1 Hardware and Software Requirements for Debugging and Building DSP/BIOS Link and RF6 Applications with Code Composer Studio and a JTAG Emulator

1.1 Hardware Requirements

Figure 1 shows the typical hardware setup used for debugging Link and RF6 applications on the OMAP platform when using CCS and a JTAG emulator. Note that the stereo speakers and CD player are only required for debugging the RF6 audio application.

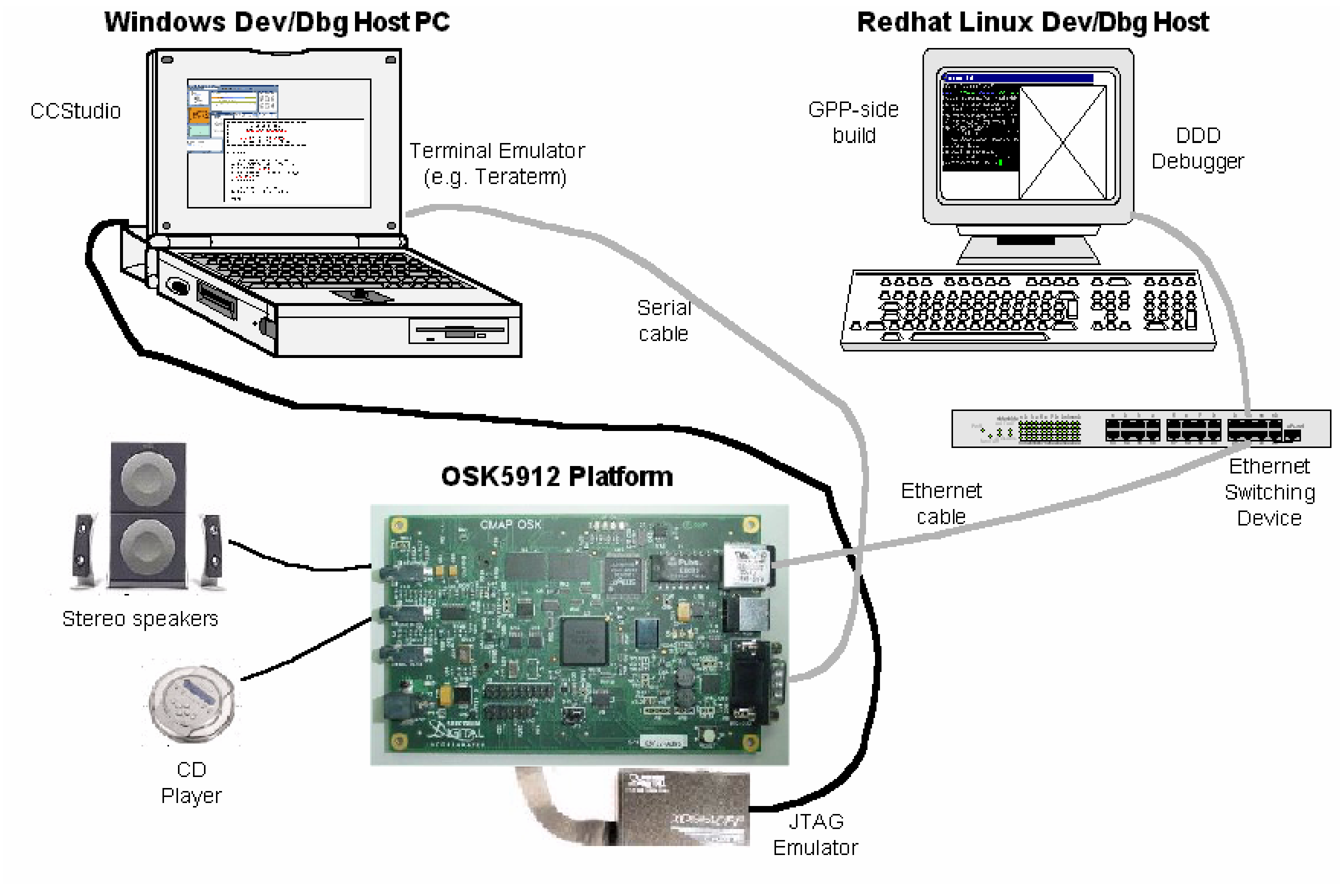


Figure 1. Hardware Device Connections

This hardware setup consists of the following platforms:

- **Windows Development/Debug Host.** This machine should be a PC running Microsoft Windows NT, 2000, or XP. It is used to build, run, and debug the DSP-side content using Code Composer Studio. The machine must have a free COM port for terminal emulation.
- **Red Hat Linux Development/Debug Host.** This may be either a dedicated Linux host workstation or a Linux Virtual machine running under Windows. It should run Red Hat Linux version 7.3 or greater. VmWare (www.vmware.com) is one such package that has been successfully used with RF6. This host is where you build the GPP-side of your RF6-based applications. The Ethernet connection allows the target board to boot up into the Linux kernel and mount its root filesystem—for example, using the Network File System (NFS). At production time, you will likely put the target filesystem, including the kernel and necessary utilities (for example, ls and pwd) into Flash memory. This enables a standalone boot, eliminating the need for Ethernet.
- **OMAP platform.**
  - **OMAP Starter Kit OSK5912.** This board contains both an ARM926EJ-S processor and a TMS320C55x DSP.

