Engineer-to-Engineer Note

EE-312

Building Complex VDK/LwIP Applications Using Blackfin® Processors

Contributed by Kaushal Sanghai   *Rev 2 – September 16, 2008*

Introduction

Analog Devices, Inc. offers a port of the light-weight TCP/IP (LwIP) stack for the Blackfin® family of embedded processors. The LwIP stack on Blackfin processors can be used to develop an embedded networking application in combination with an audio/video or an industrial automation/control application. LwIP is an attractive utility that allows you to quickly port a standalone embedded application to a networked embedded application. Several steps are necessary in the integration process in order to produce a highly efficient and robust application. This document provides guidelines to integrate LwIP with other system peripherals or applications to minimize common pitfalls in designing LwIP-based applications on Blackfin processors. This document also discusses system optimization techniques that maximize the performance of applications integrated with LwIP on Blackfin processors. The processor resources utilized by the LwIP port are also described to help you design an efficient system.

1 Motivation

Embedded networking applications are now increasingly geared toward transmitting multimedia content over wired or wireless networks. Industrial automation and control applications are other areas where adding network capability is desirable. Figure 1 shows applications in which Internet connectivity is desired.

Analog Devices offers royalty free (under license agreements) software building blocks to aid in faster development of complex embedded networking applications. The following is a list of essential and basic software components that are available with Analog Devices processor development environment suite, VisualDSP++® 4.5 and higher:

- LwIP – A light-weight TCP/IP stack [1] ported to ADSP-BF52x, ADSP-BF54x, ADSP-BF53x and ADSP-BF56x processors[1].

- Device drivers and system services libraries (SSLs) – API functions for configuring and managing system resources and peripheral devices [12].

---

[1] The LwIP stack is not available on ADSP-BF535 Blackfin processors.

- ■ VisualDSP++ Kernel (VDK) [15] – A real-time kernel. Use this low-resource kernel to efficiently manage system resources in a multi-tasking application environment.

- ■ MM SDK – Multimedia Software Development Kit [16]. This includes code examples of common real-world multimedia applications.
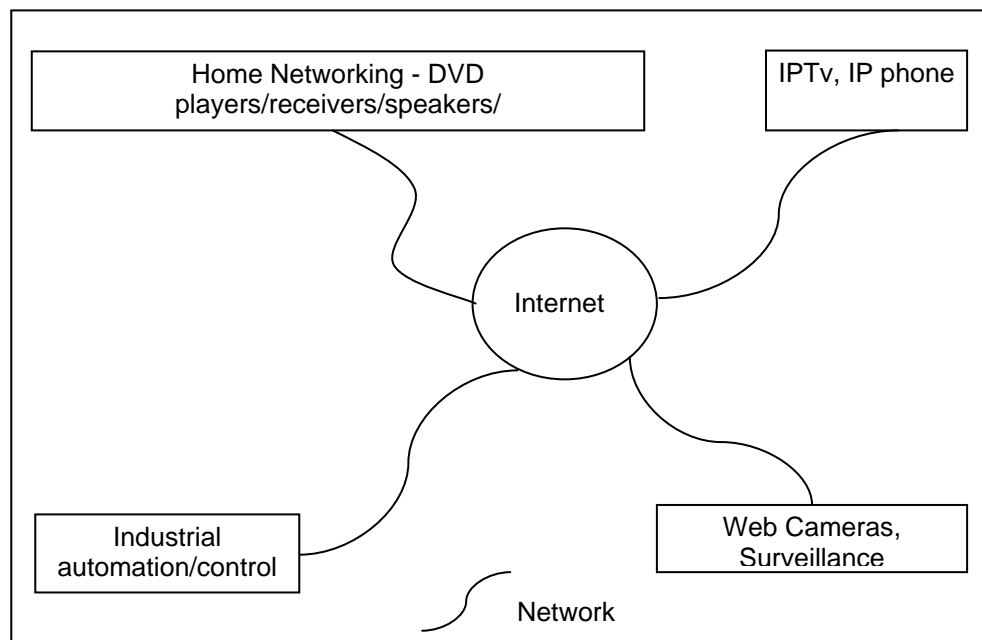

Figure 1. Typical Embedded Networking Applications

A real-world embedded networking application may involve the integration of these software building blocks into a system. The integration of different software components may cause several software and hardware resource conflicts. In addition, the use of software components with such a high level of abstraction on a resource constrained embedded platform may lead to under-utilization and mismanagement of system resources.

