Technical notes on using Analog Devices DSPs, processors and development tools
Visit our Web resources http://www.analog.com/ee-notes and http://www.analog.com/processors or e-mail processor.support@analog.com or processor.tools.support@analog.com for technical support.

VisualDSP++® Flash Programmer API for Blackfin® Processors

Contributed by Igor Likhotkin *Rev 1 – December 14, 2006*

Introduction

Embedded application testing usually involves using a JTAG emulator to debug an executable that has been loaded onto a target processor. Toward the end of the development process, it becomes critically important to test application code that has been booted into the processor from flash, PROM, or another peripheral device. Often, this software development phase is difficult to fully automate. VisualDSP++® features an additional set of API functions that facilitate the uploading (burning) of executable code and data image files into flash memory. Using the new API functions, application images can be programmed into flash, and flash contents can be filled or erased.

Hardware/Software Requirements

The Flash Programmer plug-in API interface requires VisualDSP++ Development Tools version 4.5 or higher. In addition, a suitable hardware board is needed in order to test the loading of bootable application images onto flash devices. The ADSP-BF533 EZ-KIT Lite® and the ADSP-BF537 EZ-KIT Lite evaluation systems were used to construct and test the example code described by this EE-Note. However, the Flash Programmer API supports all Blackfin® processors.

New API Functions

The API for the Flash Programmer plug-in provides a relatively small subset of functions:

- EraseAll – erases the entire contents of the flash device
- EraseBlock – erases the specified (numeric) sector of the flash device
- FillFlash – fills the flash memory with the specified number pattern
- LoadDriver – downloads the flash device driver program (`.dxe`) onto the processor, which facilitates loading the flash image onto the flash chip
- LoadFile – uploads the bootable application image into the flash device

The Flash Programmer API is part of the general VisualDSP++ API, which enables users to create scripts or programs in any language that supports Automation.

An example script (`programFlash.vbs`[1]) has been written to exercise the entire Flash Programmer API. All of the code samples and commands below refer to the example script.

Typical Use of Flash Programmer API

The most common use for the new API functions is to upload an embedded application image file into flash memory on a target board as part of a large test suite. To accomplish this, scripts can be written in a variety of Automation-enabled languages.

The example script performs each of the following tasks:

- Connects to the VisualDSP++ IDDE object
- Retrieves a handle on the Flash Programmer plug-in object
- Loads the flash chip driver program onto the target board processor
- Erases all (or specified) flash sectors
- Fills the flash memory range with a user-specified value
- Uploads the selected application image into flash memory
- Optionally sends a soft reset (boot up) signal to the processor

The example Automation script requires that the VisualDSP++ IDDE is open and that an appropriate hardware debug session is established. To automate session creation in the script, refer to VisualDSP++ online Help, which includes the complete VisualDSP++ Automation interface reference.

Connecting to IDDE and Obtaining Flash Programmer Handle

Implementing the above steps is a fairly straightforward process with the VisualDSP++ API. The flash-related part of the example script begins with obtaining the handle to the Flash Programmer plug-in object, as shown in Listing 1.

```
'...
WScript.StdOut.WriteLine "Starting the FlashProgrammer plugin..."
Set FPPlugin        = app.PluginList.Item("Flash Programmer")
Set FlashProgrammer = FPPlugin.Open()
'...
```

Listing 1. Script to Obtain Handle to the Flash Programmer Plug-In Object

Loading the Flash Driver

Next, loading the flash driver into the target processor is accomplished in a single line:

```
FlashProgrammer.LoadDriver FPDriver
```

Since the script works with the ADSP-BF533 EZ-KIT Lite, the flash driver was provided by Analog Devices. The Flash Programmer plug-in, however, can communicate with custom flash drivers for various memory chips.

Erasing Flash

The script proceeds by checking whether or not an erase option has been selected, as shown in Listing 2.

```
If ( doEraseAll = True ) Then
    WScript.StdOut.WriteLine "Erasing all flash..."
    FlashProgrammer.EraseAll()
ElseIf ( doEraseBlocks = True ) Then
    WScript.StdOut.WriteLine "Erasing Flash blocks..."
    For i = 0 To UBound ( blocks )
         strBlock = blocks(i)
         WScript.StdOut.WriteLine "Erasing block " + strBlock + "..."
         FlashProgrammer.EraseBlock CInt ( strBlock )
    Next
End If
```

Listing 2. Script to Check for Erase Options

Note that the script provides two ways to erase flash: block-by-block, and entire flash scrub. Erasing the entire flash is accomplished with a single command line, and a block-by-block erase is accomplished with a simple for-loop.

