Using the ADSP-BF592 Blackfin® Processor Tools Utility ROM

Contributed by Andy Kettler

Rev 2 – March 16, 2011

Introduction

The ADSP-BF592 Blackfin® processor provides fast, low latency on-chip memory that includes 64K bytes of L1 instruction ROM, 32K bytes of L1 instruction SRAM, and 32K bytes of L1 data SRAM. The processor has no L2 internal memory, and it does not provide an external memory interface (L3 memory). Therefore an application might be constrained by the availability of L1 instruction SRAM. One way to address this would be to take advantage of the on-chip Tools Utility ROM.

This EE-Note describes the Tools Utility ROM, it explains how to use it, and the implications involved. Also, this document assumes that applications are built using the VisualDSP++® 5.0 development tools (Update 9 or later revisions).

For compatibility with the previous release of the product, the VisualDSP++ development tools refer to the ADSP-BF592 processor as the ADSP-BF592-A processor.

Overview

The Tools Utility ROM is available with the ADSP-BF592 processor (in all silicon revisions except revision 0.0) and uses its 64K bytes of L1 instruction ROM to provide:

- Compiler support functions, which are hidden routines that the compiler knows about and relies on to provide run-time services for the code that it generates (such as emulation of floating-point arithmetic that is not supported by the hardware instruction set).
- A set of C and DSP run-time library functions including, for example, functions from `string.h` and `math.h`, matrix manipulation functions, and filter and FFT functions. Appendix A has a detailed description of the contents of the ROM.
- The core of the VisualDSP++ Kernel (VDK) referred to here as TMK.

Benefits of Using the Tools Utility ROM

The basic memory layout of the ADSP-BF592 processor is:

L1 instruction SRAM	32K bytes
L1 instruction ROM	64K bytes
L1 data SRAM	32K bytes
L1 scratchpad SRAM	4K bytes

There is no support for external memory, and so exploiting the Tools Utility ROM may overcome shortages in L1 instruction SRAM (refer to the following section which describes the ROM's L1 data SRAM requirements). For example, the linker knows that the Tools Utility ROM contains a copy of the C library function `strcmp`. So if an application calls that function then all the linker has to do is to redirect all appropriate references to where the `strcmp` function is located in the ROM. Should an application decide not to link against the contents of the Tools Utility ROM, then the linker would have to map the library-defined version of `strcmp` into L1 instruction SRAM, thus increasing the memory footprint of the application.

Clearly the extent to which an application may benefit from using the ROM will depend on how much of the code in the ROM it can exploit. However, most C and C++ based applications will rely on compiler support functions, and all multi-threaded applications that use VDK will require the VDK core.

Drawbacks of Using the Tools Utility ROM

It is important to note that there is an L1 data SRAM overhead that must be accepted if an application is linked against the contents of the Tools Utility ROM. This overhead amounts to 3.6K bytes of L1 data SRAM that is automatically reserved for the VDK core and library functions in the ROM by the default `.LDF` files and by any `.LDF` file that is generated by the IDDE. The overhead can be avoided by not using the Tools Utility ROM when linking an application – this is described in the section Linking with the Tools Utility ROM.

The reserved L1 data memory is set aside for static data used by the VDK core and for constant data for some of the run-time library functions. For revisions 0.1 and 0.2 of the ADSP-BF592 processor, this data is mapped between the memory addresses `0xFF800000` and `0xFF800E40`.

The Tools Utility ROM was built with components from VisualDSP++ 5.0 Update 8, and so improvements made to the library functions in later VisualDSP++ versions (such as to the performance and accuracy of the default single precision floating-point emulation functions) will not be available if linking against the Tools Utility ROM.