This EE-Note specifically describes several aspects of integrating LwIP with the software components described above. Following the procedures described in this document will aid you in faster development of LwIP-based applications on Blackfin processors. The remainder of the document is organized as follows:

- ■ Section 2 provides an overview of LwIP.

- ■ Section 3 discusses benchmark results and memory requirements for LwIP on Blackfin processors.

- ■ Section 4 discusses software and hardware guidelines for integrating LwIP with other applications.

- ■ Section 5 describes optimization techniques specific to the Blackfin platform to help you increase system and core performance.

This EE-Note is not a substitute for the LwIP manual, VDK manual, system services manual, or other related documentation [1] [2] [3] [4] [5] [6] [7] [8].

2 LwIP Overview

LwIP is a light-weight implementation of the TCP/IP protocol stack. LwIP is free and open source software available for download from http://savannah.nongnu.org/cgi-bin/viewcvs/lwIP/lwIP/. Due to its low memory requirements, LwIP is suitable for embedded applications. Detailed information on LwIP may be obtained from http://www.sics.se/~adam/lwIP/.

LwIP has been ported to ADSP-BF52x, ADSP-BF53x, ADSP-B54x and ADSP-BF56x processors, and the port is available with the VisualDSP++ development toolset. To start developing a TCP/IP application on Blackfin processors, you can use the ADSP-BF537 EZ-KIT Lite®, the ADSP-BF548 EZ-KIT Lite, the ADSP-BF527 EZ-KIT Lite, or the ADSP-BF526 EZ-KIT Lite evaluation boards, which have a dedicated Ethernet interface. For ADSP-BF533/BF532/BF531 and ADSP-BF561 processors, you can use the USB-LAN extender card. For details on how to get started building a TCP/IP project, refer to the *LwIP User Guide* [1]. The LwIP port to Blackfin uses the VisualDSP kernel (VDK) and the device drivers (DD) and system services libraries (SSL). Refer to [15] for VDK documentation and to [12] for system service libraries documentation.

3 LwIP Resource Usage on Blackfin Processors

This section discusses the throughput, memory requirements, and system resources utilized in the execution of LwIP on Blackfin processors.

3.1 Throughput for LwIP on Blackfin Processors

The throughput for the LwIP port was measured on Blackfin using the ADI-TTCP tool adapted from a freely available PCATTCP tool. The details on how to use the tool and PCAUSA's terms and conditions are described in the ADI-TTCP.doc, which is located in the accompanying .ZIP file [24]. Figure 2 shows peak throughput versus buffer size for LwIP on the ADSP-BF537 platform. Note that the indicated results may vary, based on network traffic and the supported bandwidth on the network. As shown in Figure 2, LwIP supports up to 11 Mbyte/s of throughput for the TCP and the UDP protocol stack.
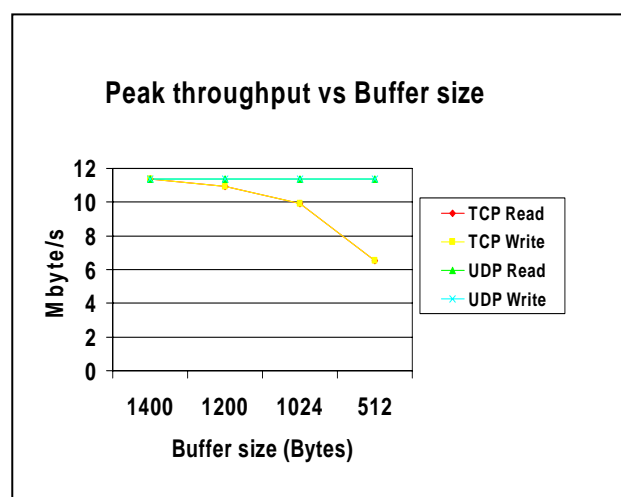

Figure 2. Buffer Size vs. Peak Throughput

3.2 Memory Footprints of LwIP on BF53x Processors

As mentioned above, the LwIP port uses VDK and the system services libraries (SSLs). The memory footprint is divided individually among these components to obtain a better estimate when integrating software components or projects that already use one of these components. Note that the numbers are approximate values and were derived from a simple "Hello world" program run on LwIP.