Filling Flash

Similar to the previous operations, filling flash with values is performed with a single line of code. The next part of the script checks for any fill option that may have been selected and performs the actual fill, as shown in Listing 3.

```
If ( doFill = True ) Then
    If ( fillCount < 0 ) Then
         fillCount = 16
    End If
    WScript.StdOut.WriteLine "Filling flash: count " + CStr ( fillCount ) + "..."
    FlashProgrammer.FillFlash fillOffset, fillCount, fillStride, fillValue
End If
```

Listing 3. Script to Fill Flash

Uploading of the Application Image

Similarly, uploading the application image is performed with a single line of API code, as shown in Listing 4.

```
If ( doLoadImage = True ) Then
    WScript.StdOut.WriteLine "Programming Flash with the image File..."
    FlashProgrammer.LoadFile LdrImage, imageFormat, doVerify, eraseOption,
imageOffset
End If
```

Listing 4. Script to Upload Application Image

Booting the Processor

At this point, the script's job is almost complete. The last step, which is optional, sends a boot-up signal to the processor, as shown in Listing 5.

```
If ( doBoot = True ) Then
    WScript.StdOut.WriteLine "Booting up the board from flash..."
    While ( app.ActiveSession.ActiveProcessor.BreakpointList.Count > 0 ):
         app.ActiveSession.ActiveProcessor.BreakpointList.RemoveBreakpointByIndex(0)
    WEnd
    app.MenuManager.ClickMenuItem "Settings:Boot Load...", False
    WScript.Sleep ( 1000 )
End If
```

Listing 5. Script to Send Boot Signal to Processor

The code above is a bit tricky. Prior to booting the processor, the script removes breakpoints that may have been introduced by the flash chip driver program. This is to ensure that the target runs without any interruptions. If the target must be halted at a particular symbol or address, the necessary breakpoints can be introduced at this point using the VisualDSP++ API.

Sending a boot-up signal to the processor is available only with Debug Agent connections to target boards (i.e., by using an EZ-KIT Lite board). ICE emulator connections do not expose this functionality. Figure 1 shows the connections.

Finally, the last piece of code in the script waits one second to give the processor enough time to bootstrap itself and to run a good chunk of the application code. If desired, a more precise method of controlling the run (i.e., breakpoints) of the application can be implemented with the full suite of the VisualDSP++ Automation API.