The hardware connections shown in Figure 1 are as follows:

- **JTAG Emulator (e.g. XDS510, XDS560, XDS510PP+).** Connects the DSP to the PC running Windows for connection to Code Composer Studio.
- **Ethernet cable.** Connects the target board to the Linux file system.
- **9-pin null modem serial cable.** Connects the GPP to the PC running a terminal emulator.

1.2 Software Requirements

The following software should be installed on the platforms used for RF6:

- **Windows Development/Debug Host.**
  - Microsoft Windows NT, 2000, or XP
  - Code Composer Studio for OMAP v2.21 or greater
  - A terminal emulator, for example Tera Term Pro 2.3 or greater. Tera Term is a freeware terminal emulation program with the ability to transfer files in binary format. It is available at http://hp.vector.co.jp/authors/VA002416/teraterm.html.
- **Red Hat Linux Development/Debug Host.**
  - Red Hat Linux version 7.3 or greater
  - MontaVista Linux Professional v3.1 or Montavista Preview Kit for the OSK, which includes the Linux kernel, bootloader, etc.
  - Network services such as telnetd, ftpd, and nfsd should be configured and available.
  - DSP/BIOS Link v1.10.01 or greater

The necessary header files or library components for the following software are supplied with RF6. However, for source-level debugging, we recommend that you download the full packages from TI DSPVillage.

- Driver Development Kit DDK v1.1 or greater
- Chip Support Library CSL v3.00.02 or greater
- DSP/BIOS Link v1.10.01 or greater
- MSGQ 1.01 or greater

2 Building and Debugging the DSP/BIOS Link Loop GPP and RF6 Audio Applications

The following section outlines the procedure by which one can build and debug the DSP/BIOS Link Loop GPP and RF6 audio applications. This will cover the following: configuring DSP/BIOS Link and building the Link driver, building and running the ARM and DSP sides of the Loop GPP and RF6 audio applications, and debugging the DSP side application via a Code Composer Studio window. Note that the steps listed below assume that you have installed the Montavista Preview Kit in the manner listed in the OSK Users Guide.

2.1 Installing the Software

1. On your Windows machine, install the MSGQ component of DSP/BIOS by double clicking on the **ALL-2.20-SA-to-UA-MQ-1.01.00.EXE** file stored in the /addons/MSGQ directory on the OSK5912 CD-ROM.

2. If you haven't already done so, install the DSP side sources for Link and RF6 on your windows machine by executing the **SetupOSK5912.exe** file in the /codecomposer directory of the OSK5912 CD-ROM. Note that this places the DSP side sources for Link and RF6 in the c:\ti\boards\osk5912\dsplink and c:\ti\boards\osk5912\referenceframeworks directory respectively.

3. On your Linux machine, copy the **osk_dsplinkv1.1_rf6v3.1.tar.gz** file from the /linux/rf6-link directory on the OSK5912 CD-ROM to your /home/username directory. Note that if you have installed Samba (please refer to the OSK users guide) you can do this in Windows by dragging and dropping this file from the CDROM to the /home/username directory on the Linux machine. This file contains the ARM side sources required for both DSP/BIOS Link and RF6. In the /home/username directory, type the following command (as username, not root) to extract the archive:

```
tar xvzf osk_dsplinkv1.1_rf6v3.1.tar.gz
```

2.2 Configuring DSP/BIOS Link

The build configuration for DSP/BIOS™ LINK is an interactive process. The generated configuration file is appropriately included during the build process. The build configuration depends upon the environment variable – DSPLINK.

1. Set up the DSPLINK environment variable by issuing the following command on the Linux host:

```
source ~/dsplink/etc/host/scripts/Linux/gppenv.bash
```

2. The DSP/BIOS Link build configuration can be initiated by executing the command:

```
dsplinkcfg
```

3. The first menu confirms if the environment variable DSPLINK is set correctly.

```
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
        DSP/BIOS(TM) LINK  Configuration Tool
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

DSPLINK is currently defined as:

L:\dsplink


1.  Continue.

2.  Quit to change.

::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

YOUR CHOICE :
```

4. If the environment variable DSPLINK points to a non-existent directory, the menu similar to one below is presented.

```
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
        DSP/BIOS(TM) LINK  Configuration Tool
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

DSPLINK is currently defined as:

d:\dummy


!! ERROR !! Invalid path assigned to DSPLINK!

Could not find following directories:
d:\dummy


1.  Continue.

2.  Quit to change.

::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

YOUR CHOICE :
```

5. Next menu allows user to choose the OS running on the GPP side. Select the Montavista Linux Preview Kit Edition.

```
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
        DSP/BIOS(TM) LINK  Configuration Tool
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

Choose the OS running on GPP

1.  Montavista Linux Preview Kit Edition

2.  Montavista Linux Pro 3.0

3.  Monterey Linux

4.  Nucleus

::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

YOUR CHOICE : 1
```

6. Next menu allows user to choose the target platform. Select the OMAP5912 OSK platform.

```
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
        DSP/BIOS(TM) LINK  Configuration Tool
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

Choose the target platform

1.  OMAP5912 OSK

2.  Innovator(TM) using OMAP5910

3.  DM310 with DM642 using HPI Interface

::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

YOUR CHOICE :  1
```

7. Next menu allows user to choose the variant of the target platform. If the variants are present, they are listed in the menu.

