBRD = to get a board's configuration data. The data is returned to the location specified by *pvValue*.

F5_CTL_RESET = This action resets the specified resource:

If hResc specifies a:	then the following is reset:
System	all boards in the System
Resource	only the parent board
Board	only the specified board

F5_CTL_INTPROC = This action causes the INT_NODE to interrupt a processor:

If hResc specifies a:	then the INT_NODE:
Board	interrupts the processor on Node x of the specified board. Use the Flags parameter (see below) to specify which node the processor resides on.
Resource (of Processor type)	interrupts that processor.

F5_CTL_GETBRD = This action gets a board's configuration data. The data is returned to the location specified by *pvValue*.

If *hResc* specifies a:	then this action retrieves configuration data for:
System	the Board identified by the BoardID in the *Flags* parameter.
Resource	the Parent Board.

Flags

- If *Action* = F5_CTL_GETBRD and *hResc* specifies a system, then *Flags* must specify the Board ID for which to retrieve configuration data. You can obtain the Board ID from the SDF file (Base Address of Board).
- If *Action* = F5_CTL_INTPROC and *hResc* specifies a board, then *Flags* must specify the Node ID (see the following table) on which the processor to interrupt resides.

pvValue Storage location for board configuration data. This parameter must be specified as HF5_BOARD type and must be cast to void.

Flag	Description
F5_FLAG_IP_NODEA	Interrupt the processor on Node A
F5_FLAG_IP_NODEB	Interrupt the processor on Node B
F5_FLAG_IP_NODEC	Interrupt the processor on Node C
F5_FLAG_IP_NODED	Interrupt the processor on Node D

Returned Values

F5_RES_OK	The function completed successfully.
F5_RES_LIB_LIBRARY_NOT_OPEN	DLL not initialized. A call to F5_SystemOpen should be made to initialize the library.
F5_RES_SYS_CORRUPT	An invalid system handle was specified.
F5_RES_RES_INVALID_RESC_TYPE	An invalid resource type was specified.

Remarks

Any driver errors (F5_RES_DEV_*) can also be returned. See *Appendix A: Status Codes* for a list of possible return codes.

Resetting a board will reset all processors on the board and any processors on boards that are connected via a JTAG chain.

You must reset an F5 board before you can load DSP code onto it.

Example Code

```
int main(int argc, char *argv[])
{
..
static HF5_SYSTEM   hSystem = NULL;
..

result = F5_SystemOpen(&hSystem, sdf, F5_FLAG_SO_SDFFILE);
..

result =  F5_Control((HF5_RESOURCE)hSystem, F5_CTL_RESET, NULL);
..

}
```

```
int main(int argc, char *argv[])
{
..

HF5_RESOURCE   hBoard,hProc;
..

result = F5_GetHandle(hSys, “Board1:SiteA:Proc1”, hProc);

result = F5_Control(hProc, F5_CTL_INTPROC, 0, (void *)0);
..

}
```

```
void main(void)
{
F5_BOARD   Board, *hBrd = &Board;
HF5_SYSTEM hSys;
..

result = F5_SystemOpen(&hSys, sdf, 0);
..

result = F5_Control((HF5_RESOURCE) hSys, F5_CTL_GETBRD, 0x123,
                    (void*)hBrd);
..

}
```

F5_ErrorMessage

Function Gets a text description for a given error code.

Include File F5user.h

Syntax `F5API STRING F5_ErrorMessage(RESULT dwError);`

Parameters *`dwError`* Return code for which to get a text description.

Returned Values See *Appendix A: Status Codes* for a list of strings returned.

Remarks The memory location used to store the string returned by this function is shared between all threads accessing the system. Therefore, you must copy the string returned from F5_ErrorMessage immediately into a local buffer. This can be done by using the following command:

```
char szBuffer[128];

..

strcpy (szBuffer, F5_ErrorMessage(…) );
```

where `szBuffer` is a locally declared buffer of characters.

Note that it is NOT sufficient to merely copy the char pointer returned by F5_ErrorMessage to a locally defined char pointer. The complete string must be copied using `strcpy`.

Example Code

```
int main(void)
{
..

result = F5_GetHandle(hSystem, "Board1:PlxLocalRegs", &hPlxLocRegs);
   if (F5_RES_OK != result)
   {
      printf("[FAILED] (%s)\n", F5_ErrorMessage(result));
      handle_error();
      return(-1);
   }
..

}
```

F5_FreeHostMem (Windows 95)

Function Frees a contiguous block of memory previously allocated with F5_AllocHostMem.

Include File F5user.h

Syntax
```
F5API RESULT F5_FreeHostMem (UINT32 dwSize,
                             UINT32 dwLinAddr,
                             UINT32 dwPhysAddr);
```

Parameters

dwSize	The size (in bytes) of the memory block to free.
dwLinAddr	The Linear address of the memory block to free.
dwPhysAddr	The Physical address of the allocated memory block to free.

Returned Values

F5_RES_OK	The function completed successfully.

Remarks Only a memory block that was previously allocated by the **F5_AllocHostMem** function can be freed by F5_FreeHostMem.

Example Code
```
..

static UINT32     hostArraySize = 0x0;
static UINT32     arrayLinAddr = 0x0;
static UINT32     arrayPhysAddr = 0x0;
..

/*--------------------------------------------------------------------*/
int main(int argc, char *argv[])
{
..

hostArraySize = strtoul(choice, NULL, 16);

   printf("\nAllocating host memory...     ");
   result = F5_AllocHostMem(hostArraySize*4, &arrayLinAddr,
&arrayPhysAddr);
..

result = F5_FreeHostMem(hostArraySize*4, arrayLinAddr, arrayPhysAddr);
..

}
```

F5_FreeHostMem (Windows NT)

Function Frees a contiguous block of memory previously allocated with F5_AllocHostMem.

Include File F5user.h

Syntax
```
F5API RESULT F5_FreeHostMem (UINT32 dwSize,
                             HF5_RESOURCE hMemResc,
                             UINT32 dwPhysAddr);
```

Parameters

dwSize	The size (in bytes) of the memory block to free.
hMemResc	The host memory resource describing the memory block to free.
dwPhysAddr	The Physical address of the allocated memory block to free.

Returned Values

F5_RES_OK	The function completed successfully.

Remarks Only a memory block that was previously allocated by the **F5_AllocHostMem** function can be freed by F5_FreeHostMem.

Example Code

```
..

static UINT32     hostArraySize = 0x0;
static UINT32     arrayLinAddr = 0x0;
static UINT32     arrayPhysAddr = 0x0;
..

/*--------------------------------------------------------------------*/
int main(int argc, char *argv[])
{
HF5_RESOURCE hAllocMem;
..

hostArraySize = strtoul(choice, NULL, 16);

   printf("\nAllocating host memory...     ");
   result = F5_AllocHostMem(hostArraySize*4, &hAllocMem, &arrayPhysAddr);
..

result = F5_FreeHostMem(hostArraySize*4, hAllocMem, arrayPhysAddr);
..

}
```

F5_GetHandle

Function Gets a handle to an available F5 board resource.

Include File F5user.h

Syntax
```
F5API RESULT F5_GetHandle(HF5_SYSTEM hSystem,
                          STRING lpFullRescName,
                          HF5_RESOURCE* phResc);
```

Parameters

hSystem	Handle to the F5 system (returned from **F5_SystemOpen**)
lpFullRescName	Character string identifying the resource to get a handle for. The resource can be either an F5 board, or a resource on an F5 board. See *Remarks*.
phResc	Pointer to F5 resource handle storage location.

Returned Value

F5_RES_OK	The command completed successfully.
F5_RES_WRONG_PARAMS	An invalid parameter was specified.
F5_RES_SYS_CORRUPT	The system handle is invalid.
F5_RES_RES_NOT_FOUND	The name of the resource was not found.

Remarks The string passed to *lpFullRescName* must contain the Full Resource Name of the resource. The Full Resource Name can be obtained from either the System Definition File (SDF) or Resource Definition File (RDF) file. For example, the Full Resource Name of Processor 0 on board 1 is: “Board1:SiteA:Proc0”. (See *sections 6.4* and *6.7* for more information about SDF and RDF files.)

The handle returned by this function contains all the necessary information to identify the resource and should be used to access the resource in subsequent ALIB_W95 function calls.

Example Code
```
int main(int argc, char *argv[])
{

   HF5_SYSTEM   hSystem = NULL;
   HF5_RESOURCE hSharedSRAM;
..

result = F5_SystemOpen(&hSystem, sdf, F5_FLAG_SO_SDFFILE);
..

result = F5_GetHandle(hSystem, "Board1:SharedRam", &hSharedSRAM);
..

}
```

F5_InstCallback

Function Installs or cancels an Interrupt Callback Function for the F5 System specified.

Include File F5user.h

Syntax
```
F5API RESULT F5_InstCallback(HF5_SYSTEM hSystem,
                             IRQFUNC pFuncCB);
```

Parameters

hSystem	Handle to the F5 system (returned from **F5_SystemOpen**)
pFuncCB	A pointer to the callback function to be installed. Specify "NULL" if you're canceling a callback function. See *Remarks* for a declaration of FuncCB.

Returned Values

F5_RES_OK	The function completed successfully.
F5_RES_SYS_CORRUPT	An invalid system handle was specified.
F5_RES_LIB_NOT_OPEN	The DLL has not been initialized. A call to F5_SystemOpen should be made.
F5_RES_DEV_TAKEN	The specified board is already being used by another process (thread).

Remarks F5_InstCallback is used to specify which function (the "callback function") will automatically execute whenever an interrupt is generated by any board on the system (specified by *hSystem*).

Only a thread which previously installed a Callback function can change it later for the same *hSystem*. However, any thread can cancel a Callback function for a given *hSystem*.

Make sure you cancel an installed callback before your program ends.

Windows 95: Only a single callback function can be installed on a system at any time. If another callback is already installed, it will be removed if possible (possible if both callbacks are installed by the same thread) and the new callback will be installed. If this is not possible, the old callback will remain and an error will be returned.

Windows NT: Only a single callback function can be installed on a system at any time. However, you can have open at the same time two or more systems which use the same System Definition File (SDF). Each of these systems can install its callback function. Each of the callback functions will be called when an interrupt on the physical board happens. Before installing a callback function in a system, your code must uninstall any previously installed (in that system) callback function.

Whenever the particular board belonging to the F5 system asserts an interrupt request, the callback function automatically executes. The callback function must be declared as follows:

Windows 95: void <*function_name*>(DWORD dwBoardID, DWORD dwIntReason, DWORD dwIntData)

Windows NT: void __stdcall <*function_name*>(DWORD dwBoardID, DWORD dwIntReason, DWORD dwIntData)

where the parameters can have the following values:

Parameter	Value / Description
dwBoardID	ID of the board which generated the interrupt.
dwIntReason	1 - the reason for the interrupt is “Local to PCI doorbell”. 2 - the interrupt was generated by processor A, B, C or D.
dwIntData	If dwIntReason = 1, then the content of the doorbell. If dwIntReason = 2, then the coded value which states which node generated the interrupt: 1 Node A 2 Node B 4 Node C 8 Node D

Any driver errors (F5_RES_DEV_*) can also be returned. See *Appendix A: Status Codes* for a list of possible return codes.