Contents of the Tools Utility ROM

This list summarizes the contents of the ADSP-BF592 Tools Utility ROM (for silicon revisions 0.1 and 0.2); refer to Appendix A — Contents of the ADSP-592 Tools Utility ROM for a more complete reference:

- The VDK core that uses (approximately) 2.8K bytes of instruction memory
- High-speed single and double precision floating-point emulation library (`-fast-fp`)
- Compiler support functions
- Core DSP algorithms (FFTs, filters for types `_Fract` and `fract16`)
- Other DSP algorithms (vector, matrix, and statistical functions for the `_Fract`, `fract16`, and floating-point data types)
- Functions from `math.h` for the `_Fract`, `fract16`, and floating-point data types
- Functions to support the 16-bit based fractional complex types, and the single precision and double precision floating-point complex types (`complex.h`)
- Selection of ETSI functions
- C run-time functions declared in `string.h` - except for `strtok` and string functions that rely on the current locale
- `setjmp`, `longjmp` (`setjmp.h`)
- `clip`, `countones`, `max`, `min`, and `div` (`stdlib.h`)

The following is a summary of what is *not* included in the Tools Utility ROM:

- I/O library (`stdio.h`)
- Fully IEEE-compliant single and double precision floating point emulation support (enabled by the compiler switch `-ieee-fp`)
- Window generators for the fractional data types (`window.h`)

- Twiddle table generators for the fractional data types (`filter.h`)
- Support for date and time (`time.h`)
- Wide character support
- Heap management functions (such as `malloc`, `free`)
- Locale control (`locale.h`)
- `qsort`, `bsearch`
- DSP algorithms for the `long _Fract` and `fract32` data types
- C++ libraries
- Middleware

The functions in the Tools Utility ROM contain all the workarounds for silicon anomalies that are available with VisualDSP++ 5.0 Update 9, with the exception that the support for the VDK core does not have a workaround for silicon errata 05000494[3]: "*EXCPT Instruction May Be Lost If NMI Happens Simultaneously*". Note that this anomaly applies to all current Blackfin parts and revisions; check the most recent silicon Anomaly List for your part:

`http://www.analog.com/en/embedded-processing-dsp/blackfin/processors/ic-anomalies/resources/index.html`

All the run-time library functions within the Tools Utility ROM are re-entrant - that is, they are interruptible and they may also be used in a multi-threaded environment.

Linking with the Tools Utility ROM

Unless specified otherwise, linking against the Tools Utility ROM is controlled by the silicon revision as shown in Table 1.

Silicon Revision	Link with the ROM	Don't Link with the ROM
any	N/A	default
none	N/A	default
0.0	N/A	default
0.1	-utility-rom	default
0.2	default	-no-utility-rom

Table 1. Linking with the Tools Utility ROM

If no silicon revision is specified, then the VisualDSP++ tools will assume a default silicon revision based on the current processor, which for VisualDSP++ 5.0 Update 9 and an ADSP-BF592 processor is silicon revision 0.1 (to discover the default silicon revision for your version of VisualDSP++, open up the file `ADSP-BF592-A-compiler.xml` that is located under your VisualDSP++ installation in the sub-directory `System\ArchDef`).

Table 1 shows that when using silicon revision 0.2, the default is to link against the Tools Utility ROM and for all earlier revisions the default is to ignore the ROM. The default action can be overridden provided that the silicon revision is not one of `any` or `none` or `0.0`

The compiler driver `ccblkfn` supports two switches that can be used to override (or confirm) the default action. These switches are:

- `-utility-rom`
- `-no-utility-rom`

As expected, the switch `-utility-rom` specifies that the linker should make use of the Tools Utility ROM; and you can avoid linking against the ROM by using the switch `-no-utility-rom`.

