

Pismo Toolset

High-Performance Target Specific Libraries for Matador and Velocia DSP boards



Features

- Target DSP example programs in source form with CCStudio project files
- DSP-BIOS device drivers and C++ classes to simplify your development
- Host side software Armada, for easy data handling, processing, viewing
- Sample host applications showing message and data passing techniques
- On-line Windows help file with hypertext cross references
- One year technical support

Windows Compatible

- High-performance ring 0, 32-bit Win2K/WinXP device drivers and DLL
- Host support applets for automatic download, terminal emulation, COFF dump and flash burning

Functions

For a detailed list of Pismo functions visit www.innovative-dsp.com/pismo

Ordering Information

Pismo Toolset for Vista	53043
Pismo Toolset for Toro	53044
Pismo Toolset for Conejo	53046
Pismo Toolset for Delfin	53047
Pismo Toolset for Quixote	53048

Overview

The Pismo Toolset is a comprehensive, state-of-the-art collection of software tools and libraries used in the development of applications for our Matador and Velocia Series of DSP boards. Pismo covers all the aspects of a new DSP project:

- Simple SW installation under Windows
- High-performance board device drivers
- Full support of Code Composer Studio including example project files
- Target specific and board specific libraries, including C++ classes
- DSP-BIOS II compliant peripheral device drivers
- Numerous example programs in source form, utilizing each peripheral at full bandwidth
- Valuable host side software for easy and efficient data streaming and processing/viewing, with example programs
- One year tech support

Target Side Programming/Downloading/Debugging

Pismo is used in conjunction with Code Composer Studio, Texas Instrument's integrated development environment, for code editing, compiling/linking, downloading and hardware-assisted debugging via JTAG. Downloading is also possible over the PCI bus (without JTAG), simply by running our Download utility. All features of CCStudio are supported and well utilized whenever possible: DSP-BIOS II with our board specific peripheral device drivers, extensive use of the Chip Support Library, and full bandwidth RTDX support. Pismo ships with a large library of carefully crafted C++ class libraries, which substantially mitigate the complexity of programming the 6711/6414 DSP hardware and facilitate use of the BIOS operating system. Numerous example programs are provided with their entire project file set and source code that illustrate the use of board-specific functions, peripheral device-drivers and communication techniques with the host program. In these examples, we demonstrate the typical usage of the board peripherals at full bandwidth and the code may serve as a starting point to the developer. Bus-mastered data transfers and asynchronous mailbox messaging are well documented and illustrated.

Host Side Programming

The host side support in our Pismo toolset goes well beyond providing a simple API through a DLL. While DLL functions are still accessible and well documented in an on-line help, the Pismo toolset includes powerful C++ classes that greatly facilitate the control of - and the communication with - the target board from a Windows application. Software components - or classes - feature properties, methods and events to easily control the board, to download programs onto its target DSP and the means of communicating with the target via busmaster streaming or message communications, once target code is running. These functions are easily called and utilized from within Visual C++ or C++ Builder and shields the user from having to fully understand the inner details of the device-driver and hardware.

Armada – Simple but powerful data streaming, processing, logging

Pismo also includes a complete set of data streaming, viewing and analysis components, called Armada, which gives the means to significantly accelerate host side development. Armada is a powerful, feature-rich and fully integrated development toolset which adds high-performance data movement and processing to Borland C++ Builder or Microsoft Visual C++ applications with a powerful suite of visual components (VCL libraries under Borland Builder and MFC classes under VC++). These components offer excellent means to application engineers for integrating sophisticated data handling, graphing, processing and logging into any Windows application with almost no code writing! Custom processing algorithms are also easily integrated, using OpenWire, a “data-passing” format that greatly simplifies coding and preserves CPU bandwidth. Armada was developed for real-time acquisition systems with all emphasis placed on minimizing PC resource usage in order to deliver the highest performance data streaming achievable on a desktop or industrial PC. It integrates Intel NSP libraries of MMX and SIMD optimized DSP functions like Bandpass, FIR, IIR filters and FFT. Application developers can now take advantage of this collection of tools too, and really focus on the particulars of their project rather than waste time on how to optimize data transfers or re-inventing graphs. Example programs using the Armada components under Borland Builder or MSVC++ are included with source code. Please refer to the Armada section for a more details.

Support Utility Applets

Pismo includes a set of practical support utilities that greatly simplifies repetitive tasks during development and even after deployment.

Download

The Download applet is a useful utility to automate downloading and launching of target program. This is convenient when Code Composer Studio is no longer required or installed on the final system, for instance in field deployment or subsequent program revisions.

UniTerminal

UniTerminal is a terminal emulator providing standard I/O functionality between the DSP and a host PC to accommodate intuitive console I/O, file I/O and real-time graphics. An extremely convenient tool, that allows developers to immediately focus on their application code, UniTerminal may be used either stand-alone or in conjunction with Code Composer.