Example Code for Windows 95:

```
/*--------------------------------------------------------------------*/
void BoardIRQ(DWORD boardID, DWORD intReason, DWORD intData)
{
   int i;


   /* local to PCI doorbell
    */
   if (1 == intReason)
   {
      printf(" local to PCI doorbell (board 0x%x) received\n", boardID);
      int_received++;
   }

   /* node to PCI interrupts
    */
   if (2 == intReason)
   {
      for (i = 0; i < 4; i++)
      {
         if (intData & (1 << i))
         {
            printf(" node %c (board 0x%x) -> host received\n",
                   ('A' + i), boardID);
            int_received++;
         }
      }
   }
}

/*--------------------------------------------------------------------*/
int main(int argc, char *argv[])
{
..

HF5_SYSTEM      hSystem = NULL;
..

result = F5_SystemOpen(&hSystem, sdf, F5_FLAG_SO_SDFFILE);
..

result = F5_InstCallback(hSystem, BoardIRQ);
..

result = F5_InstCallback(hSystem, NULL);
..

}
```

Example Code for Windows NT:

```
/*--------------------------------------------------------------------*/
void __stdcall BoardIRQ(DWORD boardID, DWORD intReason, DWORD intData)
{
   int i;


   /* local to PCI doorbell
    */
   if (1 == intReason)
   {
      printf(" local to PCI doorbell (board 0x%x) received\n", boardID);
      int_received++;
   }

   /* node to PCI interrupts
    */
   if (2 == intReason)
   {
      for (i = 0; i < 4; i++)
      {
         if (intData & (1 << i))
         {
            printf(" node %c (board 0x%x) -> host received\n",
                   ('A' + i), boardID);
            int_received++;
         }
      }
   }
}

/*--------------------------------------------------------------------*/
int main(int argc, char *argv[])
{
..

HF5_SYSTEM      hSystem = NULL;
..

result = F5_SystemOpen(&hSystem, sdf, F5_FLAG_SO_SDFFILE);
..

result = F5_InstCallback(hSystem, BoardIRQ);
..

result = F5_InstCallback(hSystem, NULL);
..

}
```

F5_InterruptProc

Function Interrupts a Processor on an F5 board.

Include File F5user.h

Syntax

```
F5API RESULT F5_InterruptProc(HF5_RESOURCE hResc,
                              UINT32 Flags);
```

Parameters

hResc	Handle to the F5 resource (returned from **F5_GetHandle**) to interrupt. See *Remarks*.
Flags	The node on which the processor to interrupt resides (see table below). Use only when *hResc* specifies a Board.

Flag	Description
F5_FLAG_IP_NODEA	Interrupt the processor on Node A
F5_FLAG_IP_NODEB	Interrupt the processor on Node B
F5_FLAG_IP_NODEC	Interrupt the processor on Node C
F5_FLAG_IP_NODED	Interrupt the processor on Node D

Returned Values

F5_RES_OK	The function completed successfully.

Remarks

If hResc specifies a:	then:
Board	the processor on Node *x* of the specified board will be interrupted.
Processor type resource	that processor will be interrupted and the *Flags* parameter will be ignored.

You can also use the **F5_Control** function to interrupt a processor on the F5 board.

Example Code

```
..
RESULT F5_InterruptProc(hBoard, F5_FLAG_IP_NODEA);
..
```

F5_Read

Function Reads a specified number of 32-bit words from a specified memory resource.

Include File F5user.h

Syntax

```
F5API RESULT F5_Read(HF5_RESOURCE hResc,
                     UINT32* pdwDest,
                     UINT32* pdwSrc,
                     UINT32 dwLength,
                     UINT32 dwFlags);
```

Parameters

hResc Handle to the F5 resource (returned from **F5_GetHandle**) to read data from.

Windows NT: This can also be a handle to a block of host memory allocated by **F5_AllocHostMem**.

pdwDest Pointer to buffer in which to store the read data.

pdwSrc Offset from the base of the resource to be read.

dwLength Number of 32-bit words to read.

dwFlags

Windows 95:	**Windows NT:**
Specify `0` (zero).	Specify one of the following: • `0` = read from an F5 board's resource (via the device driver) • `F5_FLAG_RW_DIRECT` = read directly from the User Virtual Address • `F5_FLAG_RW_HOST_MEM` = read from an allocated host memory block

Returned Values

`F5_RES_OK`	The function completed successfully.
`F5_RES_WRONG_PARAM`	An invalid parameter was specified.
`F5_RES_RES_INVALID_RESC_TYPE`	An invalid resource type was specified.

Remarks None

Example Code

```
/*--------------------------------------------------------------------------*/
int main(int argc, char *argv[])
 {
..

HF5_RESOURCE   hMailbox;
UINT32         sharedArrayOffset;
UINT32         mboxOffset = MAILBOX0_OFFSET;

..

result = F5_Read(hMailbox, &sharedArrayOffset, (UINT32*)mboxOffset, 1,
0);
..

}
```

F5_SystemClose

Function De-allocates all resources currently allocated to the specified F5 system.

Include File F5user.h

Syntax `F5API RESULT F5_SystemClose(HF5_SYSTEM hSystem);`

Parameters

phSystem	Pointer to handle of the F5 System (returned from **F5_SystemOpen**) to close.

Returned Values

F5_RES_OK	The function completed successfully.
F5_RES_SYS_CORRUPT	An invalid system handle was specified.
F5_RES_LIB_LIBRARY_NOT_OPEN	The ALIB has not been initialized. A call to F5_SystemOpen should be made to initialize the library.

Remarks Any driver errors (F5_RES_DEV_*) can also be returned. See *Appendix A: Status Codes* for a list of possible error codes.

Example Code

```
int main(int argc, char *argv[])
{
..

HF5_SYSTEM      hSystem = NULL;
..

result = F5_SystemOpen(&hSystem, sdf, F5_FLAG_SO_SDFFILE);
..

result = F5_SystemClose(hSystem);
..

}
```

F5_SystemLoad

Function Loads an application into the DSP of a specified F5 system and runs the code.

Include File F5user.h

Syntax

```
F5API RESULT F5_SystemLoad(HF5_SYSTEM hSystem,
                           STRING lpLoadFile,
                           UINT32 Flags);
```

Parameters

hSystem	Handle to the F5 system (returned from **F5_SystemOpen**) to load the application to.
lpLoadFile	Character string identifying the DSP code to load (the code must be in COFF format). Specify either: • a Load Definition File (LDF) name. For example "f5_1.ldf". An LDF must be specified if an SDF was used to open the system. (See *section 6.6* for more information about LDFs.); or • a Processor File List name taken from a Resource Definition File (RDF). This must be specified if a Resource Table name was used to open the system. See *Remarks*.
Flags	NO_FLAGS (this parameter is reserved for future functionality)

Returned Values

F5_RES_OK	The command completed successfully.
F5_RES_WRONG_PARAMS	An invalid *hSystem* or *phResource* value was specified.
F5_RES_LIB_LIBRARY_NOT_OPEN	DLL has not been initialized.
F5_RES_SYS_CORRUPT	A bad system handle was specified.
F5_RES_SYS_NO_RESET	The board or system was not reset.

Remarks The application will only be loaded if the F5 board is in RESET mode. Use **F5_Control** to reset the F5 board.

All DSP processors in the specified system will be loaded with the DSP application code.

A Processor File List name (used for *lpLoadFile)* is obtained from the Processor File List section of an RDF. This is the last section of an RDF and is identified as follows:

```
S_PROC_FILE_LIST ProcessorFileListName[]=
```

See *section 6.7.2* for an example of an RDF.

Note: Do not change the order of the processor names in the File List.

This order is calculated automatically by Resource Composer when the RDF is generated. The order of the processors in the File List indicates the order in which processors are to be loaded, and depends on the hardware connections between the processors (as defined in the SDF).

Example Code

```
int main(int argc, char *argv[])
{
..

   HF5_SYSTEM     hSystem = NULL;
   char           sdf[] = "..\\..\\include\\f5.sdf";
   char           ldf[] = "intrupts.ldf";

..

result = F5_SystemOpen(&hSystem, sdf, F5_FLAG_SO_SDFFILE);
..

result =  F5_Control((HF5_RESOURCE)hSystem, F5_CTL_RESET, 0, NULL);
..

result = F5_SystemLoad(hSystem, ldf, 0);
..

}
```

```
int main(int argc, char *argv[])
{
..

   HF5_SYSTEM     hSystem = NULL;

..

result = F5_SystemOpen(&hSystem, (STRING)F5_System, F5_FLAG_SO_RESCLIST);
..

result =  F5_Control((HF5_RESOURCE)hSystem, F5_CTL_RESET, 0, NULL);
..

result = F5_SystemLoad(hSystem, (STRING)F5_SystemFiles, 0);
..

}
```

F5_SystemOpen

Function Initializes an F5 system.

Include File F5user.h

Syntax

```
F5API RESULT F5_SystemOpen(HF5_SYSTEM *hSystem,
                           STRING lpCfgFile,
                           UINT32 Flags);
```

Parameters

hSystem — Pointer to F5 system handle storage location

lpCfgFile — Initialization information for F5 system. This can be either:

- a System Definition File (SDF) name, including the path. For example "f5_1.sdf" (see *section 6.4* for more information about SDFs), or
- a Resource Table name obtained from an Resource Definition File (RDF), for standalone applications only. See *Remarks*.

See *Flags* below.

Flags — Indicates whether the string specified in *lpCfgFile* is an SDF filename or a Resource Table name from an RDF (see the following table).

Flag	Value	Description
F5_FLAG_SO_SDFFILE	0x01	*lpCfgFile* specifies the filename and path of an SDF.
F5_FLAG_SO_RESCLIST	0x02	*lpCfgFile* specifies a Resource Table name.

Returned Value

F5_RES_OK	The function completed successfully.
F5_RES_TOO_MANY_SYSTEMS	There are too many systems open. The maximum number of boards that can be open at one time is 20.
F5_RES_WRONG_PARAMS	An invalid parameter was specified.
F5_RES_OS_NOT_ENOUGH_MEM	There is not enough memory to execute this function.

Remarks Any driver errors (F5_RES_DEV_*) can also be returned. See *Appendix A: Status Codes* for a list of possible error codes.

The handle returned by this function is used to identify the F5 system in future calls to **F5_Control**, **F5_GetHandle**, and **F5_SystemLoad**.

A Resource Table name can be obtained from the Resource Table section of an RDF. This is the first section of an RDF and is identified as follows:

`F5_RESOURCE` *`ResourceTableName`*`[] =`

See *section 6.7.2* for an example of an RDF.