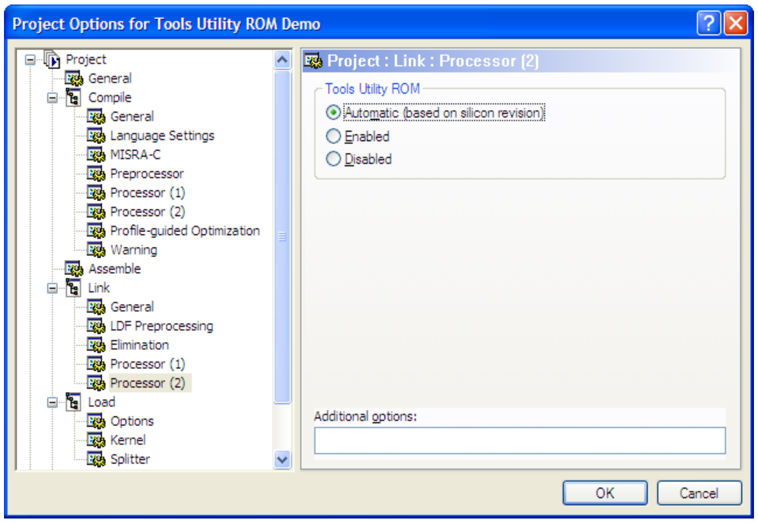
If an application is built via the IDDE's project management system then you can control the use of the Tools Utility ROM via the project options.

- Navigate to the page `Project Options -> Link -> Processor (2)`
- The page contains a `Tools Utility ROM` area with three alternative buttons:

- Automatic (based on silicon revision)
- Enabled
- Disabled

If you click on each button and then use Help (SHIFT+F1), you will get a brief explanation of what the button does. For example, the Help information for the Enabled button says "*Link your application against the Tools Utility ROM. Equivalent to the compiler's -utility-rom command-line switch.*"

Figure 1 shows the Link -> Processor (2) page.


Project Options for Tools Utility ROM Demo

Project : Link : Processor (2)

Tools Utility ROM

[✓ Automatic (based on silicon revision)] Enabled Disabled

Additional options:

Figure 1. Screenshot of the Link -> Processor (2) Project Options page

Modifying a Customized LDF

This section describes the changes required to a customized .LDF file if you want your application to link against the contents of the Tools Utility ROM. The changes required are:

- The customized LDF must define the default memory section name MEM_L1_DATA to cover the available L1 data SRAM. The name is required by the LDF fragment ADSP-BF592-A-LDF.h, which the customized LDF will #include (see below). Listing 1 shows how the default LDF defines the memory section name; refer to the default VDK .LDF file for VDK-based applications.

```
MEMORY
{
   MEM_L1_SCRATCH       { START(0xFFB00000) END(0xFFB00FFF) TYPE(RAM) WIDTH(8) }
   MEM_L1_CODE          { START(0xFFA00000) END(0xFFA07FFF) TYPE(RAM) WIDTH(8) }
#if defined(IDDE_ARGS)
#define ARGV_START 0xFF807EB0
   MEM_L1_DATA          { START(0xFF800000) END(0xFF807EAF) TYPE(RAM) WIDTH(8) }
   MEM_ARGV             { START(0xFF807EB0) END(0xFF807FFF) TYPE(RAM) WIDTH(8) }
#else
   MEM_L1_DATA          { START(0xFF800000) END(0xFF807FFF) TYPE(RAM) WIDTH(8) }
#endif
} /* MEMORY */
```

Listing 1. Example of defining MEM_L1_DATA

- #include the LDF fragment ADSP-BF592-A-LDF.h, which defines the location of each function in the Tools Utility ROM. As an example, refer to Listing 2, which is an extract based on the default .LDF file and shows where to include the fragment.
- Add the file romdata-BF592-A.doj to the list of object files to be included in the application; this object file defines the L1 SRAM data that is required by the Tools Utility ROM. If the application is using VDK, then add the file romdata-BF592-A-TMK.doj instead. Listing 3 is another extract based on the default .LDF file and shows how the LDF ensures that the file romdata-BF592-A.doj is included in the application.

```
PROCESSOR p0
{
  OUTPUT( $COMMAND_LINE_OUTPUT_FILE )

  /* Following address must match the reset PC address */
  RESOLVE(start,0xFFA00000)

#if defined(IDDE_ARGS)
  RESOLVE(___argv_string, ARGV_START)
#endif

  KEEP(start,_main)

#include "ADSP-BF592-A-LDF.h"
```