Software Component/Protocol	UDP (code)	TCP (code)	UDP (data)	TCP (data)
LwIP	35kB	52kB	10kB	12kB
System Service lib	15kB	15kB	1.2kB	1.2kB
VDK lib + miscellaneous	28kB	28kB	43kB	43kB
Total	**78kB**	**95kB**	**54.2kB**	**56.2kB**

Table 1. Code and data memory footprints for an LwIP-based application

Application Type	Driver buffers	Heaps (KB)	Memory pool (KB)	Packet buffers (KB)	Total size (KB)
Default	Rx = 8 * 1560 Tx = 2 * 1548 **~16KB**	~64kB	~7kB	~64kB	~151kB
Applications (Audio/ Video) (Large packet size)	Rx = 16 * 1560 Tx = 8 * 1560 **~50KB**	~128kB	~7kB	~128kB	~313kB
Small Applications (Control Applications) (Small packet size)	Rx = 8 * 256 Tx = 8 * 256 **~4KB**	~8kB	~7kB	~8kB	~27kB

Table 2. Heap area requirements for LwIP

3.3 Other Required System Resources

3.3.1 Interrupts and DMA

When using LwIP on ADSP-BF537, ADSP-BF527, and ADSP-BF526 processors, the drivers use the dedicated Ethernet controller. By default, the Ethernet DMA is mapped to IVG11 for which there is a dedicated DMA channel.

For ADSP-BF533/BF532/BF531, ADSP-BF54x and ADSP-BF561 processors, LwIP uses the USB-LAN extender card. By default, the device driver uses a single interrupt level for frame reception and transmit completion. IVG8 is used on ADSP-BF533 processors as the host interrupt level, and IVG11 is used on ADSP-BF561 processors. Refer to [3] [4] for additional information on device driver design.

4 Software and Hardware Guidelines

4.1 Software Guidelines

LwIP is ported to Blackfin on a VDK platform. It is expected that users of LwIP are familiar with general real-time operating system (RTOS) concepts. A discussion of detailed RTOS concepts is not in the scope of this application note. Refer to [22] for general RTOS concepts.

This section lists and describes common software conflicts that may arise when integrating LwIP with other software components or applications.

- At a minimum, any software component that involves the use of system service libraries will require the use of interrupts and the DMA manager. Integrating two or more projects that use system services libraries requires one interrupt and DMA manager for the entire application. This is also true for the device manager or deferred callback manager and all services used within the system.

- Integrating more devices or resources into the system requires additional memory allocation for the device driver manager and interrupt manager. The following listing shows the macros used to assign memory for DMA channels and devices. Use an appropriate multiplication factor, based on the number channels/devices used within the system.

```
// DMA Manager data (base memory + memory for 2 DMA channels)
static u8 DMAmgr_storage[ADI_DMA_BASE_MEMORY + (ADI_DMA_CHANNEL_MEMORY * 2)];

// Device Manager data (base memory + memory for 2 devices)
static u8 devmgr_storage[ADI_DEV_BASE_MEMORY + (ADI_DEV_DEVICE_MEMORY * 2)];
```

- Each software component may have its set of initialization routines, which may conflict on integration. Common initialization routines include interrupt initialization, device initialization, clock and system power settings, external memory, heap memory, and so on. The following code listing is extracted from the `sys_init()` function of the default VisualDSP++ LwIP project.

```
/* initialize the interrupt manager */
adi int Init(intmgr storage, sizeof(intmgr storage), &response count,
&critical_reg);
/* initialize the device manager */
adi dev Init(devmgr storage, sizeof(devmgr storage), &response count,
&devmgr_handle, &imask_storage);
```