Example Code

```
int main(int argc, char *argv[])
{
..

   HF5_SYSTEM     hSystem = NULL;
   char           sdf[] = "..\\..\\include\\f5.sdf";
   char           ldf[] = "intrupts.ldf";

..

result = F5_SystemOpen(&hSystem, sdf, F5_FLAG_SO_SDFFILE);
..

}
```

```
int main(int argc, char *argv[])
{
..

   HF5_SYSTEM     hSystem = NULL;

..

result = F5_SystemOpen(&hSystem, (STRING)F5_System, F5_FLAG_SO_RESCLIST);
..

}
```

F5_Write

Function Writes a specified number of 32-bit words to the specified memory resource.

Include File F5user.h

Syntax

```
F5API RESULT F5_Write(HF5_RESOURCE hResc,
                      UINT32* pdwDest,
                      UINT32* pdwSrc,
                      UINT32 dwLength,
                      UINT32 dwFlags);
```

Parameters

hResc	Handle to the F5 resource (returned from **F5_GetHandle**) to write data to. **Windows NT:** This can also be a handle to a block of host memory allocated by **F5_AllocHostMem**.
pdwDest	Offset from the base of the resource to write to.
pdwSrc	Pointer to the buffer in which to get the data to be transmitted
dwLength	Number of 32-bit words to transmit
dwFlags	**Windows 95:** Specify `0` (zero). **Windows NT:** Specify one of the following: • `0` = write to an F5 board's resource (via the device driver) • `F5_FLAG_RW_DIRECT` = write directly to the User Virtual Address • `F5_FLAG_RW_HOST_MEM` = write to an allocated host memory block

Returned Values

`F5_RES_OK`	The function completed successfully.
`F5_RES_WRONG_PARAM`	An invalid parameter was specified.
`F5_RES_RES_INVAL_RESC_TYPE`	An invalid resource type was specified.

Remarks None

Example Code

```
/*--------------------------------------------------------------------*/
int main(int argc, char *argv[])
 {
..

HF5_RESOURCE   hSharedSRAM;
UINT32         sharedArrayOffset;
UINT32         locArray[SHARED_ARRAY_SIZE];
..
result = F5_Write(hSharedSRAM, (UINT32*)sharedArrayOffset, locArray,
                  SHARED_ARRAY_SIZE, 0);

..
}
```

8 Development of DSP Applications

Spectrum provides a high level application library (ALIB_C4x) to help you develop DSP code for the F5 board. ALIB_C4x provides your DSP applications with functions to configure and transfer data to F5 resources. This chapter describes some of these functions and discusses important C4x development requirements when using the F5 ALIB_C4x.

8.1. Calling F5 ALIB_C4x Functions

The first function called in your DSP code must be **C4X_Open** to initialize the C4x library. The last function called in your host code must be **C4X_Close** to close the C4x library.

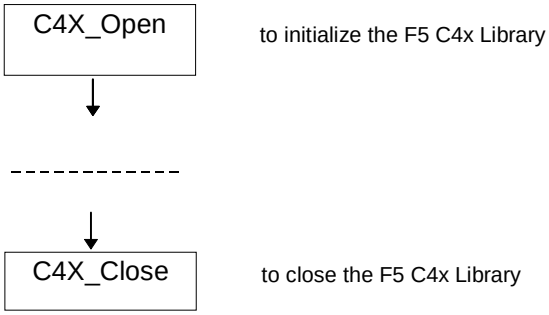


Figure 9 Order of DSP Function Calls

8.2. Including F5 ALIB_C4x Functionality

In order for your application to use the F5 ALIB_C4x functions, you must:

- Include the **f5_c4x.h** file in your DSP code. This file contains all the required F5 function prototypes and defines and can be found in the *<F5RootDirectory>\include* directory.
- Link your code with the **f5_c4x.lib** library file. A sample linker command file, **c4xlink.cmd**, has been provided in the *<F5RootDirectory>\include* directory and is presented in *Appendix B: Sample Linker Command File*. Refer to the *Dakar F5 Carrier Board Technical Reference* manual for detailed memory maps of the C4x resources.

Note: Certain memory locations are reserved by the F5 library for code download (see the **c4xlink.cmd** file in *Appendix B: Sample Linker Command File* for more details).

The F5 C4x application library has been compiled with the following options:

- Small memory model
- Stack-based model
- Optimization level 2

8.3. Transferring Data Between DSPs Shared SRAM and the PCI Bus

The **C4X_Read** and **C4X_Write** ALIB_C4x functions allow you to perform optimized data transfers between C4x resources. The two types of transfers that can be performed are:

- Any node's near memory to/from Far Global (shared) SRAM; and
- PCI to/from Far Global SRAM, initiated by node A via PCI9060/9080's DMA.

The transfers are shown in the following two figures.

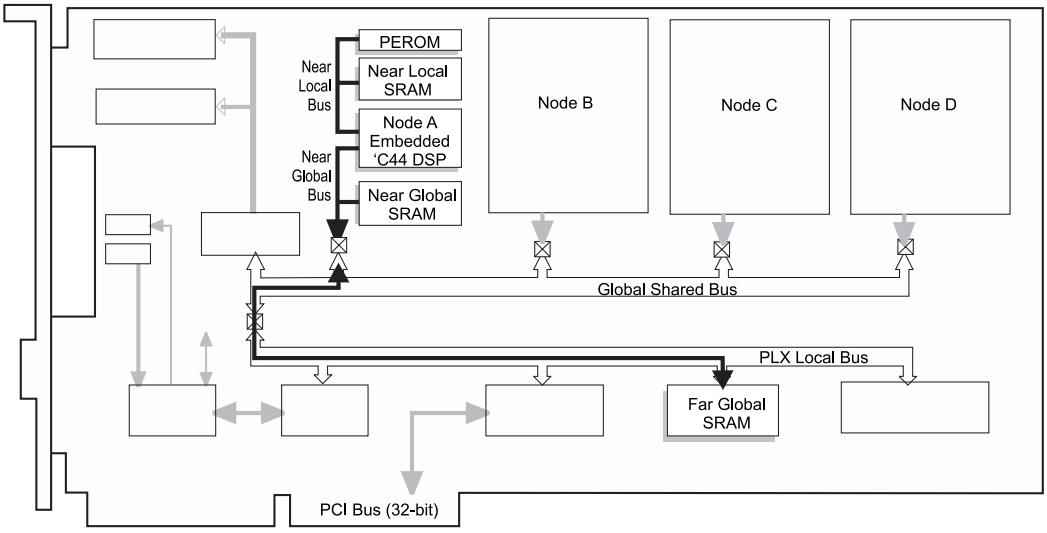


Figure 10 Near Memory /Far Global SRAM Transfers

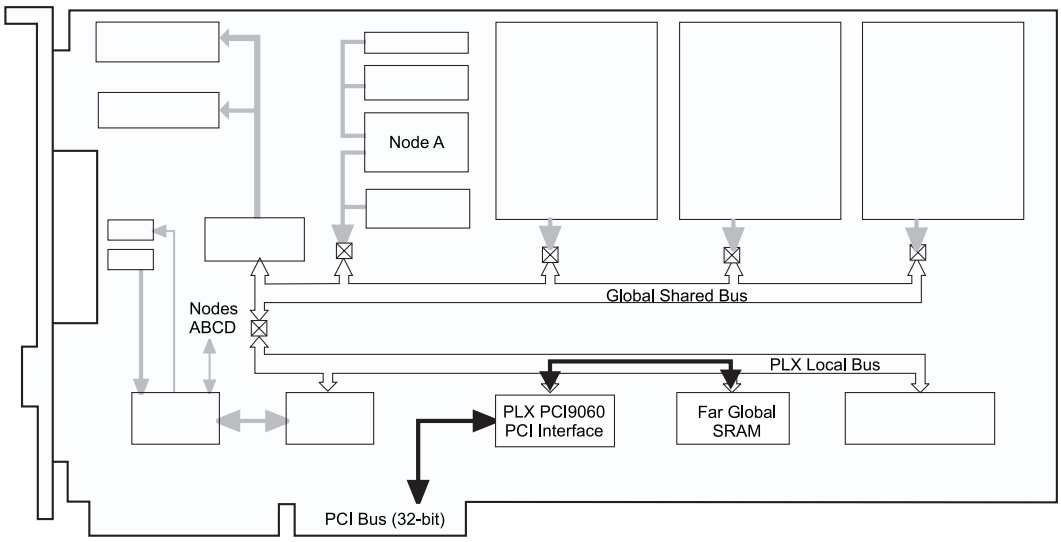


Figure 11 PCI/Far Global SRAM Transfers

Note: The Intel 430FX chipset does not support DMA Bus Mastering. If your computer is equipped with this chipset, the F5 Carrier Board cannot initiate a DMA transfer to the PCI bus as a PCI master.

8.4. Running F5 DSP Code via a Debugger

This section describes the software setup required when using Texas Instruments' standard TMS320C4x debug monitor. If you're using a third party debug monitor, refer to that product's documentation.

Note: Texas Instruments' TMS320C4x debug monitor is optional and can be ordered through Spectrum.

For details on the JTAG hardware interface and connecting to JTAG, refer to the *Dakar F5 Carrier Board Technical Reference* manual.

There are two files required for TI debugging: **board.cfg** and **init.cmd**.

8.4.1. board.cfg

The **board.cfg** is a configuration file that tells your debugging software which processors are in your system's JTAG scan chain and their order in the chain. A sample **board.cfg** (shown below) is provided by Spectrum when you order Texas Instruments' TMS320C4x debug monitor. This sample **board.cfg** file defines a single F5 board with its embedded node A and a single populated TIM-40 site. Unpopulated sites are commented out using a semicolon ";". Edit this file, uncommenting the TIM-40 sites that are populated in your system.

```
;"CPU_D"         TI320C4x                ; TIM-40 site D
;"CPU_C"         TI320C4x                ; TIM-40 site C
 "CPU_B"         TI320C4x                ; TIM-40 site B
 "CPU_A"         TI320C4x                ; Embedded node A
```

Note: Processors listed in the **board.cfg** file are listed in reverse order than they actually occur in the JTAG chain. The last board listed in the **board.cfg** file should be the first board in the JTAG chain.

You must generate a **board.dat** file each time you edit board.cfg in order for your changes to take effect. To generate the board.dat file, run **composer.exe**.

8.4.2. init.cmd

The **init.cmd** file describes all memory mapped C4x resources. The following init.cmd file is provided by Spectrum when you order Texas Instruments' TMS320C4x debug monitor and describes the embedded C44 resources.