If there is no variant (or not currently supported by DSP/BIOS™ LINK), following menu appears:

```
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
        DSP/BIOS(TM) LINK  Configuration Tool
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
```

```
No variant is supported for this platform.

Press <ENTER> to continue...

::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

YOUR CHOICE :
```

8. Next menu allows user to choose the components to be included while building DSP/BIOS™ LINK. Select option 1, PROC+CHNL+MSGQ.

```
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
        DSP/BIOS(TM) LINK  Configuration Tool
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

Choose the target platform

1.  PROC + CHNL + MSGQ

2.  PROC + CHNL

3.  PROC

4.  DSP  (Minimal interface for DSP Control)

::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

YOUR CHOICE :  1
```

9. Next menu allows user to choose if the debug trace is enabled/ disabled. Select the "no" option.

```
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
        DSP/BIOS(TM) LINK  Configuration Tool
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

Enable debug trace?

0.  No

1.  Yes

::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

YOUR CHOICE : 0
```

10. Next menu allows user to choose the level of profiling information to be collected during execution. Select the "No" option.

```
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
        DSP/BIOS(TM) LINK  Configuration Tool
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
```

```
Enable profiling?

0. No

1. Yes. Basic only.

2. Yes. Detailed.

:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

YOUR CHOICE : 0
```

Based on the input values, the configuration parameters are stored in the file CURRENTCFG.MK.

2.3 Build DSP/BIOS Link ARM side sources

To build the GPP side sources, follow the steps below:

1. Set up necessary environment variables.

```
$ source ~/dsplink/etc/host/scripts/Linux/gppenv.bash
```

❑ *The above command assumes that you are using `bash` shell. If you are using `tcsh` shell, an equivalent script '`gppenv`' is shipped with the release package that can be used.*

This script sets/ modifies following environment variables:

DSPLINK	Defines the root for DSP/BIOS LINK installation.
PATH	Appends the path to include scripts provided in the installation.

❑ *This command can be included in the '.rc' file corresponding to your shell.*

2. Change to the source directory:

```
$ cd ~/dsplink/gpp/src
```

3. Start the build process:

```
$ gmake –s [debug | release]
```

❑ *The '-s' option causes gmake to build silently. Please refer to gmake documentation for other options.*

4. Upon successful completion of build, the kernel module and user library shall be created in the following directories:

```
~/dsplink/gpp/export/BIN/Linux/OMAP/DEBUG
~/dsplink/gpp/export/BIN/Linux/OMAP/RELEASE
```

2.4 Build DSP/BIOS Link DSP side sources

Build DSP side sources

To build the DSP side sources, follow the steps below:

<u>IMPORTANT</u>: You must first run <$(CCS_INSTALL_DIR)>\dosrun.bat so that the timake command and the TI Code Generation Tools are found.

1. First, copy the textual configuration files provided in the installation package of DSP/BIOS™ LINK into the CCS area (below assumes the CCS directory is in L:\ti).

```
$ copy L:\ti\boards\osk5912\dsplink\dsp\src\tcf\OMAP\*.tci
L:\ti\bin\utilities\tconf\include
```

2. Change to the `make` folder within your workspace:

```
L:\> cd ti\boards\osk5912\dsplink\dsp\make\OMAP
```

3. Start the build process:

```
L:\dsplink\dsp\make\OMAP\> timake.exe dsplink.pjt debug
```

4. Build the DSP/BIOS LINK messaging library, use the following command:

```
L:\dsplink\dsp\make\OMAP\> timake.exe dsplinkmsg.pjt debug
```

2.5 Building and Running the DSP/BIOS Link Loop GPP Example

2.5.1 Overview

This sample illustrates basic data streaming concepts in DSP/BIOS™ LINK. It transfers data between a task running on GPP and another task running on the DSP .

On the DSP side, this application illustrates use of TSK with SIO or SWI with GIO.

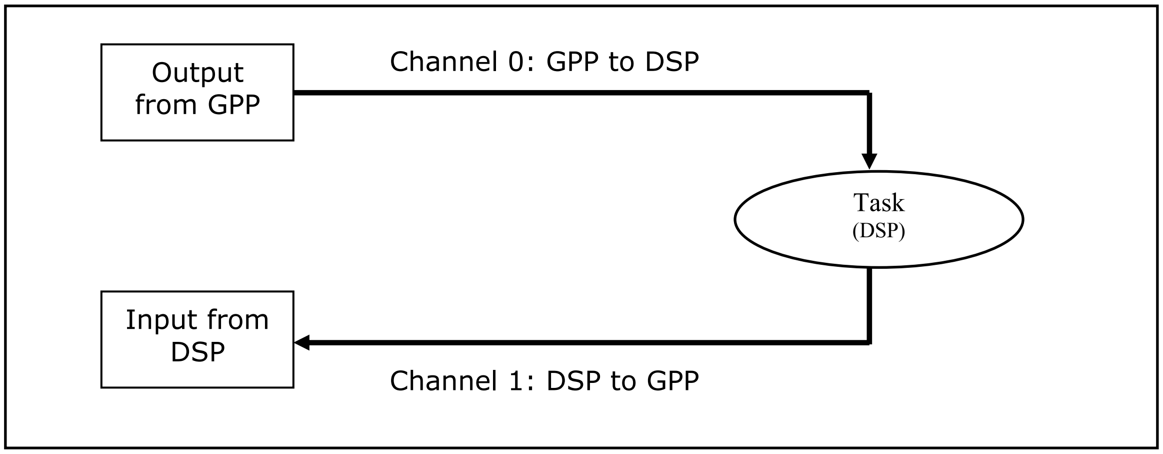


Figure 1. Data flow in the sample application – LOOP

This application illustrates a very simple data transfer scenario:

On the GPP side

INITIALIZATION

1. The client sets up the necessary data structures for accessing the DSP. It then attaches to the DSP identified by ID_PROCESSOR.
2. It loads DSP executable (`loop.out`) on the DSP.
3. It creates channels CHNL_ID_INPUT and CHNL_ID_OUTPUT for data transfer.
4. It allocates and primes one buffer each of specified size for data transfer on these channels.