4.2 Hardware Guidelines

A problem that often prevents a functional standalone system from working with other components is the limited bandwidth available on an embedded platform. Blackfin processors include internal buses that have different widths (ADSP-BF53x and ADSP-BF52x processors are 16 bits wide, and ADSP-BF54x and ADSP-BF561 processors are 32 bits wide). Each bus can run at a maximum speed of 133 MHz, which implies a maximum data rate of 266 or 532 Mbyte/s, depending on the size of the internal bus. Several factors can lower the maximum available bandwidth, such as system bus read/write turnaround times, bank conflicts, latency due to arbitration logic, core and DMA conflicts, and so on. These factors are application-specific and depend on the run-time interaction of peripheral data movement and core accesses. Section 5 discusses optimization techniques that help to minimize the factors affecting the system bandwidth, which result in a more effective use of the system resources.

4.2.1 Managing Interrupts

Several peripherals may be mapped to a single IVG level, and several software components may assume their ISR's as a primary interrupt handler for a specific IVG level. Upon integration, the interrupt service routine from only one of the software component will be registered in the event vector table (EVT) overwriting any previously assigned ISR's to that IVG level. In other words, make sure that you map only one primary interrupt service routine to an IVG level with additional peripherals interrupts routines mapped as secondary handlers. To avoid secondary handlers, identify peripheral priorities and map them to different IVG levels, if available.

4.2.2 Code/Data Memory Mapping

Any memory-mapping optimizations (for example, mapping critical or most executed code/data sections to L1 memory) would need to be re-analyzed when integrating software components. This is often ignored when software components are taken from different vendors that have individually hand-optimized the memory map for their software component. The behavior of the application (such as the execution percentage of certain code/data sections) may change in the integrated application, eventually resulting in higher memory access latencies compared to the individual execution of the software components. Code/data mappings should therefore be re-analyzed in the integrated application.

4.2.3 Exception and Hardware Error Handlers

VDK uses service exceptions to manage task scheduling and other kernel activities. It maps the exception handler to a routine in the kernel code. It passes all the error exceptions to a user exception routine, which it creates by default. For application that only use the system services libraries, you have to map the exception handler. However, when integrating a system services application into a VDK project, the exception handler should not be separately mapped and all exceptions should be handled in the default user exception routine created within a VDK-enabled project. Figure 3 shows the default exception handler routine created by a VDK project.

```
/* UserExceptionHandler */
/* User Exception Handler entry point */
.GLOBAL     UserExceptionHandler;
UserExceptionHandler:
    /**
     * Handle the exception...
     *
     * Remember that the VDK reserves the user exception 0, which
     * is accessed with the following instruction:
     *
     *   EXCPT 0;
     *
     * Any other exception (user or system), is your responsibility to
     * handle here.
     */
    RTX;
.UserExceptionHandler.end:
```

Figure 3. Default exception handler routine created by a VDK-enabled project

4.2.4 Error Interrupts

Most peripheral error interrupts are mapped to `IVG7`. Therefore, it is essential to explicitly resolve the various DMA or peripheral error to debug the application efficiently. Using the error interrupts in your system is the best way to determine when a peripheral has an overrun or an underrun. For example, a higher priority resource may prevent a lower priority resource from accessing memory, which may contain data or even a DMA descriptor.

5 System Optimizations

This section presents several hardware and software optimization techniques that will help you to fully take advantage of the Blackfin architecture. Users who are well versed with the Blackfin platform and the VisualDSP++ debugging environment can skip to Section 6.

5.1 Hardware Considerations

Although, several system optimization techniques can be incorporated, the following are the recommended minimum:

- **16/32-bit transfers**: Use maximum bus width for all peripheral DMA and memory DMA transfers. Initiate 16-bit transfers for ADSP-BF53x and ADSP-BF52x processors, and initiate 32-bit transfers the ADSP-BF561 and ADSP-BF54x processor.