Edit this file to properly reflect the resources for each node in your system (refer to the *F5 Carrier Board Technical Reference* manual for information on the board's memory resources).

```
mr
ma      0x00000000,0x800, rom         ; 2K EPROM LOCAL, STRB0
ma      0x002ff800,0x400, ram         ; INTERNAL BLK 0
ma      0x002ffc00,0x400, ram         ; INTERNAL BLK 1
;
ma      0x00300000,0x40000, ram       ; LOCAL SRAM 128K
ma      0x80000000,0x40000, ram       ; NEAR GLOBAL SRAM 128K
ma      0xc0300000,0x80000, ram       ; FAR GLOBAL SRAM 512K
ma      0xc0230000,0xD, ram           ; IRQ Registers
;
ma      0x40000000,0x8000, ram        ; Local PEROM 256K
ma      0x70000000,0x8000, ram        ; PEROM ID
;
ma      0xC0000000,0x10000, ram       ; DSP~LINK3 A
ma      0xC0010000,0x10000, ram       ; DSP~LINK3 B
ma      0xC0020000,0x10000, ram       ; DSP~LINK3 C
ma      0xC0030000,0x10000, ram       ; DSP~LINK3 D
ma      0xC0040000,0x10000, ram       ; DSP~LINK3 reset registers

; Embedded C44 only
;
ma      0xc0200000,0x5, ram           ; ARB Registers
ma      0xc0220000,0x4C, ram          ; PCI9060/9080 Registers

; C4x specific
;
ma      0x00100000,1, ram             ; global port control
ma      0x00100004,1, ram             ; local  port control
ma      0x00100020,9, ram             ; Timer 0
ma      0x00100030,9, ram             ; Timer 1
ma      0x00100040,3, ram             ; comm port 0
ma      0x00100050,3, ram             ; comm port 1
ma      0x00100060,3, ram             ; comm port 2
ma      0x00100070,3, ram             ; comm port 3
ma      0x00100080,3, ram             ; comm port 4
ma      0x00100090,3, ram             ; comm port 5
ma      0x001000A0,9, ram             ; DMA 0
ma      0x001000B0,9, ram             ; DMA 1
ma      0x001000C0,9, ram             ; DMA 2
ma      0x001000D0,9, ram             ; DMA 3
ma      0x001000E0,9, ram             ; DMA 4
ma      0x001000F0,9, ram             ; DMA 5
dasm pc

e *0x100000=0x3D840000   ; GMCR
e *0x100004=0x3D840000   ; LMCR

; user specific
;
alias exit, "runf;quit"
```

```
alias q, "quit"
alias n, "next"
alias s, "step"
alias gmcr, "? *0x100000"
alias lmcr, "? *0x100004"
```

Note: When loading your application via JTAG, the Global Memory Control Register (GMCR) and Local Memory Control Register (LMCR) have to be set in the **init.cmd** file in order for your program to load and for the processors to access their memory correctly. You'll need a separate **init.cmd** file for each type of TIM module. Refer to the documentation provided with your TIM modules for the correct GMCR and LMCR values.

Refer to the *TMS320C4x C Source Debugger User's Guide* for further details on using TI's TMS320C4x debug monitor.

Note: If you are using an external Test Bus Controller (TBC) via the JTAG IN connector, you cannot use the F5's TBC at the same time. The F5 board must be reset when switching between on-board and off-board TBCs for the change to take effect.

8.4.3. Testing the Software Setup

You can test the software setup of your debug monitor by running the **blink.out** executable file that's found in your *<F5RootDirectory>\examples\dspload* directory. Load and run this file via your debugger. The on-board LED should blink if the code is running on the embedded C44.

9 DSP Software Functions

This chapter describes the high-level ALIB_C4x “C” language functions provided with the F5. The functions are listed in alphabetical order and are summarized in the following table.

Table 8 F5 ALIB DSP Functions

Function Name	Description
C4X_Close	Closes the F5 C4x Application Library (ALIB_C4x).
C4X_Control	Performs a control operation on an F5 resource.
C4X_Open	Initializes and opens the F5 C4x Application Library (ALIB_C4x).
C4X_Read	Reads data from an F5 resource.
C4X_Write	Writes data to an F5 resource.

C4X_Close

Function Closes the F5 C4x application library.

Include File f5_c4x.h

Syntax
```
RESULT C4X_Close(void);
```

Parameters None

Returned Values

`NO_ERROR`	The function completed successfully.

Remarks None

Example Code
```
RESULT result;
..
result = C4X_Close();
..
```

C4X_Control

Function Performs a control operation on an F5 resource.

Include File f5_c4x.h

Syntax

```
RESULT C4X_Control(RESOURCE Resource,
                   UINT32 Operation,
                   UINT32 Flags,
                   void *Value);
```

Parameters

Resource	The F5 resource on which to perform the control operation, either: • C4X_NODE; • SHARED_SRAM; • PCI, INTERRUPTS; or • F5_REGISTERS.
Operation	The control operation to perform. The types of operations that can be performed depend on the Resource specified (see the table on the following page).
Flags	The Operation specific flag (see the table on the following page).
**Value*	Pointer to memory location for returned results.

Returned Values

NO_ERROR	The function completed successfully.
ILLEGAL_RESOURCE	The resource specified is unknown.
ILLEGAL_OPERATION	Unknown resource/operation combination or the operation cannot be performed for the current node.
ILLEGAL_FLAGS	The value specified is invalid.

Example Code

```
RESULT  result;
NODE_ID node_id;
..
result = C4X_Open(SHARED_SRAM_512K);
..
result = C4X_Control(C4X_NODE, GET_NODE_ID, 0, &node_id);
..
```

Resource	Operations	Description	Flag	Value
C4X_NODE	GET_NODE_ID	Return the calling Node's ID.	N/A	Set to node ID
	SET_NODE_CONFIG	De-asserts the /CONFIG line.	N/A	N/A
	SET_DMA_CHANNEL	Set the C4x DMA channel for C4x Read/Write transfers.	Channel: 0 to 5	N/A
	GET_DMA_CHANNEL	Return the current C4x DMA channel for C4x Read/Write DMA transfers.	N/A	Set to C4x DMA channel
PCI	GET_MAILBOX	Get the value of the PCI9060/9080 mailbox.	Mailbox: 0 to 7	Set to value of mailbox
	SET_MAILBOX	Set the value of the PCI mailbox.	Mailbox: 0 to 7	Value to be written to mailbox
	SET_DOORBELL	Set the value of PCI Local to PCI doorbell.	Value to be written to doorbell	N/A
	SET_DMA_CHANNEL	Set the PCI9060/9080 DMA channel for C4x Read/Write transfers.	Channel: 0 or 1	N/A
	GET_DMA_CHANNEL	Return the current PCI9060/9080 DMA channel for C4x Read/Write DMA transfers.	N/A	Set to PCI9060/9080 DMA channel
INTERRUPTS	GET_INT_SOURCES	Get all current interrupt sources for the given IIOF pin.	IIOF pin: 0 to 3	ORed interrupt sources.
	CLEAR_INT	Clear a specified interrupt.	Interrupt to clear	N/A
	ASSERT_INT	Assert a specified interrupt.	Interrupt to assert	N/A
	ENABLE_PCI9060/9080INT	Enable/disable a specified PCI9060/9080 interrupt.	Interrrupt	0 disable 1 enable
F5_REGISTERS	GET_LATENCY_TIMER	Get the current value of F5 latency count.	N/A	Set to current latency
	SET_LATENCY_TIMER	Set the value of F5 latency count.	Latency: from 0 to 31	N/A
	LED_SET	Set the on-board LED on or off.	0 off 1 on	N/A
	LED_TOGGLE	Toggle the state of the on-board LED.	N/A	N/A

C4X_Open

Function Initializes the F5 C4x Application Library.

Include File f5_c4x.h

Syntax `RESULT C4X_Open(UINT32 Flags);`

Parameters *Flags* F5 hardware configuration (see table below).

Flag	Description
NO_GLOBAL_BUS	The calling node has no Global Bus access. If this flag is specified, it must be ORed with *NODE_ID* in order to specify the calling node.
NODE_ID	*NODE_ID* identifies the calling node when there is no Global Bus access. *NODE_ID* can be either: NODE_A, NODE_B, NODE_C, or NODE_D.
	NODE_ID should be ORed with the NO_GLOBAL_BUS flag. For example: NO_GLOBAL_BUS | NODE_A
SHARED_SRAM_128K	F5 memory size is 128K x 32-bit shared SRAM
SHARED_SRAM_512K	F5 memory size is 512K x 32-bit shared SRAM
NO_CONFIG_ASSERT	Prevents this function from asserting /CONFIG

Returned Values

NO_ERROR	The function completed successfully.
ILLEGAL_FLAGS	The flag specified is unknown to the current node.

Remarks This function must be called prior to calling any other C4x Application Library function.

Example Code The embedded C44 has access to the C4x global bus:

```
RESULT  result;
..
result = C4X_Open(SHARED_SRAM_512K);
..
```

For a C4x on a TIM-40 module without global bus connector:

```
RESULT  result;
..
result = C4X_Open(SHARED_SRAM_512K | NO_GLOBAL_BUS | NODE_B);
..
```

C4X_Read

Function Reads a 32-bit data block from a specified F5 resource.

Include File f5_c4x.h

Syntax

```
RESULT C4X_Read(RESOURCE Resource,
                UINT32 *Dest,
                UINT32 *Src,
                UINT32 Length,
                UINT32 Flags);
```

Parameters

Resource The F5 resource from which to read data, either:

- SHARED_SRAM, or
- PCI (**use only if node A is the calling processor**)

**Dest* The destination start address to place the data read. The destination depends on the resource specified:

Resource	Destination
SHARED_SRAM	Near Local or Near Global SRAM
PCI	Far Global SRAM

**Src* The source start address of the data to read. The source depends on the resource specified:

Resource	Source
SHARED_SRAM	Far Global SRAM
PCI	PCI physical address (see *Remarks*).

Length Length of the block of data to transfer, in 32-bit words.

Flags Transfer mode (see table below).

Flag	Description
DMA_ENABLE	Use DMA to perform the data transfer. This flag must be specified if performing a PCI to Shared SRAM transfer.
DMA_SYNC	Perform a synchronous DMA transfer; the function will only return once the DMA transfer is complete. This flag should be ORed with the DMA_ENABLE flag. For example: DMA_ENABLE | DMA_SYNC
STATIC_DST	Keep the destination address static.
STATIC_SRC	Keep the source address static.

Returned Values

NO_ERROR	The function completed successfully.
ILLEGAL_RESOURCE	The resource specified is unknown to the current node.
ILLEGAL_OPERATION	A PCI to Shared SRAM transfer was specified, but DMA was not enabled. Ensure that DMA_ENABLE is specified for this type of transfer.
ILLEGAL_ADDRESS	Destination address/block-length is out of bounds.

Remarks

If you specify the DMA_ENABLE flag, a DMA transfer will be performed using the currently selected DMA channel. Use C4X_Control to set the DMA channel. In addition, if you select SHARED_SRAM as the resource, a C4X_NODE DMA transfer will be performed, and if you select PCI as the resource, a PCI9060/9080 DMA transfer will be performed. If the DMA_ENABLE flag is **not** specified, an optimized block transfer will be performed.

If the *Resource* is PCI, and you are reading from the host, you can get the PCI physical address by calling **F5_AllocHostMem.** (See the **busmastr** example for an illustration of PCI bus mastering from the embedded node A via the PCI9060/9080's DMA.)