Listing 2. Example of #including ADSP-BF592-A-LDF.h

```
/*
** define linked objects list
*/
$OBJECTS =
   CRT,                         /* C startup object                 */
   $COMMAND_LINE_OBJECTS ,      /* defined by linker */
#if defined(USE_PROFILER0)      /* Profiling initialization funcs.  */
   RT_OBJ_NAME(prfflg0_532),
#elif defined(USE_PROFILER1)
   RT_OBJ_NAME(prfflg1_532),
#elif defined(USE_PROFILER2)
   RT_OBJ_NAME(prfflg2_532),
#endif
   __initsbsz532.doj,           /* meminit support                  */
   romdata-BF592-A.doj,         /* ROM-referenced data              */
#if defined(USER_CPLBTAB)
   USER_CPLBTAB ,               /* custom cplb configuration        */
#else
   cplbtab592-a.doj,            /* default cplb configuration       */
#endif
   RT_OBJ_NAME(crtn532)         /* CRT end object                   */
```

Listing 3. Example of how to include romdata-BF592-A.doj

Make sure that any definition of the MEM_ARGV memory section does not conflict with the L1 data SRAM for the ROM functions (that for silicon revisions 0.1 and 0.2 is allocated between the memory addresses 0xFF800000 and 0xFF800E40).

- For VDK projects, replace the library TMK-BF532.dlb with the TMK ROM definition library TMK-BF532_ROM_DEF.dlb. Listing 4 is a snapshot of the default VDK .LDF file and shows how it selects the alternative library when it is linking against the contents of the ROM.

```
$LIBRARIES =
#if !defined (USE_UTILITY_ROM)
  #ifdef _ADI_SOV_DETECTION
     TMK-BF532_sov.dlb,
  #else
      TMK-BF532.dlb,
  #endif
#else
    TMK-BF532_ROM_DEF.dlb,
#endif
    VDK_LIB_NAME_(CORE),
    VDK_LIB_NAME_(VDK_IFLAG_),
    $BASE_LIBRARIES;
```

Listing 4. Example of selecting TMK-BF592_ROM_DEF.dlb

The files ADSP-BF592-A-LDF.h, romdata-BF592-A-TMK.doj, and romdata-BF592-A.doj are specific to a particular silicon revision of the ADSP-BF592 processor, and are installed under your VisualDSP++ installation in the library revision folder:

Blackfin/lib/bf592-a_rev_<revision-id>

where <revision-id> corresponds to a silicon revision, such "0.1"

Note that an individual set of library files may cover more than one specific silicon revision and so not every silicon revision has its own library revision folder; for example, silicon revision 0.1 and 0.2 share the same libraries.

When you use either the IDDE or ccblkfn to build your application then the linker will automatically know where to find the appropriate version of the files ADSP-BF592-A-LDF.h, romdata-BF592-A-TMK.doj, and romdata-BF592-A.doj. (If possible, you should avoid calling the linker directly).

One way of working out what changes to make to a customized LDF is to inspect the default (or default VDK) LDF.

Excluding Specific Functions in the ROM

One of the effects of using the Tools Utility ROM to link an application is that the linker will always choose the function in the ROM over a function with the same name that is defined in the application, or is defined in a supplied object file or library. There are occasions therefore when you may want the linker to ignore specific entry points in the ROM. This can be done by re-defining the ROM entry point as a unique symbol that is not otherwise used by the application.