- **Efficient use of DMA channels**: Do not carry out two simultaneous memory transfers on the same DMA channel.

- **SDRAM bank partitioning**: Partition the SDRAM into four banks to ensure simultaneous access to multiple data buffers and to minimize turn-around times. For example, the ADSP-BF561 EZ-KIT Lite board has 64 Mbytes of SDRAM, which can be configured as four 16-Mbyte internal SDRAM banks. To take advantage of the bank structure, do not map two data buffers, which are accessed simultaneously, to the same SDRAM bank.

- **DMA traffic control**: Use DMA traffic control registers for efficient system bandwidth utilization. The traffic control registers provide a way to influence how often the transfer direction on the data buses may change, by automatically grouping same-direction transfers. Refer to [14] and [20] for details on DMA performance optimization.

- **Minimize DMA/Core conflict**: Minimize core and DMA conflicts. Avoid core access to external memory. By default, core has higher priority over DMA for external memory accesses, but this can be changed by setting the `CDPRIO` bit in the `EBIU_AMGCTL` register. By giving DMA higher priority than core, incoming and outgoing data traffic from the peripheral devices is not halted on a core access to the external memory.

- **Instruction and data caching**: Enabling instruction and data caches can drastically increase application performance. If the application size is huge, which is usually the case with LwIP/VDK applications, cache can help to increase the performance greatly.

Refer to [14] [13], and [8] for a more detailed description of optimization techniques.

5.2 Software Considerations

5.2.1 Executing Interrupt Service Routine

Avoid data processing in ISRs or callback functions. Process data buffers in user threads. Interrupts run at a higher priority than user-defined threads. User-defined threads always run at IVG 15 (that is, at the lowest priority), and ISRs and callbacks [12] are executed at the corresponding IVG level. Use deferred callbacks to lower the priority of the corresponding ISR, permitting more efficient use of CPU time.

A lower-priority ISR will hold off a higher-priority ISR when the lower-priority ISR is not reentrant. Blackfin processors provide interrupt nesting, enabling higher-priority peripherals to interrupt the execution of lower-priority interrupts.

5.2.2 Efficient buffer management scheme

Use a double-buffering scheme to avoid processor stalls. Double-buffering or multiple buffering ensures that the processor can process the current buffer while the other buffers are filled in by peripheral DMA. Figure 4 shows a simple data flow diagram for managing data in a double-buffer implementation. Refer to [14] where more data flow diagrams show how to manage double/multiple data buffers in an application.
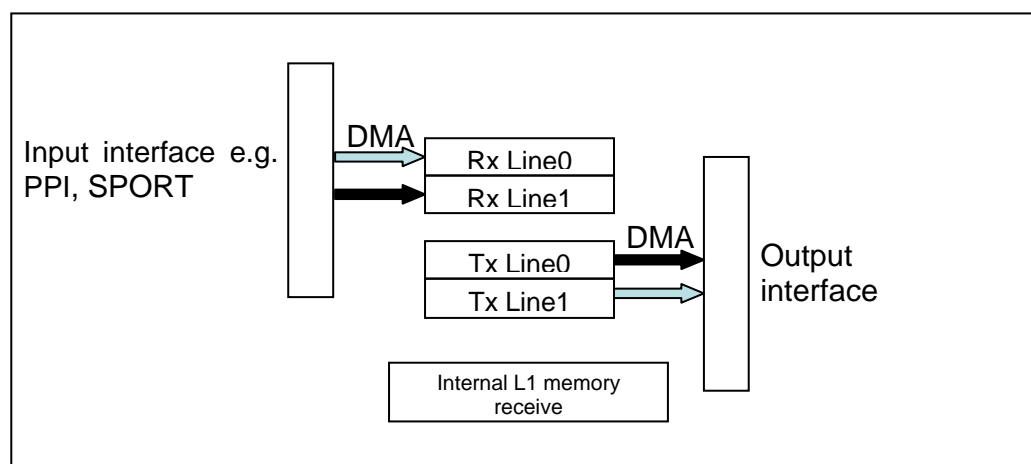

Figure 4. Data flow diagram for a simple double-buffering scheme

5.2.3 Compiler optimizations

Several compiler optimizations can be applied to increase the performance. The use of the -o switch and the -ipa switch provide the most improvement. Using profile-guided optimization (PGO) also helps to further increase the performance. For detailed optimization techniques, refer to the chapter entitled "Achieving Optimal Performance from C/C++ Sources" in the compiler manual [18]. Some ISRs and critical computation functions can be hand-optimized in assembly to increase performance.