The Intel 430FX chipset does not support DMA Bus Mastering. If your computer is equipped with this chipset, the F5 Carrier Board cannot initiate a DMA transfer to the PCI bus as a PCI master.

Example Code

Far Global SRAM to Internal RAM transfer:

```
RESULT  result;
..
result = C4X_Read(SHARED_SRAM, (UINT32 *)0x2ff800,
                  (UINT32 *)0xC0301000, 0x10, NO_FLAGS);
..
```

Far Global SRAM to COMM port 0:

```
RESULT  result;
..
result = C4X_Read(SHARED_SRAM, (UINT32 *)0x100042,
                  (UINT32 *)0xC0301000, 0x10, STATIC_DST);
..
```

PCIbus to Far Global SRAM (node A only):

```
RESULT  result;
..
result = C4X_Read(PCI, (UINT32 *)0xC0301000, pci_addr, 0x10,
                  DMA_ENABLE | DMA_SYNC);
..
```

C4X_Write

Function Writes a 32-bit data block to a specified resource.

Include File f5_c4x.h

Syntax

```
RESULT C4X_Write(RESOURCE Resource,
                 UINT32 *Dest,
                 UINT32 *Src,
                 UINT32 Length,
                 UINT32 Flags);
```

Parameters

Resource — The F5 resource to write data to, either:

- SHARED_SRAM, or
- PCI **(use only if node A is the calling processor)**

**Dest* — The destination start address to place the data read. The destination depends on the resource specified:

Resource	Destination
SHARED_SRAM	Far Global SRAM
PCI	PCI physical address (see *Remarks*)

**Src* — The source start address of the data to transfer. The source depends on the resource specified:

Resource	Source
SHARED_SRAM	Near Local or Near Global SRAM
PCI	Far Global SRAM

Length — length of block, in 32-bit words

Flags — Transfer flag (see table below).

Flag	Description
DMA_ENABLE	Use DMA to perform the data transfer. This flag must be specified if performing a Shared SRAM to PCI transfer.
DMA_SYNC	Wait for DMA to complete.
STATIC_DST	Keep the destination address static.
STATIC_SRC	Keep the source address static.

Returned Values

`NO_ERROR`	The function completed successfully.
`ILLEGAL_RESOURCE`	The resource specified is unknown to the current node.
`ILLEGAL_OPERATION`	A Shared SRAM to PCI transfer was specified, but DMA was not enabled. Ensure that `DMA_ENABLE` is specified for this type of transfer.
`ILLEGAL_ADDRESS`	Destination address/block-length out of bounds.

Remarks

If you specify the `DMA_ENABLE` flag, a DMA transfer will be performed using the currently selected DMA channel. Use **C4X_Control** to set the DMA channel. In addition, if you select `SHARED_SRAM` as the resource, a C4X_NODE DMA transfer will be performed, and if you select `PCI` as the resource, a PCI9060/9080DMA transfer will be performed. If the `DMA_ENABLE` flag is **not** specified, an optimized block transfer will be performed.

If the *`Resource`* is PCI, and you are writing to the host, you can get the PCI physical address by calling **F5_AllocHostMem.** (See the **busmastr** example for an illustration of PCI bus mastering from the embedded node A via the PCI9060/9080's DMA.)

Direct PCIbus mastering is **not** supported.

The Intel 430Fx chipset does not support DMA Bus Mastering. If your computer is equipped with this chipset, the F5 Carrier Board cannot initiate a DMA transfer to the PCI bus as a PCI master.

Example Code

Internal RAM to Far Global SRAM transfer:

```
RESULT  result;
..
result = C4X_Write(SHARED_SRAM, (UINT32 *)0xC0301000,
                   (UINT32 *)0x2ff800, 0x10, NO_FLAGS);
..
```

COMM port 0 to Far Global SRAM:

```
RESULT  result;
..
result = C4X_Write(SHARED_SRAM, (UINT32 *)0xC0301000,
                   (UINT32 *)0x100041, 0x10, STATIC_SRC);
..
```

Far Global SRAM to PCIbus (node A only):

```
RESULT  result;
..
result = C4X_Write(PCI, pci_addr, (UINT32 *)0xC0301000, 0x10,
                    DMA_ENABLE | DMA_SYNC);
..
```

10 Example Programs

The examples provided with the F5 SDK demonstrate the functionality of the board and provide you with a framework for building your own software application. These examples can be found in your *<F5RootDirectory>\examples* subdirectories. This chapter discusses the purpose of each program, and how to run and rebuild the example programs.

10.1. Purpose of Each Program

All of the following programs (except **guisamp**) are provided in both C and Visual Basic versions.

- **busmastr** illustrates how to master the PCI bus from the embedded C4x node using the PCI9060/9080's DMA.
- **dspload** illustrates how to download C4x code from the Host to the F5.

  Note that you can load and run **blink.out**, a DSP program which flashes the on-board LED, directly from a debugger.
- **glbsram** illustrates how to use Far Global (Shared) SRAM to share data between C4x nodes and the Host PC.
- **guisamp** illustrates how to make a simple GUI program in C to download C4x code from the Host to the F5.

  This is a GUI version of the **dspload** example program (see above).
- **interrupts** illustrates the following interrupts:
  - Host to C4x Node interrupts
  - C4x Node to Host interrupts
  - Nodes B, C, D to Node A interrupts
  - PCI doorbell interrupts

10.2. How to Run the Example Programs

Note: The Visual Basic (VB) examples are for the Windows NT environment only. To run the Visual Basic examples, the user must have Visual Basic version 5.0 or higher and must possess a working knowledge of VB 5.0 development environment and programming.

To run the examples:

1. Make sure your system configuration corresponds to the SDF used by the examples.

   All examples use the **F5.sdf** System Definition File (SDF) that's in *<F5RootDirectory>\include*. This SDF defines the following default system configuration:

   - a single board system
   - board ID = 0x1
   - one embedded processor, no TIM modules

   Note: To run the **entire** interrupts example, an F5 board with one embedded processor and 3 TIM modules are required.

2. If your system does not have the above default configuration, locate the SDF that corresponds to your system's configuration. All SDFs can be found in the *<F5RootDirectory>\include* directory. You may have to create a new SDF, using Toolbox, or edit one of the SDF using a text editor (see *section 6.4 Defining your System's Hardware Configuration*).

3. Replace the **F5.sdf** file with the SDF that properly reflects your system's configuration. You can do this using the DOS **Copy** command. For example, the following command copies the contents of the F5_x.sdf file into the F5.sdf file:

   ```
   Copy F5_x.sdf F5.sdf
   ```

4. If the system configuration is different from the default you will have to create a new LDF. If you generated a new SDF using Toolbox, the proper LDF file will already be created. Copy this LDF file to the example program's folder and edit it so that the program's name is in place of the highlighted sections below.

   ```
   [Files]
      Board1:NodeA:Proc0 = ________.out  ;  // Embedded TIM A
      Board1:NodeB:Proc0 = ________.out  ;  // TIM B
      Board1:NodeC:Proc0 = ________.out  ;  // TIM C
      Board1:NodeD:Proc0 = ________.out  ;  // TIM D

   [end]
   ```

5. **For C programs:** From a command prompt, go to the subdirectory containing the example you want to run and type the example name, then press **Enter**. For example,

to run the dspload example, go to the *<F5RootDirectory>\examples\dspload* directory and at the command prompt enter:

```
dspload
```

To see what text should be displayed on the screen, consult the next section (*section 10.3*).

No errors should occur if your system is correctly set up.

6. **(Windows NT only) For Visual Basic programs:**

a) Start Visual Basic.

b) Go to the subdirectory containing the example you want to run.

c) Open and run the corresponding Project file (has extension **.VBP**).

d) Click **"Start"**.

No errors should occur if your system is correctly set up.

The screen displays for the Visual Basic examples are shown on the following pages.

10.3. Screen Displays of the Example Programs.

All of the following are displays when running the Visual Basic example programs. When running the C example programs, the same text is displayed, except the text is not in a window. The exception to this is **guisamp**, which is a C program for Windows NT and, as shown in *Figure 15*, displays text in a window.