EXECUTION

1. The client starts the execution on DSP.
2. It fills the output buffer with sample data.
3. It then issues the buffer on CHNL_ID_OUTPUT and waits to reclaim it. The reclaim is specified to wait forever.
4. The completion of reclaim operation indicates that the buffer has been transferred across the physical link.
5. It issues an empty buffer on CHNL_ID_INPUT and waits to reclaim it. The reclaim is specified to wait forever.
6. Once the buffer is reclaimed, its contents are compared with those of the buffer issued on CHNL_ID_OUTPUT. Since this is a loop back application the contents should be same.
7. The client repeats the steps 3 through 6 for number of times specified by the user.
8. It stops the DSP execution.

FINALIZATION

1. The client frees the buffers allocated for data transfer.
2. It deletes the channels CHNL_ID_INPUT and CHNL_ID_OUTPUT.
3. It detaches itself from ID_PROCESSOR and destroys the PROC component.

On the DSP side

If Using TSK with SIO

INITIALIZATION

1. The client task `tskLoop` is created in the function `main ()`.
2. This task creates SIO channels for data transfer - TSK_INPUT_CHANNEL and TSK_OUTPUT_CHANNEL.
3. It allocates and primes the buffers for to be used for data transfer.

EXECUTION

1. The task issues an empty buffer on TSK_INPUT_CHANNEL and waits to reclaim it. The reclaim is specified to wait forever.

2. It then issues the same buffer on TSK_OUTPUT_CHANNEL and waits to reclaim it. The reclaim is specified to wait forever.
3. The completion of reclaim operation indicates that the buffer has been transferred across the physical link.
4. These steps are repeated until the number of iterations passed as an argument to the DSP executable is completed.

FINALIZATION

1. The task frees the buffers allocated for data transfer.
2. It deletes the SIO channels TSK_INPUT_CHANNEL and TSK_OUTPUT_CHANNEL.

If Using SWI with GIO

INITIALIZATION

1. In the function `main ()`, GIO channels for data transfer - SWI_INPUT_CHANNEL and SWI_OUTPUT_CHANNEL are created.
2. A SWI object is created for doing the data transfer. One of the attributes for the SWI object is the callback function `loopbackSWI`. This function is called when the SWI is posted on completion of READ and WRITE requests on the GIO channels.
3. The buffers for to be used for data transfer are allocated and primed.

EXECUTION

1. To initiate the data transfer a READ request on the input buffer is submitted on the SWI_INPUT_CHANNEL.
2. Once the SWI is posted, contents of input buffer are copied to the output buffer.
3. The *empty* input buffer is reissued onto the input channel and the filled buffer is issued onto the output channel.
4. The SWI is posted again after the completion of both requests.
5. Steps 2 to 4 continue till the time GPP application is issuing buffers.

FINALIZATION

In the sample, the SWI is continuously posted due to READ and WRITE requests. So it would never reach the finalization. The finalization sequence, however, would be:

1. The buffers allocated for data transfer are freed.
2. The GIO channels SWI_INPUT_CHANNEL and SWI_OUTPUT_CHANNEL are deleted.

2.5.2 Building the GPP-Side of the Application

The procedure to build loop application is same as that of dsplink.

1. Change to the directory containing loop sample.

   ```
   $ cd /home/username/dsplink/gpp/src/samples/loop
   ```

2. To build the application:

   ```
   $ gmake –s debug
   ```

3. Upon successful completion of build, `loopgpp` shall be created in the directories:

   `~/dsplink/gpp/export/BIN/Linux/OMAP/DEBUG`

2.5.3 Building the DSP-Side of the Application

As described above, the DSP side application can either be executed with TSK or SWI. This configuration is controlled by the variable APPLICATION_MODE in file `main.c`.

Modify this value according to the configuration in which you want the DSP side to execute.

```
#define APPLICATION_MODE TSK_MODE
```

OR

```
#define APPLICATION_MODE SWI_MODE
```

As shipped in the release package the sample is configured to run in TSK_MODE.

For building the DSP side, change to the sample directory and build the project using `timake.exe`.

```
L:\> cd ti\boards\osk5912\dsplink\dsp\src\samples\loop\OMAP
L:\ti\boards\osk5912\dsplink\dsp\src\samples\loop\OMAP> timake loop.pjt
debug
```

NOTE: You must first run <$(CCS_INSTALL_DIR)>\dosrun.bat so that the timake command and the TI Code Generation Tools are found.

The DSP side binaries are created in the directories – `loop\OMAP\Debug`.

Loop sample built using the commands shown above contains both TEXT and BIOS sections in internal memory. To build the application with any of these sections in external memory, select another project file accordingly.

`Loop_bios.pjt`	`.bios` section in external memory
`Loop_text.pjt`	`.text` section in external memory
`Loop_bios_text.pjt`	Both `.bios` and `.text` sections in external memory

2.5.4 Executing the Loop GPP Application

Copying files to target file system

The generated binaries on the GPP side and DSP side and the data files must be copied to the target directory.