5.2.4 Linker optimizations

Blackfin processor memory consists of L1 SRAM, which often is not used effectively. Mapping critical memory and most executed memory to L1 SRAM memories can eliminate most of the lost cycles due to cache miss penalties. Use the statistical profiler (Tools -> Statistical profiler) to locate the most executed code and to map those functions in L1 SRAM memory. Also, map critical ISR to L1 SRAM to ensure deterministic access time. Refer to [14] for information about code mapping on Blackfin processors.

6 General Debugging Tips

Apart from the usual debugging capabilities such as breakpoints and using the Expressions window or the Locals window, the combination of register set on the Blackfin architecture and the VisualDSP++ environment provides a rich set of debugging tools that enable you to debug complex systems. These are some of the key debug tools:

6.1 VDK History Window

Use the VDK History window (Figure 5) to track thread status, events, and race conditions between threads, and so on. This window is particularly useful when you debug applications that involve multiple threads. Open the VDK History window from the VisualDSP++ IDDE main menu by choosing `View -> VDK windows -> History.`
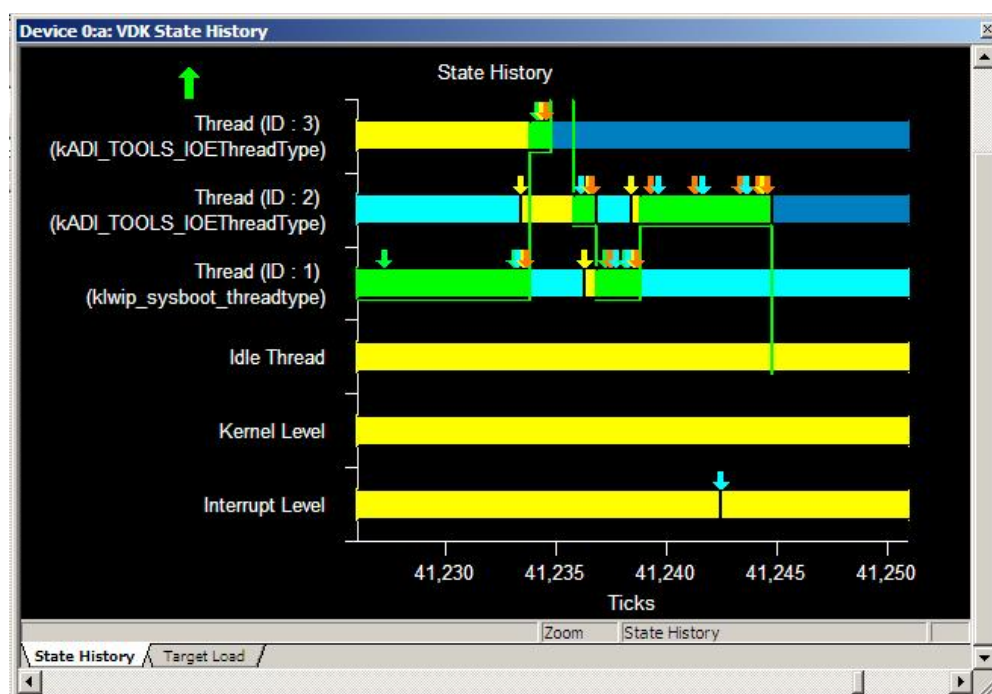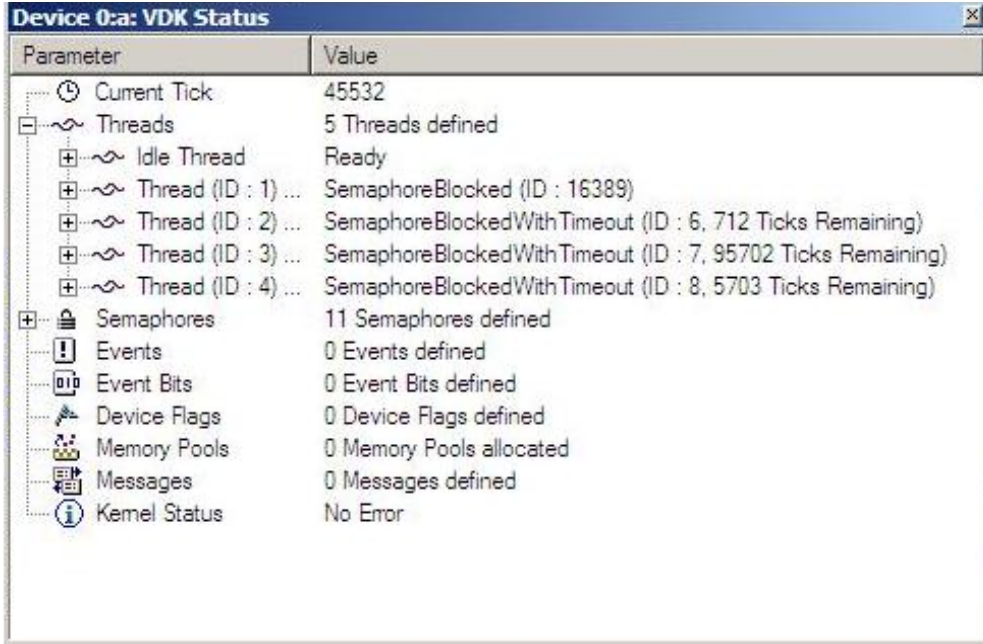