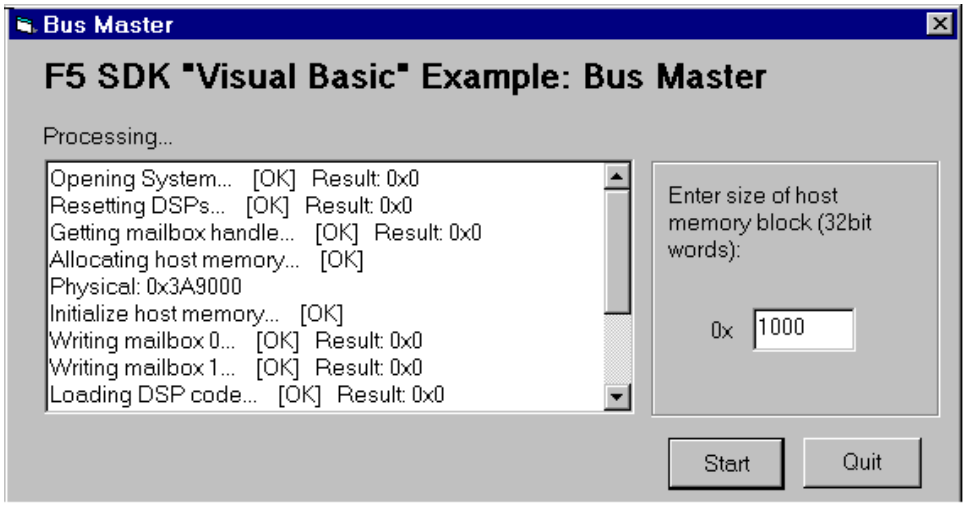


Figure 12 Screen Display of the "busmastr" Example Program

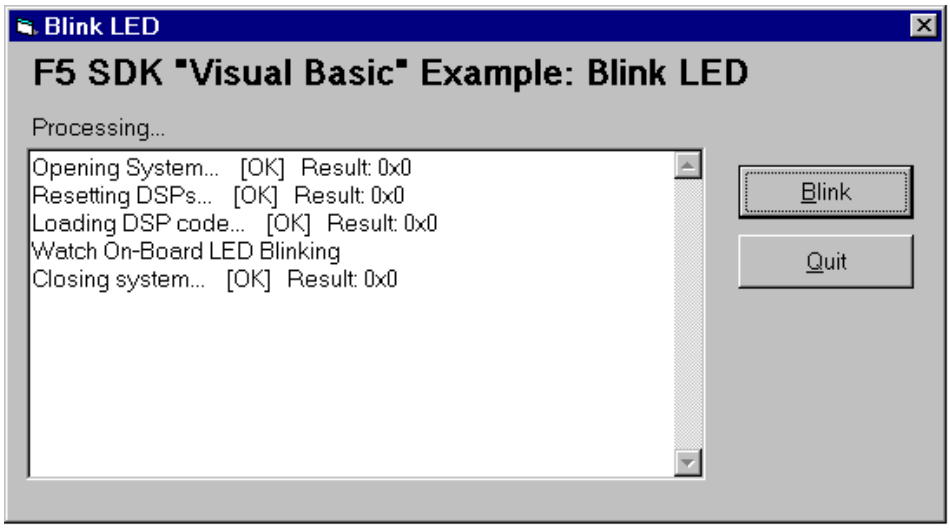


Figure 13 Screen Display of the “dspload” Example Program

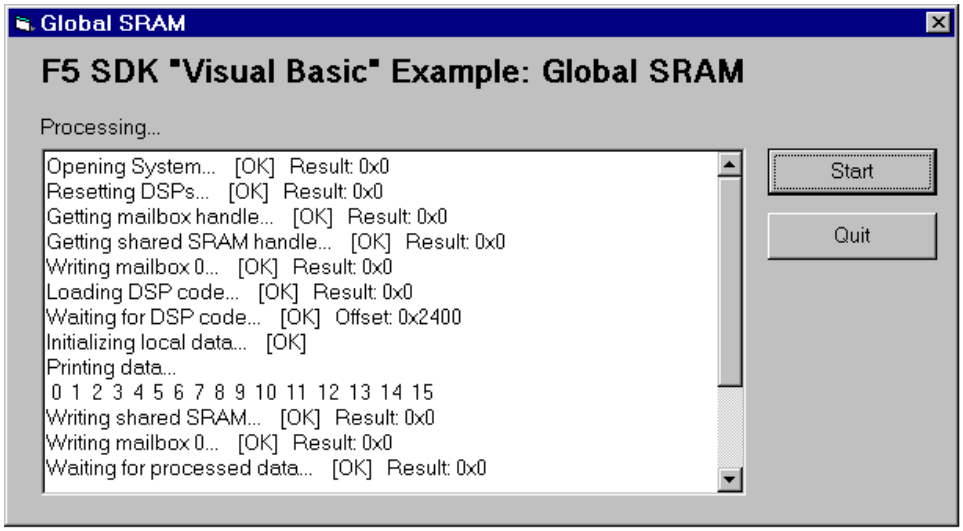


Figure 14 Screen Display of the “glbsram” Example Program

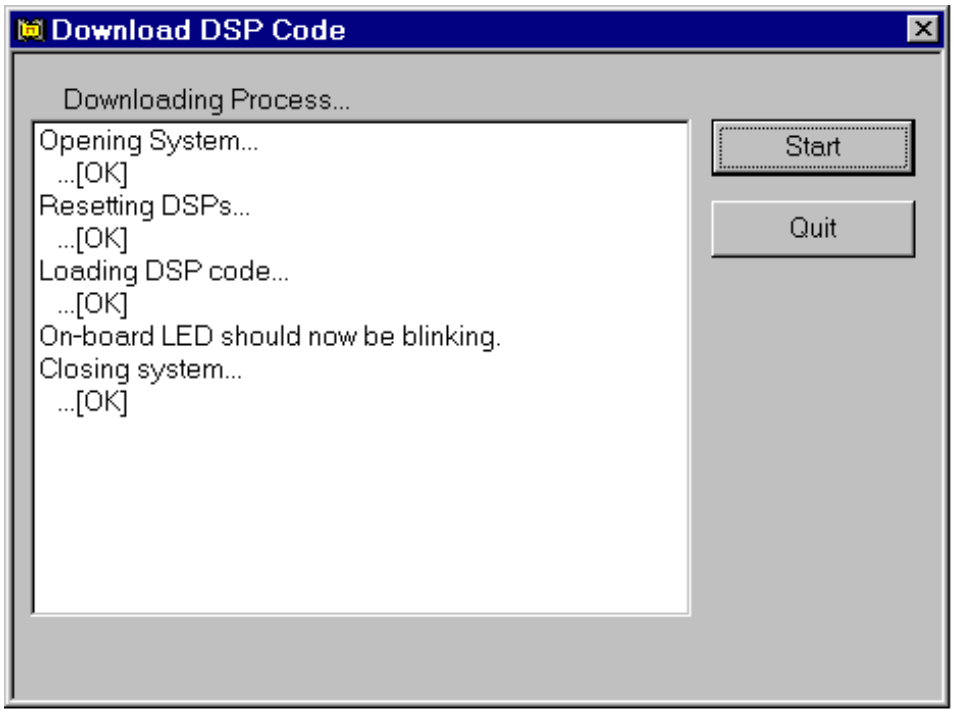


Figure 15 Screen Display of the “guisamp” Example Program

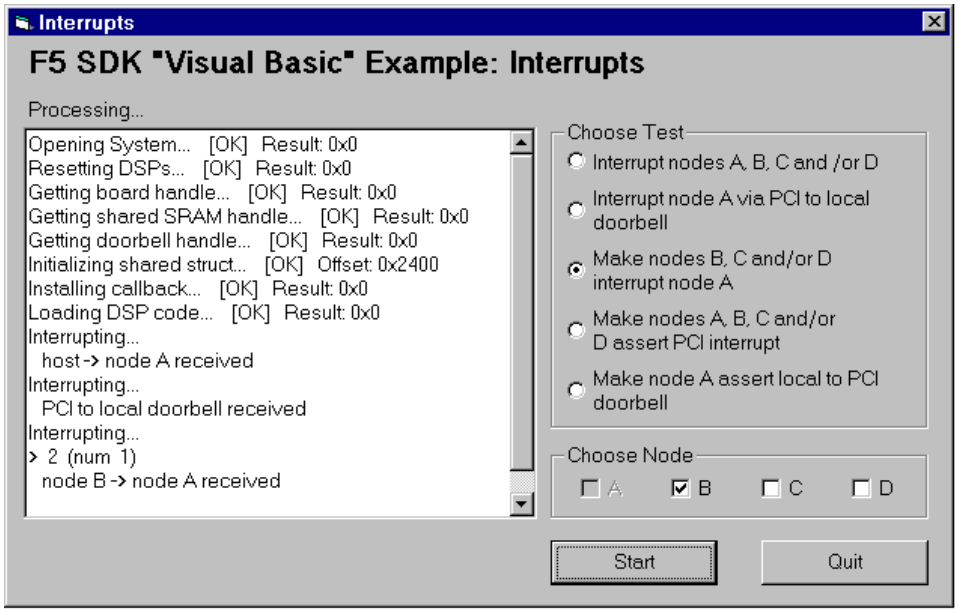


Figure 16 Screen Display of the “intrupts” Example Program

10.4. Verifying the Device Driver and Host Library Installation

To test the device driver and the host Application Library (ALIB):

1. From the **Start** menu, select **Programs**, then **F5 SDK**, then **F5 SDK Tester.**
2. To see the options available, after the program (F5LB_T) starts running, type `?` at the `>` prompt.

If your hardware, device driver, and host ALIB have been correctly installed, the F5LB_T program console (shown below for Windows NT; Windows 95 would be similar) , will appear on your screen.

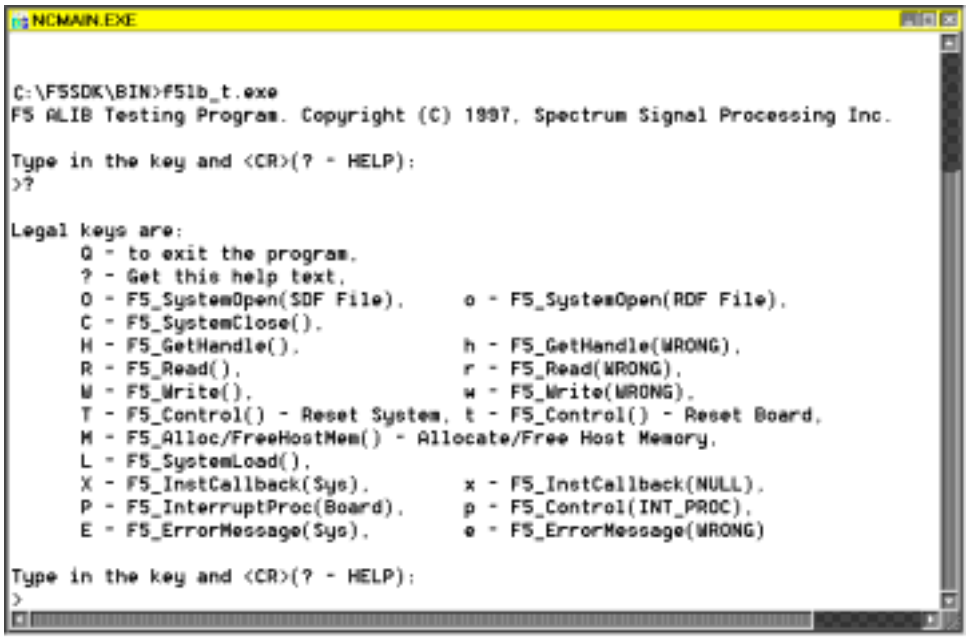


Figure 17 Screen Display of F5 SDK Tester

You can test individual ALIB functions by entering the letter that corresponds to the function at the command prompt. Note that the commands are case-sensitive. For example, to load a DSP application onto an F5 board:

1. Enter “O” to initialize and open the system. Specify `..\INCLUDE\F5.SDF` as the System Definition File (SDF). This file is found in *<F5RootDirectory>\include*. See *section 6.4 Defining your System’s Hardware Configuration* for more information on SDFs.
2. Enter “T” to reset the system.
3. Enter “L” to load the DSP code. Specify `dspload.ldf` as the Load Definition File (LDF). This file is found in *<F5RootDirectory>\examples\dspload*. See *section 6.6 Defining the DSP Code to Download* for more information on LDFs.
4. Check that the code’s expected results have actually taken place. For example, for `dspload`, check that the on-board LED is blinking.

5. Enter “C” to close the system.
6. Enter “Q” to exit the program.

10.5. Tips and Troubleshooting

- Pressing `<Ctrl><C>` will close the example program you are running in F5 SDK Tester.
- If you encounter problems, ensure that SSP_PATH is properly set (see *section 6.3*). To check SSP_PATH, go to the command prompt and type `set`
- If you get a message displayed by **F5 SDK Tester**, check the following table.

Table 9 Troubleshooting - Messages Displayed by F5 SDK Tester

Message displayed in F5 SDK Tester	How to interpret
PASSED	The function was successful.
FAIL. Error code: ….	The function was unsuccessful. To see a text message describing the error code, at the > prompt of the **F5 SDK Tester**, type E followed by a space and the error code number. For example, if the message “FAIL. Error code: 0x0040301” was displayed, type `E 40301`
ILSEQ	Functions were not called in a “logical” sequence. Functions must be called in a “logical” sequence. For example, you cannot perform an operation on a system until the system has been initialized with the **F5_SystemOpen** function. See *section 6.2 Calling Host Functions* for more information on the order of host function calls.

10.6. How to Rebuild the Host Library

The F5 host ALIB builds the following targets:

1) f5alib.dll

2) f5alib.lib

`f5alib.dll` requires two static libraries `cofflib` and `sdflib`.