For example, suppose that an application has its own local version of the cosine function cosf. The entry point for this function is __cosf. (The entry point name associated with each library function in the Tools Utility ROM is documented in the spreadsheet that is included in the associated `.ZIP` file (see Appendix A — Contents of the ADSP-592 Tools Utility ROM). If the application is being built under the IDDE, then navigate to the page `Project Options -> Link -> LDF Preprocessing`, and add:

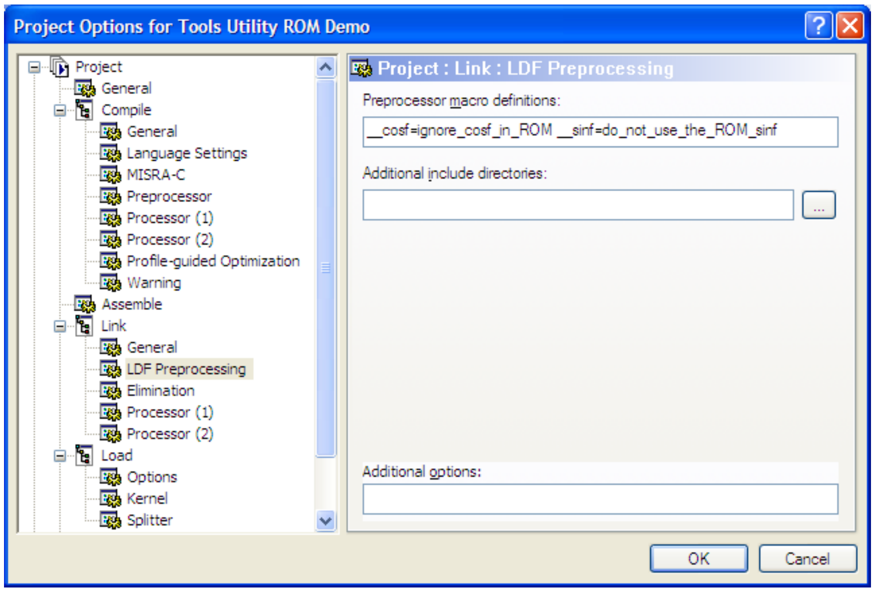
`__cosf=ignore_cosf_in_ROM`

to the box labeled `Preprocessor macro definitions`. The actual value that is assigned to the macro `__cosf` is not important - what is important is that the name is unique to the application being linked. Supply a corresponding definition for each function in the ROM that the linker should ignore.

Figure 2 is a screenshot of the Linker's `LDF Preprocessing` Project Options page and contains an example that shows that the Tools Utility ROM-based versions of the functions `cosf` and `sinf` should not be referenced by the executable that the linker will build.

If the application is being built without the IDDE, then include the switch `–flags-link -MD__cosf=ignore_cosf_in_ROM` amongst the switches used to build the application.

Do not attempt to use this method to exclude a specific ROM-based TMK function – doing so may cause the application to fail. VDK requires that TMK is linked entirely from the ROM or entirely from the VDK run-time libraries.



Preprocessor macro definitions: `__cosf=ignore_cosf_in_ROM __sinf=do_not_use_the_ROM_sinf`

Figure 2. Screenshot of the Linker's `LDF Preprocessing` Project Options page

Debugging Applications that use the ROM

You can use the IDDE to develop, test and debug your application on the ADSP-BF592 Blackfin processor. If you want to monitor and control the execution of your application in a simulated ADSP-BF592 environment, then you can use the cycle accurate ADSP-BF5xx single processor simulator and the IDDE will automatically display the entry points and contents of the Tools Utility ROM in its disassembly window. However the IDDE only directly supports the Tools Utility ROM of the default silicon revision of the ADSP-BF592 processor. Refer to Appendix B — How to Modify the Registry to Load a Different ROM Image if you want the IDDE to simulate a different version of the ROM.

The Blackfin family compiled simulator is not supported with the ADSP-BF592 processor.

If you are using an ADSP-BF592 EZ-KIT Lite evaluation board or a JTAG ICE emulator then the IDDE will display the contents of the L1 instruction ROM - but you should be aware that you can only set hardware breakpoints in the ROM if the debug target is a JTAG emulator - ordinary software breakpoints cannot be set in a non-simulated ROM.

Constraints and Limitations

This section documents restrictions that you should be aware of when using the ADSP-BF592 Tools Utility ROM.