Figure 5. VDK History window

6.2 VDK Status Window

The VDK Status window (Figure 6) displays the status of all threads, semaphores, messages, and so on defined within the system. The VDK Status window also indicates the error code in case of a Kernel Panic. Common mistakes that generate kernel panic include calling restricted VDK API functions from ISRs or callbacks or accessing resources of a thread that has been destroyed. Open the VDK Status window from the VisualDSP++ IDDE main menu by choosing `View -> VDK windows -> Status.`

Figure 6. VDK Status window

6.3 Trace Unit and Sequencer Registers

To debug an exception, you must first locate the cause of the exception by examining the EXCAUSE field in the SEQSTAT register. The address of the instruction that caused the exception is stored in RETX register. Rerun the application by placing a breakpoint at the excepting address and enabling the Trace Buffer Unit by setting two bits in the TBUFCTL register. Then you can examine the sequence of instructions from the Trace Buffer Unit to determine the exact cause of exception.

6.4 Watchpoint Unit

The watchpoint register provides a means of debugging situations in which an error occurs only on the nth access to an instruction or data. In such cases, an emulation event can be generated using the watchpoint registers the $n-1^{th}$ time the instruction or data is accessed; then you step through to find the cause of exception or error.

6.5 Peripheral Status Registers

If a DMA or peripheral error is detected, you can examine the DMA/peripheral status registers to locate the cause of error. Apart from configuration mistakes, errors (such as DMA overrun or underrun errors) occur due to inefficient system design. At this point, it may help to manage the data buffers more efficiently or further divide the application into smaller tasks.

Also, if using system service libraries (SSL), most of the error codes are identified by the return codes of a system services API call and event codes in the callback function. The SSL covers events causing the DMA or peripheral errors. Refer to any of the examples included in the Multimedia starter kit [16].

The above list provides an overview of some debugging methods. For more information, refer to [19].

7 Conclusion

Integrating several software components is a complex and challenging task. When you use multiple software components, a system's complexity grows in terms of analyzing system resource utilization, bandwidth requirements, resolving potential software and hardware conflicts, debugging the application, and so on. The guidelines in this document can help you to avoid common pitfalls that may arise in integrating software components. The guidelines aid in developing an optimized and robust application.