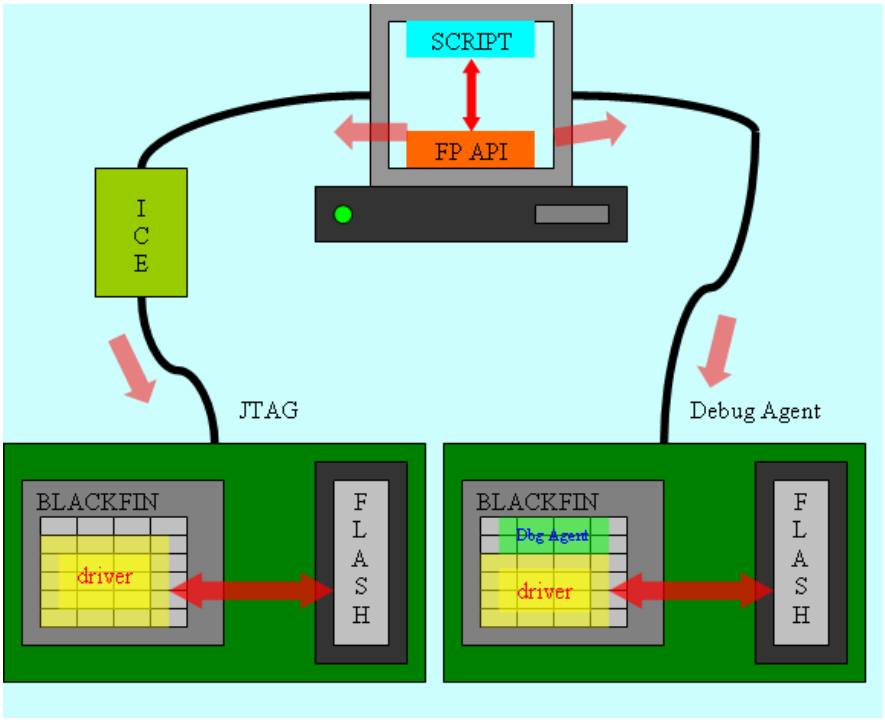

Figure 1: Communication Path Between Host PC Script and Flash Chip Via Flash Programmer API

Running the Script

The example script also contains usage information that displays when the script is run with no command-line options. Following is a typical command-line sequence that burns an application image file into flash.

```
C:\>cscript.exe programFlash.vbs --driver "BF537EzFlashDriver.dxe" --image
"Blink.ldr" –eraseAll --format 0 --boot
```

Note that the example script requires the absolute paths of the driver and the image file. Also, the script requires that the IDDE is open and is connected to the correct hardware debug session.

To fill flash with specified values, the sample script can be run as follows:

```
C:\>cscript.exe programFlash.vbs --driver "BF537EzFlashDriver.dxe" --eraseBlocks 0,1
--fill --value 0xabcd --stride 1 --count 0x100 --offsetF 0
```

When filling flash or uploading the image file, you must erase the affected flash sectors. In both of the examples above, the entire flash (or first two sectors) are erased before any fills or image uploads take place.

Beyond Basics

The beauty of the Flash Programmer automation API is that it exposes the flash control and processor initialization process for debugging. With the aid of this new API, flash can be preloaded with known values in order to check the results of the processor's boot up. In the same way, flash can be used as storage for library routines and large data tables (pre-built as flash image files), which can be loaded and erased while testing the main program.

The Flash Programmer API works with ICE (emulator) or Debug Agent (EZ-KIT Lite board) connections. This flexibility and ease of use allow for simplified solutions and quicker turnaround times. For instance, an engineer troubleshooting a malfunctioning device can use a simple script to reprogram its flash-based firmware.

Another advantage of the Flash Programmer API is that it can work with custom hardware setups and specialized flash chips. Since the flash driver is nothing more than a program built for the target processor, custom drivers can be created for a variety of flash memory chips using VisualDSP++ tools.

Refer to the Help topics in VisualDSP++ online Help under `Graphical Environment` > `Emulation Tools` > `Flash Programmer` for more information on writing custom flash drivers conformant with the Flash Programmer API framework. Also, look for Flash Programmer driver examples within the VisualDSP++ distribution for additional help with code samples.

Appendix A

Below is the listing of the entire flash programming script.

```
cscript.exe --driver <driver file> [OPTIONS]
REQUIRED:

--driver <driver file>  : specifies <driver file> for communication with the flash
                          device

OPTIONS:
--image <image file>    : specifies <image file> for loading into flash
--offsetL [addr]        : address offset to use when executing image file load
                         (default is 0x0)
--format                : format of the image file; 0 - Intel Hex (default), 1 -
                          binary, 2 - ASCII
--verifyWrites          : verify write operations during the image file load
                          (default is false)
--eraseOption           : option to clear the flash area where the image load is
                          occurring; 0 - erase affected (default), 1 - erase all,
                          2 - erase none
--fill                  : fill flash with values
--boot                  : boot up the hardware board with the image in flash
--offsetF [addr]        : address offset to use when executing fill (default is
                          0x0)
--count [n]             : number of values to write during fill operation (default
                          is 16)
--stride [s]            : offset of the next write during fill operation (default
                          is 1)
--value [val]           : value to be written into flash during fill (default is
                          0x0)
--eraseAll              : erase entire flash
--eraseBlocks [0,1,..]  : erase specified sectors of flash (comma-separated list
                          with no spaces); if sector list is omitted, the default
                          is 0,1,2,3
```

References

[1] *Associated ZIP File. Rev* 1, December 2006. Analog Devices, Inc.

Document History

Revision	Description
Rev 1 – December 14, 2006 *by Igor Likhotkin*	Initial Release.