Silicon revision any or silicon revision none

Potentially each revision of the ADSP-BF592 processor may have its own version of the Tools Utility ROM, and so the linker must be told which particular silicon revision to build an application for. This information is provided either via the IDDE's project options or via the compiler switch `-si-revision`. However, the linker will not know which version of the Tools Utility ROM to use if the revision is specified as either `any` or `none`.

Under these circumstances, the default ADSP-BF592 `.LDF` files, and the `.LDF` files generated by the IDDE, will cause the linker to ignore the Tools Utility ROM and they will output the following warning:

```
The Tools Utility ROM will not be used
to link this application because the
silicon revision does not specify a
specific revision number. This message
may be suppressed by defining the LDF
macro NO_UTILITY_ROM.
```

The above message will not appear if the LDF macro `NO_UTILITY_ROM` has been defined.

Silicon revision 0.0

Processor silicon revision 0.0 does not have a Tools Utility ROM and any attempt to use the ROM will be silently ignored.

Silicon Revision 0.1

The production version of the ADSP-BF592 processor silicon revision 0.1 does not support the Tools Utility ROM and so the default for this revision is to not use the ROM.

However, support for the ROM can be enabled for engineering samples as explained in the section Linking with the Tools Utility ROM above. An additional change will be required to enable the support for the ROM if a VDK-based application is using the default `.LDF` file from VisualDSP++ 5.0 Update 9 development tools.

The change required is to make a copy of the default VDK `.LDF` file and delete the text `"&& __SILICON_REVISION__ != 0x1"` from the following lines in the file which guards the setting of macro `USE_UTILITY_ROM`:

```
#elif __SILICON_REVISION__ != 0 \
   && __SILICON_REVISION__ != 0x1
    #define USE_UTILITY_ROM
#endif
```

VisualDSP++ simulator directly supports one version of the Tools Utility ROM

When using the simulator to debug an application that runs on an ADSP-BF592 processor, the IDDE will ensure that a copy of the latest version of the Tools Utility ROM is available to the simulator so that it can display and monitor the progress of the application whenever it executes a function in the ROM. There is currently a restriction in the VisualDSP++ tools in that it can only support the latest version of the Tools Utility ROM in this way.

If you want to use the simulator to single step and debug functions in other versions of the ROM, then you will have to manually edit the registry.

Appendix B — How to Modify the Registry to Load a Different ROM Image, describes how to make the necessary changes to the registry.

The Tools Utility ROM reserves L1 data SRAM

This has already been referred to in detail in the section Drawbacks of Using the Tools Utility ROM but is repeated here for completeness.

The linker uses L1 data SRAM of the ADSP-BF592 processor to satisfy the data requirements of the Tools Utility ROM. The data is used by some run-time library functions to define constant data, and by the VDK core to store non-constant static data. The total amount of L1 data SRAM reserved is approximately 3.6K bytes.

Debugging using hardware

Ordinary breakpoints cannot be set on instructions that are installed in ROM. The IDDE does, however, support hardware breakpoints provided that the debug target is a JTAG emulator[2].

Blackfin family compiled simulator is not available

The Blackfin family compiled simulator does not support the Tools Utility ROM and so is not supported for the ADSP-BF592 processor.

The ROM contains the -fast-fp emulation routines

The Blackfin compiler provides two alternative sets of functions for emulating IEEE floating-point arithmetic.

The default set is the high-speed emulation library that relaxes some of the IEEE floating-point standard rules for checking inputs against Not-a-Number (`NaN`) and denormalized numbers to improve performance; the alternative set is slower and occupies more L1 instruction memory but obeys all of the IEEE floating-point standard rules. The first set is the default and corresponds to the switch `-fast-fp` while the second set can be selected by using the switch `-ieee-fp`. The Tools Utility ROM can only contain one set of the floating-point emulation functions and the set selected corresponds to the `-fast-fp` switch.