Note: The supplied “make” files are shipped in MS VC (Microsoft Visual C) 5.0 - compatible format. MS VC version 4.2 cannot process Make files from version 5.0.

To rebuild the F5ALIB host library:

1. Set up your environment for compiling. For example, for MSVC version 5.0, run the following batch file:

   `<MSDEVRootDirectory>\bin\vcvars32.bat`

2. Go to the directory `<F5RootDirectory>\Alib\Host\Build` that contains the make file `f5alib.mak`. `F5alib.mak` builds both `f5alib.dll` and `f5alib.lib`.

3. Enter the following commands:

   - For *Release* version:

   ```
   nmake /f f5alib.mak CFG="f5alib - Win32 Release" clean
   nmake /f f5alib.mak CFG="f5alib - Win32 Release"
   ```

   The targets can be found in the following directories:

   ```
   f5alib.dll - <F5RootDirectory>\Alib\Host\Bin\Release
   f5alib.lib - <F5RootDirectory>\Alib\Host\Lib\Release
   ```

   - For *Debug* version:

   ```
   nmake /f f5alib.mak CFG="f5alib - Win32 Debug" clean
   nmake /f f5alib.mak CFG="f5alib - Win32 Debug"
   ```

   The targets can be found in the following directories:

   ```
   f5alib.dll - <F5RootDirectory>\Alib\Host\Bin\Debug
   f5alib.lib - <F5RootDirectory>\Alib\Host\Lib\Debug
   ```

4. Copy these target files into the following directories:

   ```
   f5alib.dll - <F5RootDirectory>\Bin and C:\Windows
   f5alib.lib - <F5RootDirectory>\Lib
   ```

5. Now the new library can be used to rebuild the example programs.

10.7. How to Rebuild the Example Programs

When you are building your own application and are encountering compiler/linker errors you may want to try rebuilding the example programs. Rebuilding the example programs can tell you if your compiler/linker is compatible or not.

If the example programs compile without any trouble, there is likely some problem with your application code. If the example programs do not compile, you may have incompatible versions of compiler and linker.

Note: The supplied "make" files are shipped in MS VC (Microsoft Visual C) 5.0 - compatible format. MS VC version 4.2 cannot process Make files from version 5.0.

To rebuild the F5 host-based example programs:

1. Set up your environment for compiling. For example, for MS VC version 5.0, run the following batch file:

   <MSDEVRootDirectory>\bin\vcvars32.bat

2. Go to the subdirectory containing the example that you want to rebuild, and enter the following command:

   `nmake -f examplename.mak`

   where *examplename* corresponds to the example that you're rebuilding. For example, to rebuild the **dspload** example, go the dspload examples subdirectory, and enter the following:

   ```
   nmake -f dspload.mak
   ```

To rebuild the DSP-based example programs the mkdepend.exe GNU utility is used and installed from the PC GNU utilities disk.

1. Set the following environment variable:

   `SET C_DIR = <TIToolsDirectory>`

   where *TIToolsDirectory* is the name of the directory containing Texas Instruments development tools (compiler, linker, etc..).

2. Make sure that TI's compiler and linker files are included in your path.

3. Go to the example subdirectory containing the example that you want to rebuild, and enter the following commands:

   ```
   nmake -f makefile.c4x depend
   nmake -f makefile.c4x clean
   nmake -f makefile.c4x
   ```

Appendix A: Status Codes

This appendix lists the possible F5 host and DSP application library status codes.

Table 10 Status Codes for Host Library Functions

Status Code	Description
F5_RES_WRONG_PARAMS	An invalid parameter has been passed to an F5 host ALIB function.
F5_RES_TOO_MANY_SYSTEMS	Too many systems are open. A maximum of 20 systems can be open at one time.
F5_RES_BOOT_LOADER_NFOUND	The sspboot.out file was not found. It should be in your current directory.
F5_RES_LIB_DRIVER_NOT_OPEN	The Driver has not been opened.
F5_RES_LIB_DRIVER_NOT_FOUND	The Driver was not found. The vf5d.vxd file should be in your *windows**system* directory.
F5_RES_LIB_DRIVER_WRONG_VER	The driver version is incompatible with F5 ALIB.
F5_RES_LIB_LIBRARY_NOT_OPEN	The ALIB has not been initialized. A call to F5_SystemOpen should be made to initialize the library.
F5_RES_SDF_CORRUPT	SDF syntax error.
F5_RES_SDF_NOT_FOUND	The SDF file was not found.
F5_RES_SDFSYS_CORRUPT	The system specified in the SDF file is invalid.
F5_RES_SDF_CODE_NFOUND	User code file specified in the SDF was not found.
F5_RES_LDF_CORRUPT	LDF syntax error.
F5_RES_LDF_NOT_FOUND	The LDF file was not found.
F5_RES_LDF_RW_ERR	An error occurred trying to read to/write from an LDF.
F5_RES_LDF_RESC_NFOUND	A resource specified in an LDF was not found.
F5_RES_LDF_CODE_NFOUND	User code file specified in the LDF was not found.
F5_RES_OS_NOT_ENOUGH_MEM	Your PC does not have sufficient memory to allocate the required space.
F5_RES_OS_WRONG_VER	The version of the operating system is not supported.
F5_RES_OS_CANT_OPEN_SEMA	The O/S cannot create a vital multi-tasking object.
F5_RES_DRV_CALL_FAILURE	The Driver cannot perform the requested operation.
F5_RES_DRV_BRD_NOT_FOUND	The specified board was not found.
F5_RES_DRV_WRONG_PARAMS	An invalid parameter was passed to the driver.

Status Code	Description
F5_RES_DRV_CANT_ALLOC_MEM	Not enough memory to allocate memory for the request.
F5_RES_DRV_NO_CONTIG_MEM	Not enough contiguous memory to allocate memory for the request.
F5_RES_DRV_CANT_LOC_MEM	The Driver cannot lock the memory requested.
F5_RES_DEV_TAKEN	The specified board is already being used by another process.
F5_RES_DEV_NOT_PRESENT	The specified board cannot be found..
F5_RES_DEV_TIMEOUT	The O/S elapsed waiting for the DSP to complete an operation.
F5_RES_RES_CORRUPT	The resource is invalid.
F5_RES_RES_NOT_FOUND	The name of the resource was not found.
F5_RES_RES_RLST_CORRUPT	The Resource Table (from the RDF file) is invalid.
F5_RES_RES_INVAL_RESC_TYPE	An invalid resource type was specified.
F5_RES_RES_BOARD_NOT_FOUND	The resource cannot find the specified board.
F5_RES_RES_CODE_NFOUND	The User Code File List (from an RDF file) references a file which does not exist.
F5_RES_SYS_CORRUPT	An invalid system handle was specified.
F5_RES_SYS_NOT_FOUND	The system handle is invalid.
F5_RES_SYS_RUNNING	The system handle was not found.
F5_RES_SYS_NO_KERNEL	There is no kernel present on Node A of the specified system.
F5_RES_SYS_NO_RESET	The board or system was not reset before attempting to load user code.

Table 11 Status Codes for DSP Library Functions

Status Code	Description
NO_ERROR	The function completed successfully.
ILLEGAL_FLAGS	The flag specified is invalid.
ILLEGAL_RESOURCE	The resource specified is invalid or unknown to the current node.
ILLEGAL_OPERATION	An unknown Resource/Operation or Resource/Flag combination was specified or the operation cannot be performed for the current node.
ILLEGAL_ADDRESS	Destination address/block-length is out of bounds.

Appendix B: Sample Linker Command File

C4xlink.cmd

```
-cr
-l f5_c4x.lib    /* F5 C4x application library */
-l rts40.lib     /* TI run-time support libraray */


/* Specify standard memory configuration  */
MEMORY
{
   IRAM0:  origin = 002FF800h   length = 0400h  /* Internal RAM 0, 1K    */
   IRAM1:  origin = 002FFC00h   length = 03F0h  /* Internal RAM 1,       */
   NLRAM:  origin = 00300000h   length = 20000h /* Near Local SRAM, 128K */
   PEROM:  origin = 40000000h   length = 8000h  /* PEROM, 32K            */
   NGRAM:  origin = 80000000h   length = 20000h /* Near Global SRAM, 128K*/
   FGRAM:  origin = 0C0300900h  length = 7F700h /* Far Global SRAM, 512K */
}

/* Specify output sections */
SECTIONS
{
   .text           : { }   > NLRAM
   .data           : { }   > NLRAM
   .bss            : { }   > NLRAM
   .const          : { }   > NLRAM
   .cinit          : { }   > NLRAM
   .stack          : { }   > NLRAM
   .vector         : { }   > IRAM1
   .user1          : { }   > FGRAM
}
```

Appendix C: System Definition File: Description and Example

System Definition File (SDF) Description

An SDF is a text file that defines your system's hardware components. The SDF is broken up into sections identifying the F5 system, F5 boards, TIM modules, board resources (such as common RAM), processors, and boot processors.

Sections and Attributes

A section is identifiable by a name enclosed in square brackets, for example [`TestSystem`]. Each section can contain several attributes describing the section. For example, TIM modules and F5 board sections contain Component and COMMConnection attributes identifying processors and COMM port connections. Section names cannot contain spaces and are case sensitive.

Sections should be defined in the SDF in the following order:

1. Processor sections
2. Resource (PLX Register, Common RAM, Arbiter Register, IRQ Register) sections
3. TIM module sections
4. Board sections
5. System section
6. Boot Processor section

Note: A section can be referenced in other sections. However, the section must be defined prior to being referenced by other sections.

Reserved Section Names:

- BootProc
- End

Types and Models

The type of each section, except for the reserved sections, is identified by the Type attribute. Type can be one of the following:

- Processor
- TIM
- Resource
- Board
- System

The Model attribute is used to provide additional information about the section. The Model attribute **does not** affect the SDF and can be anything you want.

BaseAddress

The BaseAddress attribute of a board definition section identifies the Board ID.

> **Note:** Each board in your system must have a unique Board ID.

Components

The Components attribute identifies the components belonging to a specific TIM module, Board, or System definition section.

Syntax:

```
Components = { (<identifier>, <# COMM ports>, <defining section name>),
               (<identifier>, <# COMM ports>, <defining section name>)
             }
```

The following is an example of a Components attribute declaration of a TIM module:

```
Components  = { (Proc0, 4, PROCA_4x)
              }
```

You can identify a component by the component's Section name. However, the component must have been previously defined in the SDF. In the above example, the component's processor is defined by its section name, PROCA_4x.

COMMConnections

The COMM port connections for TIM modules, F5 boards, and F5 systems are identified by the COMMConnections attribute. There can only be one entry for each bi-directional COMM port connection. COMMConnections are not required for systems with only one board and no TIM modules.