RtdxEventLog

This utility sets up the RTDX interface to view target test string messages generated in real-time by DSP applications, under Code Composer Studio.

VsProm

This flash programming utility is provided to automate the update of logic firmware on the board.

ReserveMemDsp

User can force the operating system to allocate contiguous, non-paged memory for use by the board during bus mastering.

CoffDump

Developers use this applet to quickly display sizes and addresses of all sections of a COFF output file.

BinView

BinView is a data display tools specifically conceived to allow simplified viewing of binary data stored in disk files or shared DSP memory. It offers powerful time- and frequency-domain display and analysis tools with practical scroll/zoom buttons and measuring cursors for a fast analysis of data sets.

DSP/BIOS Support and Extensions - Built Right Into Pismo

DSP/BIOS is a powerful, dedicated real-time operating system than runs directly on the C6000 DSP to manage all DSP resources, including tasks (threads), peripherals and board peripheral devices. BIOS is provided within Code Composer Studio, bundled with all Innovative Integration DSP DevPacks, and applications built atop BIOS may be deployed royalty-free.

DSP/BIOS provides a software infrastructure which allows application programmers to:

1. Develop high-performance, prioritized, multithreaded DSP applications
2. Easily communicate between board-level peripherals and tasks at full bandwidth
3. Monitor the behavior of DSP application software in real-time during development

DSP/BIOS services include: interrupt handling, scheduling of multi-tasks with user-specified priorities, management of semaphores, mailboxes and queues, data pipes and streams, runtime event logging and statistics that are viewable from CCStudio via JTAG. The developer chooses which services to use in the BIOS configuration tools. Only the kernel objects required for these services are then compiled and built into the final executable file. It is important to note that BIOS has been engineered to take advantage of the advanced architectural features of the TI ' C6000 DSP family. As a result, CPU overhead and memory usage associated with this kernel is exceptional and negligible for most applications.

While it is possible to develop simple DSP applications in the absence of BIOS, the features of BIOS make the authoring of such applications even simpler. The resources provided by BIOS allow application code to be factored and operated as autonomous, more-readily understandable subunits, reinforcing modularity and maintainability. BIOS supports creation of more capable, sophisticated applications without burdening the application programmer with unwanted complexity. For example, a typical DSP application usually involves functions to perform data acquisition, data analysis and command processing. These routines must operate at different priorities in order to guarantee meeting all real-time constraints. In a non-BIOS application, the data acquisition function would probably run as an interrupt service routine while the data analysis and command processing functions would be combined and handled within the mainline. This division of labor is a direct result of the absence of multitasking support. By contrast, a DSP/BIOS-based solution to the above application could simply utilize three separate tasks to perform these three functions, each operating at distinct priority levels. Once familiarized with the multiprogramming capabilities of BIOS, most programmers find it far more intuitive and natural as well as more powerful.

Pismo DSP/BIOS Support

In the absence of BIOS drivers, application programmers are forced to deal with peripheral devices in an explicit and direct manner. Generally speaking, vendor-supplied library API functions are called within interrupt service routines or within the mainline in order to initialize peripheral devices and to manage real-time data flow. Consequently, application programs are littered throughout with peripheral device-specific details. This effectively reduces program readability and maintainability.

BIOS features use of device drivers to support access to board-specific peripherals, such as A/Ds, D/As, PCI interface, serial ports, etc. When coupled with well-crafted peripheral device drivers supplied by a DSP board vendor such as Innovative Integration, the model imposed by DSP/BIOS has significant advantages.

Innovative Integration supports and extends each of the features of DSP/BIOS on the Matador and Velocia Series of DSP boards through a seamless integration of advanced C++ class libraries, BIOS-compliant DSP peripheral device drivers, and clear, illustrative examples. These software components are included in the Pismo Toolsets. The device drivers fully exploit the available DMA channels in the C6711 and C6414 chips so that hardware interrupt rate rarely exceeds one kHz! The net effect is that virtually all of the bandwidth of the CPU is available for application processing.

Due to the relative complexity involved in programming DSP DMA channels compared to using CPU interrupt handlers for data movement, most application programmers simply avoid use of DMA entirely, resulting in highly inefficient use of CPU computational resources. With BIOS-compliant device drivers at hand for every peripheral, the developer may focus exclusively on the end-application, rather than the myriad details involved in peripheral setup, initialization and servicing. Using the Pismo drivers insures maximal CPU availability for application use. For example, the adjacent code fragment illustrates all of the steps necessary to fully initialize and stream a sine wave to the audio output codec present on the our Vista board at 44.1 kHz

```
//-----  
// IIMain() -- Generate a waveform on both audio channels  
//-----  
  
void IIMain()  
{  
    volatile bool run = true;  
    const int BufSize = 0x1000;  
  
    Stream Aout("/AudioTee/AudioOut", smOutput, BufSize);  
    Aout.Open();  
  
    //AnalogPair Sample;  
    Stream Ain("/SineAin", smInput, BufSize);  
    Ain.Open();  
  
    // Start the Dds  
    Aout.Control(dcSetSampleRate, 44100);  
  
    // Generate waveform, send to D/As  
    while(run)  
        Aout.PutFrom(Ain);  
  
    Ain.Close();  
    Aout.Close();  
}
```