GPP Side

For executing the DEBUG build, follow the steps below to copy the relevant binaries:

```
$ cd ~/dsplink
$ cp gpp/export/BIN/Linux/OMAP/DEBUG/loopgpp ../montavista/filesys/opt/loop
$ cp gpp/export/BIN/Linux/OMAP/DEBUG/dsplinkk.o
../montavista/filesys/opt/loop
```

Enter the commands shown above in single line. Note that you will have to create the "loop" directory in ~/montavista/filesys/opt before executing the commands.

DSP Side

The DSP binaries are built on the Windows host. These binaries must be copied into the target file system. Samba or a FTP client can be used for this transfer.

For executing the DEBUG build:

1. Transfer following files into the directory `~/montavista/filesys/opt/loop`:

   ```
   C:\ti\boards\osk5912\dsplink\dsp\src\samples\loop\OMAP\Debug\loo
   p.out
   ```

Loading the kernel module: dsplinkk.o

To load the device driver and create the device node, login as 'root' and enter following commands on the command prompt:

```
$ cd /opt/loop
$ insmod -f dsplinkk.o
$ mknod /dev/dsplink c 230 0
```

This action may generate a warning indicating that the kernel module does not contain the GPL license. This warning can be safely ignored.

Running the application

To invoke the application enter the following commands:

```
$ cd /opt/loop
$ ./loopgpp loop.out <buffersize> <iterations>
```

e.g.

```
$ ./loopgpp loop.out 128 10000
```

2.6 RF6 Audio Application Overview

This section describes the process by which one can build and debug the audio demo application that is stored in the flash on the OSK5912 device (referred to in section 1.1 of the OSK Users Guide document).

To summarize, the RF6 audio application processes an incoming stereo audio signal on the DSP, then sends the data to the GPP, which has the option to process the data before sending it back to the DSP for further processing. Finally, it sends the signal to the output codec. A control thread on the GPP sets DSP algorithm parameters such as volume and high-/low-pass filtering.

Figure 2 shows the overall processing flow in the default RF6 application.

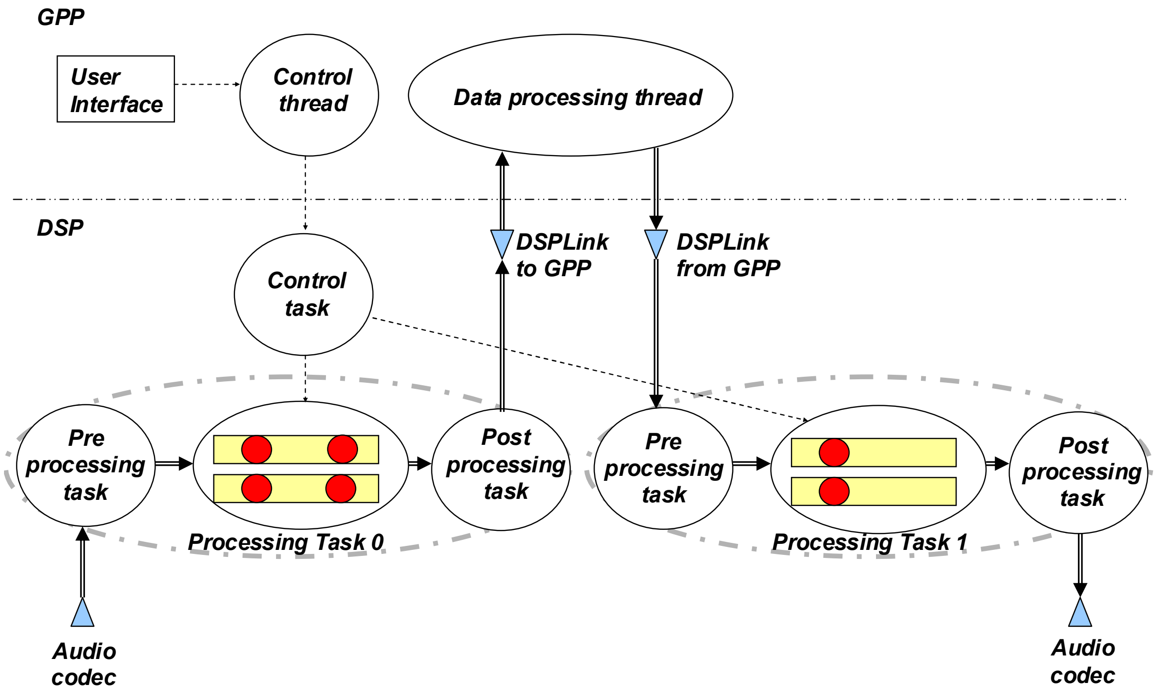


Figure 2. RF6 Application Processing Flow

The application takes the incoming stereo audio signal and converts it to digital data at a given sampling rate. One sampling of the signal gives a block of two signed 16-bit integers, one for the left and one for the right channel. The application groups blocks into frames of a given size before processing them.

For processing task 0, the application splits each incoming interleaved stereo frame into two single-channel frames. One frame contains only left-channel samples; the other contains only right-channel samples. The application processes these frames separately in processing task 0. To process one channel frame, the application applies a FIR filter and a volume control algorithm to it. Filter coefficients (low-pass, high-pass, or passthrough) are set for this task via the GPP-side application. Volume control messages are ignored by this task in the default application.