Syntax:

```
COMMConnections = {(<source identifier>, <COMM #>, <dest identifier>, <COMM #>),
                   (<source identifier>, <COMM #>, <dest identifier>, <COMM #>)
                  }
```

The following is an example of a COMMConnections attribute of a TIM module:

```
COMMConnections = { // Src,   ID,  Dst,  ID
                        (Proc0,  1,  TIM,  1), // Off Cluster connection
                        (Proc0,  2,  TIM,  2), // Off Cluster connection
                        (Proc0,  4,  TIM,  4), // Off Cluster connection
                        (Proc0,  5,  TIM,  5), // Off Cluster connection

                    }
```

> **Note:** The COMMConnections of System definitions specify the front panel COMM port connections in your system.

The <source identifier> and <dest identifier> can be that section's "Type". For example, in a TIM definition section, the keyword "TIM" can be used as a source or destination identifier in the COMMConnections attribute. Likewise, in a board definition section, the keyword "Board" can be used. Such assignments connect the Link ports of the components to a logical connector on the actual TIM or board. These associations are then used to connect the logical TIM Link ports to the logical "Site" Link ports in the Board Definition.

Processor Definitions

Processor definition sections define the types of processors in your system. The following is an example of a processor definition section:

```
[PROCA_4x] // Processor for Site_A
   Type        = Processor;
   Model       = C4x;
   BootPort    = ALL;    // Can boot from any Com port
   Speed       = 40;
   LMCR        = 0x3D840000;
   GMCR        = 0x3D840000;
   MemOffset   = 0x0;    // Proc_ID for Site_A
```

The **BootPort** attribute is used to identify which COMM ports the processor is capable of booting from. Two keywords (ALL and NONE) and any number in the specified range for the processor are valid. "ALL" indicates to the SDF parser that any COMM port is suitable for loading. "NONE" indicates that the processor is not to be loaded, but exists in the F5 System Structure. A number indicates to the SDF software that loading is to be done using the specified COMM port only. If a processor is not reachable for loading by COMM port, the SDF software will produce an error.

TIM module Definitions

TIM sections define TIM modules. A module's processors are defined in the Components attribute, and its COMM port connections are defined in the COMMConnections attribute.

The following is an example of a TIM module definition section.

```
[F5_SITE_A]
   Type = TIM;
   Model = F5Site;
   Components = {
                  (Proc0,   4,   PROCA_4x)
                }
   COMMConnections = {  // Src,  ID,   Dst,     ID
                        (Proc0,  1,    TIM,     1),    // Off Cluster connection
                        (Proc0,  2,    TIM,     2),    // Off Cluster connection
                        (Proc0,  4,    TIM,     4),    // Off Cluster connection
                        (Proc0,  5,    TIM,     5)     // Off Cluster connection
                     }
```

Board Definitions

Boards are comprised of resources, COMM connections (for boards with multiple TIM sites), and attributes. The BaseAddress attribute is a string defining an I/O resource descriptor.

The following is an example of a Board definition section.

```
[F5_BOARD]
       Type = Board;
       Model = F5;
       BaseAddress = 0x123;  // F5 Board's own ID!!!
       Components =
              {
              // LocalName,        #Conn.,  GlobalName
              (SiteA,              4,       F5_SITE_A),
              (SiteB,              4,       F5_SITE_B),
              (PlxLocalRegs,       0,       PLX_REGS),   // Mailboxes + Doorbells
              (SharedRam,          0,       COMMON_RAM), // Global Shared RAM
              (ArbRegs,            0,       ARB_REGS),   // Arbitration Regs
              (IrqRegs,            0,       IRQ_REGS)    // F5 IRQ Regs
              }

       COMMConnections =
              {
              (SiteA,        5, SiteB,   2)        // Proc_A
              }
```

Resource Definitions

Resource definition sections define your board resources: PLX registers, Common RAM, arbiter registers, and IRQ registers.

The following is an example of a Resource definition section.

```
[PLX_REGS]
   Type        = Resource;
   Model       = PLX_REG;
   TypeID      = 0x100;
   MemBase     = 0x0;   // Inherited from board base address
   MemOffset   = 0x40;  // Bytes
   MemLength   = 0x30;  // Bytes
```

System Definition

The System section defines your F5 system. An F5 System is composed of boards and their COMM port connections (for systems with multiple boards only). There should be only one System section defined in your SDF.

The following is an example of a System definition section.

```
[TestSystem]
    Type = System;
    Model = F5_System;
       Components =
              {
              (Board1, 8, F5_BOARD)
              }
       // no COM connections for single board
```

Note: The COMMConnections portion of a System definition section specifies the front panel COMM port connections.

BootProc

The BootProc section defines the boot processors which are responsible for loading software onto the other processors in the system, and are accessible from the host directly. At least one boot processor must be present in the BootProc definition.

Syntax:

```
<Board Name>:<Cluster Name>:<Boot Processor Name> = <dummy number>;
```

Note that any number up to eight hexadecimal digits can be specified for <dummy number>. This parameter is present for syntax consistency purposes only. The following is an example of a BootProc section:

```
[BootProc]
    Board1:SiteA:Proc0 = 100;
```

System Definition File (SDF) Example

The following is a sample SDF for an F5 system with one board that has one embedded processor and one TIM module installed. This and other sample SDFs can be found in your *<F5RootDirectory>include* directory.

```
[PROCA_4x] // Processor for Site_A
   Type        = Processor;
   Model       = C4x;
   BootPort    = ALL;    // Can boot from any Com port
   Speed       = 40;
   LMCR        = 0x3D840000;
   GMCR        = 0x3D840000;
   MemOffset   = 0x0;    // Proc_ID for Site_A

[PROCB_4x] // Processor for Site_B
   Type        = Processor;
   Prototype   = PROCA_4x;
   MemOffset   = 0x1;    // Proc_ID for Site_B

[PLX_REGS]
   Type        = Resource;
   Model       = PLX_REG;
   TypeID      = 0x100;
   MemBase     = 0x0;   // Inherited from board base address
   MemOffset   = 0x40;  // Bytes
   MemLength   = 0x30;  // Bytes

[COMMON_RAM]
   Type        = Resource;
   Model       = SRAM;
   TypeID      = 0x101;
   MemBase     = 0x0;   // Inherited from board base address
   MemOffset   = 0x0;
   MemLength   = 0x200000; // bytes for 512K SRAM; (0x80000 for 128K SRAM)

[ARB_REGS]
   Type        = Resource;
   Model       = SRAM;
   TypeID      = 0x102;
   MemBase     = 0x0;   // Inherited from board base address
   MemOffset   = 0x0;
   MemLength   = 0x20;

[IRQ_REGS]
   Type        = Resource;
   Model       = SRAM;
   TypeID      = 0x104;
   MemBase     = 0x0;   // Inherited from board base address
   MemOffset   = 0x0;
   MemLength   = 0x20;

[F5_SITE_A]
   Type = TIM;
   Model = F5Site;
   Components = {
                  (Proc0,   4,   PROCA_4x)
                }
   COMMConnections = {  // Src,  ID,   Dst,     ID
                        (Proc0,  1,    TIM,     1),    // Off Cluster connection
                        (Proc0,  2,    TIM,     2),    // Off Cluster connection
                        (Proc0,  4,    TIM,     4),    // Off Cluster connection
                        (Proc0,  5,    TIM,     5)     // Off Cluster connection
                     }

[F5_SITE_B]
   Type = TIM;
   Model = F5Site;
   Components = {
                  (Proc0,   4,   PROCB_4x)
                }
   COMMConnections = {  // Src,  ID,   Dst,     ID
                        (Proc0,  1,    TIM,     1),    // Off Cluster connection
                        (Proc0,  2,    TIM,     2),    // Off Cluster connection
                        (Proc0,  4,    TIM,     4),    // Off Cluster connection
```

```
                        (Proc0,  5,    TIM,     5)     // Off Cluster connection
                    }

[F5_BOARD]
        Type = Board;
        Model = F5;
        BaseAddress = 0x1;  // F5 Board's own ID!!!
        Components =
                {
                // LocalName,        #Conn.,  GlobalName
                (SiteA,              4,      F5_SITE_A),
                (SiteB,              4,      F5_SITE_B),
                (PlxLocalRegs,       0,      PLX_REGS),   // Mailboxes + Doorbells
                (SharedRam,          0,      COMMON_RAM), // Global Shared RAM
                (ArbRegs,            0,      ARB_REGS),   // Arbitration Regs
                (IrqRegs,            0,      IRQ_REGS)    // F5 IRQ Regs
                }

        COMMConnections =
                {
                (SiteA,        5, SiteB,   2)        // Proc_A
                }

[TestSystem]
    Type = System;
    Model = F5_System;
        Components =
                {
                (Board1, 8, F5_BOARD)
                }
        // no COM connections for single board

[BootProc]
    Board1:SiteA:Proc0 = 100; // Any number up to eight hexadecimal digits can be entered.
[end]
```

Appendix D: Definitions and Acronyms

ALIB	Application Library
ANSI	**American National Standards Institute** – the United States government body responsible for approving US standards in many areas, including computers and communications. ANSI is a member of ISO (International Organization for Standardization).
API	Application Programming Interface.
C4x	TMS320C40 or C44 Texas Instruments Digital Signal Processor.
COFF	**Common Object File Format** – a binary file format generated by linkers which contains code, loading information, and debugging information. Refer to the *TMS320 Floating-Point DSP Assembly Language Tools User's Guide* for more information.
COMM Port	8-bit parallel communication port on TMS320C40 or TMS320C44 DSPs. Used for communication between processors at up to 20 megabytes per second (50 MHz).
DLL	**Dynamic Link Library** – a library which is linked to application programs at runtime rather than as the final phase of compilation. This means that the same block of library code can be shared between several tasks rather than each task containing separate copies of the routines it uses.
DMA	**Direct Memory Access** – an electronic device allowing automatic data transfer between the Host's memory and outer memory without invoking the processor.
DSP	Digital Signal Processing or Digital Signal Processing chip
GMCR	Global Memory Control Register
I/O	Input/Output

IRQ	**Interrupt Request** – the name of an input found on many processors which causes the processor to suspend normal instruction execution temporarily and to start executing an interrupt handler routine. Such an input may be either: • "level sensitive" - the interrupt condition will persist as long as the input is active; or • "edge triggered" - an interrupt is signalled by a low-to-high or high-to-low transition on the input. Some processors have several interrupt request inputs allowing different priority interrupts.
ISR	Interrupt Service Routine.
JTAG	**Joint Test Action Group** - a serial boundary-scan interface used to control on-chip emulation functions of DSPs.
LMCR	Local Memory Control Register
O/S	Operating System
PCI	**Peripheral Component Interconnect** – an electrical interface standard for interconnecting system components. For desktop PC cards it also defines the physical standard.
PEROM	Programmable Erasable Read-Only Memory. Can be erased and reprogrammed in-circuit.
PnP	**Plug and Play Architecture** – a specification for automatic computer hardware configuration which prevents conflicts between devices, allows hot docking and enables enumeration of devices on the system.
SDK	Software Development Kit
TBC	Test Bus Controller

Index

H

I

J

L

M

P

R

S

T

U

V