If an application is built with the `-ieee-fp` switch then the corresponding emulation routines will be added to the application and will have a corresponding effect on its L1 instruction SRAM occupancy. Note however that any ROM-based run-time library function that relies on floating-point emulation will continue to use the default set of emulation routines that are installed in the ROM irrespective of the setting of the `-ieee-fp` switch.

Appendix A — Contents of the ADSP-592 Tools Utility ROM

The section Contents of the Tools Utility ROM, in the main body of this document, provides a broad overview of what is included in the Tools Utility ROM. The purpose of Appendix A is to describe the contents of the ROM in more detail.

The Tools Utility ROM for silicon revision 0.1 and 0.2 of the ADSP-BF592 Blackfin processor contains:

- 66 compiler support routines
- 21 ETSI functions
- 257 C and DSP run-time library functions
- and the VDK core, referred to in this document as TMK

Most of the Tools Utility ROM is composed of functions from the C and DSP run-time libraries, which are described in the *Blackfin Compiler and Library Manual for VisualDSP++ 5.0* [1]. The ROM also contains a selection of ETSI functions, which are also described in the manual. A complete list of library functions that are included in the ROM is available in the spreadsheet in the associated `.ZIP` file.

The spreadsheet contains the following information for each run-time library function included in the ROM:

- The prototype of the function
- The entry-point name of the function
- Associated header file
- Other relevant information

The spreadsheet does not include either the VDK core or the compiler support routines that are included in the ROM because these are hidden interfaces that are not directly callable from user source code. The compiler support routines provide several different services for compiler generated code; a large proportion of these functions provide support for floating-point arithmetic; others provide support for the integer division and modulus operators, and there are also routines for converting between different C data types. Most C and C++ applications will depend on a selection of compiler support routines. All VDK-based applications will use TMK.

The Tools Utility ROM reserves 3.6K bytes of L1 data SRAM – refer to the main body of this document for more details.

Appendix B — How to Modify the Registry to Load a Different ROM Image

When you are using the simulator to debug your application on an ADSP-BF592 processor, the simulator assumes that you are using the Tools Utility ROM for the default silicon revision of the processor (which for VisualDSP++ 5.0 Update 9 tools is revision 0.1). If this is not the version of the ROM that you intend to use, then you shall have to update your system registry so that the simulator will select an image of the ROM that corresponds to the version that you want to use.

These are the steps required:

1. Enter the system registry via the `\start\run\regedit` command.
2. Open the key `HKEY_LOCAL_MACHINE\SOFTWARE\Analog Devices`
3. If you have more than one instance of the VisualDSP++ tools installed, then find the key that corresponds to the specific installation that you want to modify. You can do this by clicking each `VisualDSP++` key until you find the version whose `Install Path` matches that of the VisualDSP++ installation that you want to modify.
4. Once you have located the appropriate `VisualDSP++` key, open the key `Tool Manager\ADSP-BF592-A` and modify the value `LoadPrerequisite.0` - the first four characters the data must be `SIM,` followed by the pathname to a `.DXE` file that corresponds to the image of the ROM that you want to simulate. For example:

   ```
   LoadPrerequisite.0 = SIM,C:\Image Of My ROM.dxe
   ```

5. Close the registry and reboot.

References

[1] *VisualDSP++ 5.0 C/C++ Compiler and Libraries Manual for Blackfin Processors*. Rev 5.4, January 2011. Analog Devices Inc.

[2] *VisualDSP++ 5.0 User Guide*. Rev 3.0, August 2007. Analog Devices Inc.

[3] *ADSP-BF592 Blackfin Anomaly List for Revisions 0.0, 0.1*. Rev A, September 2010. Analog Devices, Inc.

Document History

Revision	**Description**
Rev 2 – March 16, 2011 by A. Kettler	Updated document to add support for VisualDSP++ 5.0 Update 9. Initial public release.
Rev 1 – September 21, 2010 by A. Kettler	Initial draft release.