After it processes each channel, the application joins the independent channel frames back into one interleaved stereo frame. This frame is then sent to the GPP via DSP/BIOS Link. The default GPP-side application performs a simple memcpy() and sends the frame back to the DSP via DSP/BIOS Link. The memcpy() is a place-holder for adaptation (such as algorithm processing) on the GPP side.

Processing task 1 is similar to processing task 0, however it uses only the VOL algorithm since applying the FIR algorithm twice would degrade the signal. Volume control values for this task are set via the GPP-side application.

The pre-processing, processing, and post-processing tasks form a triplet that isolates the execution of a particular data path. Tasks are organized this way for easier adaptation.

As with all Reference Framework levels, the pre- and post-processing can be optimized in either of the following ways:

- Pre- and post-processing may be folded into the main processing task, thus reducing the total number of TSKs in the system
- Pre- and post-processing may be executed in the IOM codec device driver directly. For example, the C55x DMA controller could be programmed to sort left and right channel data directly.

2.7 Building the RF6 Audio Application

This section describes how to build both the DSP-side and GPP-side RF6 base applications.

2.7.1 Building the RF6 GPP-Side Application on a Linux Workstation

The following steps describe how to build the RF6 application.

1. In /home/username/rf6_gpp/apps/rf6/Linux/OMAP, build the GPP application with the following command:

```
[>] make
```

   The GPP application should build without errors. Both debug and release versions should be generated as a result.

   If the build fails, it may be because the dsplink library was not built, was built incorrectly (for example, the MSGQ component was not enabled), or the Rules.make DSPLINK_DIR variable was not set to point to the Link GPP code. Remember that RF6 does not ship a prebuilt dsplink library and kernel module on the GPP-side to ease potential versioning issues. Hence you first need to build dsplink.lib by following the instructions listed in the “Building the DSP/BIOS Link ARM side sources” section.

2. Copy the resulting executable, rf6_gpp, from /bin/rf6/Linux/OMAP/Debug to the NFS-mounted target file system. Type the following command at a single command prompt:

```
[>] cp ~/rf_gpp/bin/rf6/Linux/OMAP/debug/rf6_gpp
/home/username/montavista/filesys/opt/
```

2.7.2 Building the DSP-Side Application on a Windows Workstation

After you unzip the Windows package, you can build the DSP-side of the RF6 application.

1. Use the timake command to rebuild the DSP-side application from the Windows command prompt:

```
[>] cd c:\ti\boards\osk5912\referenceframeworks\apps\rf6\projects\target
[>] timake app.pjt DEBUG
```

NOTE: You must first run <$(CCS_INSTALL_DIR)>\dosrun.bat so that the timake command and the TI Code Generation Tools are found.

2. Copy the resulting DSP-side application (app.out) from apps\rf6\projects\target\debug to the target’s file system, in the same directory

where the GPP-side application is located (/home/username/montavista/filesys/opt). If you have samba installed, you can drag and drop this file this location in Windows.

2.8 Executing the RF6 Audio Application

This section assumes you have already set up the MontaVista Linux operating system on the target platform. On the NFS-mounted target file system, follow this procedure, which assumes you copied both the GPP-side and DSP-side RF6 applications to /home/username/montavista/filesys/opt:

1. Copy the following file to the /home/username/montavista/filesys/opt directory using Linux:
   – The dsplinkk.o Linux kernel module from your DSP/BIOS Link installation (choose either the debug or release version).

```
[>] cd /home/username/montavista/filesys/opt
[>] cp ~/dsplink/gpp/export/BIN/Linux/OMAP/DEBUG/dsplinkk.o .
```

2. Ensure that the appropriate file permissions have been set to load the various modules and execute RF6 by entering the following command:

```
[>]chmod –R u+rwx *
```

3. Reboot the target board and then start the Linux kernel via the bootloader.

4. At the prompt of the target board file system, enter the following commands:
   – Go to the directory containing the application:

```
[>] cd /opt
```

   – Load the DSP/BIOS Link kernel module, and create the dsplink and dsp device nodes.

```
[>] insmod –f dsplinkk.o
[>] mknod /dev/dsplink c 230 0
[>] mknod /dev/dsp c 14 3
```

   – Run the rf6_gpp user-mode process, which loads the DSP executable app.out with the AIC23 codec running at 44.1 kHz, and starts full data streaming through DSP/BIOS Link.

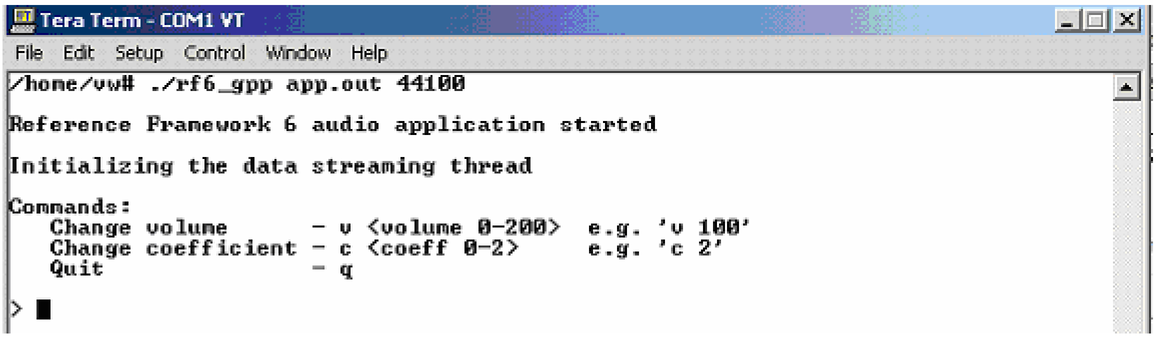
```
[>] ./rf6_gpp app.out 44100
```

   The last parameter is the frequency in Hz. Any frequency supported by the audio codec is allowed (for example, 8000, 16000, or 44100).