Supported LwIP Features

For a list of current features available with LwIP concurrent with the release of this EE-Note, refer to the associated `.ZIP` file. Check for features added to VisualDSP++ in the latest release notes located here:

http://www.analog.com/processors/blackfin/evaluationDevelopment/crosscore/toolsUpgrades/index.html

Additional Reading

[1]   *LwIP User Guide*, *<install_path>*\Analog Devices\VisualDSP 5.0\Blackfin\lib\src\lwip\docs.

[2]   *BF537EthernetDeviceDriverDesign.doc, <install_path>*Analog Devices\VisualDSP 5.0\Blackfin\lib\src\lwip\docs.

[3]   *SMSCLAN91C111_DeviceDriver.doc, <install_path>*Analog Devices\VisualDSP 5.0\Blackfin\lib\src\lwip\docs.

[4]   *ADSP-BF533 Blackfin Processor Hardware Reference*. Rev 3.2, July 2006. Analog Devices, Inc.

[5]   *ADSP-BF561 Blackfin Processor Hardware Reference*. Rev 1.1, February 2007. Analog Devices, Inc.

[6]   *ADSP-BF537 Blackfin Processor Hardware Reference*. Rev 3.0, December 2007. Analog Devices, Inc.

[7]   *ADSP-BF52x Blackfin Processor Hardware Reference (Volume 2 of 2).* Rev 0.3, September, 2007. Analog Devices, Inc.

[8]   *ADSP-BF54x Blackfin Processor Hardware Reference (Volume 2 of 2).* Rev 0.4, August 2008. Analog Devices, Inc.

[9]   *Embedded Media Processing*. David Katz and Rick Gentile. Newnes Publishers., Burlington, MA, USA, 2005.

[10] *Digital Video and HDTV*. Charles Poynton. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[11] *Video Framework Considerations for Image Processing on Blackfin Processors (EE-276).* Rev 1, September 2005. Analog Devices Inc.

[12] *VisualDSP++ 5.0 Device Drivers and System Services Manual for Blackfin Processors*. Rev 3.0, August 2007. Analog Devices, Inc.

[13] *Video Templates for Developing Multimedia Applications on Blackfin Processors (EE-301).* Rev 1, September 2006. Analog Devices Inc.

[14] *PGO-Linker- A Code Layout Tool for the Blackfin Processors. (EE-306).* Rev 1, December 2006. Analog Devices Inc.

[15] *VisualDSP 5.0 Kernel (VDK) Users Guide*. Rev 3.0, August 2007. Analog Devices, Inc.

[16] *Multimedia Starter Kit.* http://www.analog.com/processors/platforms/msk.html. Analog Devices Inc.

[17] *VisualDSP++ 5.0 Linker and Utilities Manual,* Rev 3.0, August 2007. Analog Devices Inc.

[18] *VisualDSP++ 5.0 Blackfin C/C++ Compiler and Library Manual*, Rev 5.0, August 2007, Analog Devices Inc.

[19] *Blackfin Processor Troubleshooting Tips Using VisualDSP++ Tools (EE-307).* Rev 1, December 2006, Analog Devices Inc.

[20] *System Optimization Techniques for Blackfin Processors (EE-324).* Rev 1, July 2007, Analog Devices Inc.

References

[21] *Unix Network Programming*, volumes 1-2. W. Richard Stevens, Prentice Hall, USA, 1998.

[22] *MicroC/OS-II.* Jean J. Labrosse. CMP Books, San Francisco, CA, USA, 2002.

[23] *Computer Networks*, Andrew Tanenbaum, Prentice Hall, USA, 2002.

[24] *Associated ZIP File.* Rev 2, September 2008. Analog Devices, Inc.

Document History

Revision	Description
Rev 2 – September 16, 2008 *by Kaushal Sanghai*	Updated supported processors and tools revision. Also included the latest version of the ADI-TTCP tool in the associated .ZIP file.
Rev 1 – March 2, 2007 *by Kaushal Sanghai*	Initial release.