In addition to minimal processor loading, automatic DMA configuration and optimal bus utilization, the Pismo BIOS drivers support efficient cooperation in multitasking applications. For example, the call to PutFrom() within the IIMain() function will efficiently block until data is available from the Ain streaming device, allowing other tasks within the application to execute.

Pismo C++ Class Libraries

Pismo ships with a large library of carefully crafted C++ class libraries, which substantially mitigate the complexity of programming the 6711/6414 DSP hardware and facilitate using the BIOS operating system. Pismo provides C++ extensive support for:

- Dynamic creation and runtime control of tasks
- Simplified management of and access to all TI Chip Support Library (CSL) and BIOS API functions: Semaphores, Mutexes, Mailboxes, Timers, EDMA, QDMA, Atoms, McBSP, Timebases, Counters, etc.
- Data exchange using RTDX
- Streaming I/O
- Foundation (base) classes for DMA-driven device driver development
- Templated queues
- Partial standard-template library functionality via STLPort

For example, the code fragment to the right uses the Pismo `IntBuffer` class to initialize a QDMA to perform a memory-to-memory move of a constant value (0) into a 4096-word buffer (at `Src`), then to copy the source buffer (`Src`) to the destination buffer (`Dst`).

```
// Create a source buffer of 0x1000 integers
IntBuffer Src(0x1000);
// Initialize the source buffer with zeros
Src.Set(0);
// Create a destination buffer of 0x1000 integers
IntBuffer Dst(0x1000);
Dst.Copy(Src);
```

By like manner, peripheral-specific class libraries dramatically simplify access to board-specific peripheral features. For example, the adjacent code fragment illustrates real-time processing and display of NTSC/PAL video images running on the Vista DSP board.

```
Rect r(360, 525);
Rect r1(Point(2, 10), 320, 480);

while (1)
{
    // Capture next video image
    VideoIn.Get();

    // CanvasIn maps a VideoBuffer to the acquired image
    VideoBuffer CanvasIn(VideoIn.Buffer().Addr(), r1);
    // CanvasOut maps a VideoBuffer to the generated image
    VideoBuffer CanvasOut(VideoOut.Buffer().Addr(), r);

    // Interlace the acquired data into a scratch video buffer
    Scratch.Interlace(CanvasIn);

    // Draw a rectangle around the region of interest
    Rect re(Point(3, 37), Point(357, 517));
    Scratch.Rectangle(re);
    // Draw a crosshair to segment the image
    Scratch.Line(re.TL(), re.BR());
    Scratch.Line(re.BL(), re.TR());

    // Interlace the resultant image in preparation for display
    CanvasOut.Deinterlace(Scratch);

    // Send the image to the video monitor
    VideoOut.Put();
}
```

Even for C++ beginners, the code is quite clear and understandable. One of the benefits of using C++ is that while it helps to mitigate and manage complexity to support creation of larger, more sophisticated applications, it is often simply used as a “better” dialect of the C language. One may freely intermix calls to legacy ‘C’ functions, newly-written C functions, assembler functions and C++ functions (called methods) within C++ programs. The user does not need to fully understand all of the enhanced capabilities and features of C++ to fully exploit the features of the class libraries provided in Pismo.

Speed Up Development and Utilize Full DSP Bandwidth

Innovative Integration’s Pismo libraries allow the developer to reap all the benefits of DSP/BIOS and fully exploit the features of the Matador and Velocia boards. The functionality of each board peripheral has been encapsulated in a device driver that can readily be controlled within DSP/BIOS applications, including PCI interface, analog I/O, external bus and memory, serial ports and other I/O devices.

These BIOS device drivers expose all the necessary parameters needed to efficiently control each function of the peripherals. Any peripheral board resource may be instantiated, configured and shared among program tasks. The device drivers also take care of assigning default values for unspecified or non-critical parameters of a function. C++ is used as the foundation for the Pismo libraries, but C programmers may use Pismo freely, without having to learn C++ details or C++ extensions to the C language. The C++ libraries provided in Pismo are far more capable, complete and easy-to-use than any previous generation of DSP peripheral support libraries.

Illustrative, real-time example programs are included in the software suite along with complete project files and DSP/BIOS modules. The examples act as a springboard for the development of custom, high-performance DSP application programs.