NOTE: If you are using the XDS510PP+ emulator, open SDConfig.exe and click the red “R” icon in the toolbar to reset the SD emulator.

(The emulator holds the DSP in reset when the latter is connected via JTAG.)

3. Start your CD player or other audio input device.
4. You should hear the FIR filtered audio output through the speakers connected to the target board.
5. You will see the application's GPP-side command menu:



```
/home/vw# ./rf6_gpp app.out 44100

Reference Framework 6 audio application started

Initializing the data streaming thread

Commands:
   Change volume      - v <volume 0-200>  e.g. 'v 100'
   Change coefficient - c <coeff 0-2>     e.g. 'c 2'
   Quit               - q

> █
```

Figure 3. RF6 GPP-Side Application Menu

Within this application, you can send either filter coefficients change commands or volume change commands. Table 1 shows the acceptable commands:

Table 1. GPP-Side Commands

Command	Use	Acceptable value range
c <value>	Change FIR filter coefficients	0 (low-pass filter) 1 (high-pass filter) 2 (all-pass filter)
v <value>	Change volume	0 to 200 (% of input volume)
q	Quit application gracefully	

When you have finished running the application, use the q command to quit. You can re-run the application as many times as you like using the following command:

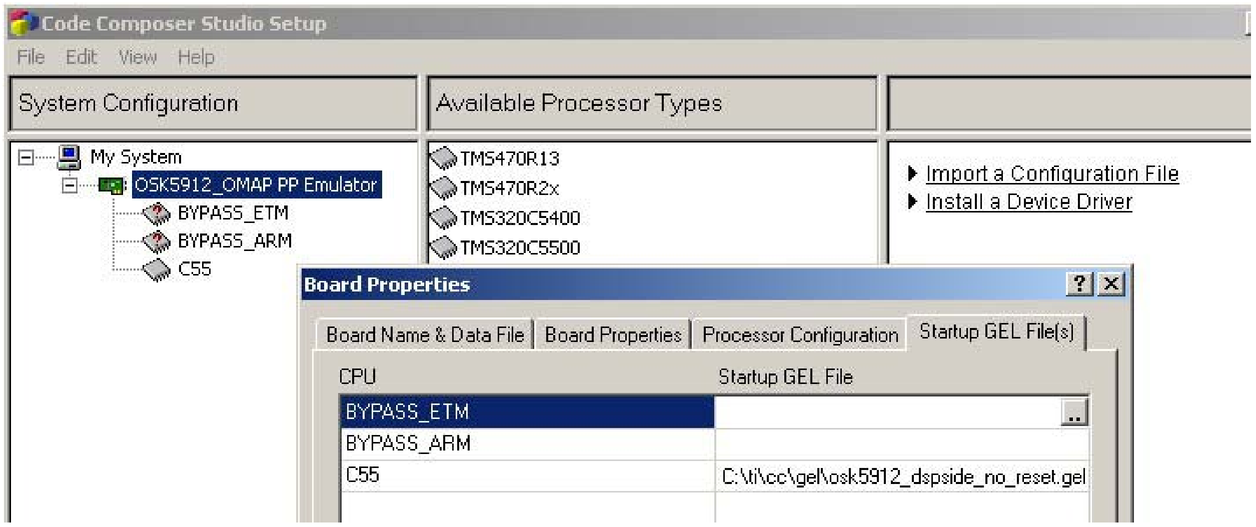
```
[>] ./rf6_gpp app.out 44100
```

2.9 Debugging the DSP-Side Application with Code Composer Studio

1. Obtain the **osk5912_dspside_no_reset.gel** GEL file for your target from the c:\ti\boards\osk5912\gel folder. Note that the line for the GEL_Reset() call is either commented out or removed from the GEL file. This change is necessary because when Code Composer Studio is launched, GEL_Reset() moves the Program Counter to the beginning of the code. This causes c-init record processing and other initialization to be repeated, which could cause problems (SDSsq33006).

```
/* GEL_Reset(); */
```

2. Use Code Composer Studio Setup, to configure your emulator as follows. (Figure 4 shows configuration of the OSK5912.)
   - Bypass 8-bit
   - Bypass 4-bit
   - C55 (TMS320C5500). Use the GEL file mentioned above.



CPU	Startup GEL File
BYPASS_ETM	
BYPASS_ARM	
C55	C:\ti\cc\gel\osk5912_dspside_no_reset.gel

Figure 4. Code Composer Studio Setup Settings

NOTE: The 4-bit bypass is used instead of TMS470R2 (ARM9), thus bypassing the ARM in the JTAG scan chain. This disables the ability to debug the ARM9 from Code Composer Studio. However, this is not a major issue since there are a number of feature-rich debuggers on the GPP-side, such as Data Display Debugger, a graphical frontend to the well-known gdb debugger.

3. Save the configuration and exit Code Composer Studio Setup.

4. With the application running and waiting for the next command, launch Code Composer Studio. You might encounter a few error messages that say "Trouble reading target CPU memory". Simply dismiss them by clicking Cancel. (SDSsq33006)

5. In Code Composer Studio, choose **Project→Open** and select the DSP-side project from the *RF_DIR*\apps\rf6\projects\target folder.

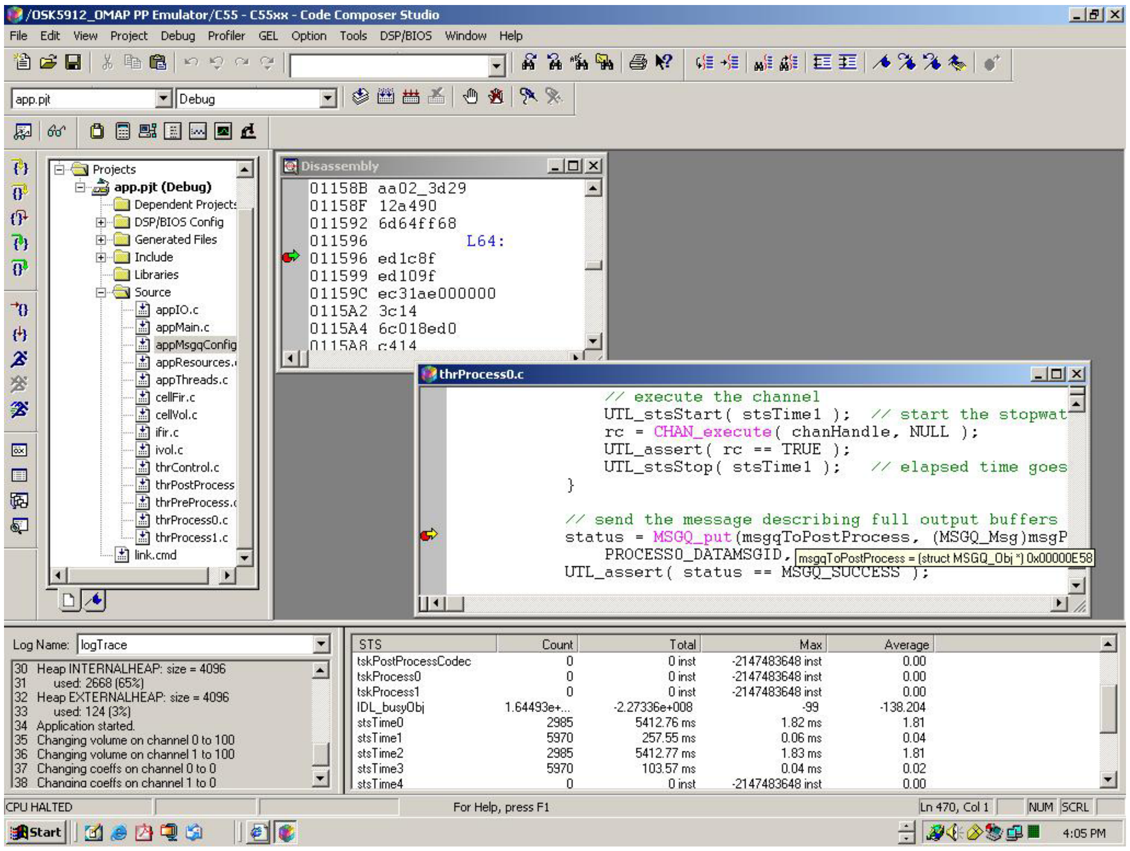
6. In Code Composer Studio, choose **File→Load Symbols→Load Symbols Only**. Note that you do NOT load the program here, since it has already been loaded by the GPP-side application via DSP/BIOS Link.

7. In the Load Symbols dialog, select the DSP executable that is being run (for example, app.out). This informs Code Composer Studio about the program loaded on the target, and allows Code Composer Studio to make use of symbolic debugging.

8. At this point, you may debug the DSP executable as you would other DSP programs. For example, you can set breakpoints, run, halt, watch variables, view LOG messages and STS statistics. Within Code Composer Studio, you can use the same DSP/BIOS Analysis Tools you use with non-GPP connected applications.

9. You can verify the data rate by following these steps:
   - Choose **DSP/BIOS→Statistics View** to open the statistics analysis tool.
   - Right-click on the Statistics View area and choose Property Page.
   - In the Units tab, set the unit for the stsTime0 object to milliseconds. (This object collects statistics for the thrProcess0 task.)
   - Notice that the average for stsTime0 average is 1.81 ms. This is consistent with the 44.1 kHz sample rate and the 80-sample-size buffers.

10. When you have finished debugging, leave the DSP running free (F5) and close Code Composer Studio. This is necessary so that the GPP-side application can shutdown cleanly when you use the quit (“q”) command.



```
01158B aa02_3d29
01158F 12a490
011592 6d64ff68
011596          L64:
011596 ed1c8f
011599 ed109f
01159C ec31ae000000
0115A2 3c14
0115A4 6c018ed0
0115A8 c414
```

```
            // execute the channel
            UTL_stsStart( stsTime1 );  // start the stopwat
            rc = CHAN_execute( chanHandle, NULL );
            UTL_assert( rc == TRUE );
            UTL_stsStop( stsTime1 );   // elapsed time goes
        }

        // send the message describing full output buffers
        status = MSGQ_put(msgqToPostProcess, (MSGQ_Msg)msgP
            PROCESS0_DATAMSGID, msgqToPostProcess = (struct MSGQ_Obj *) 0x00000E58
        UTL_assert( status == MSGQ_SUCCESS );
```

```
30 Heap INTERNALHEAP: size = 4096
31     used: 2668 (65%)
32 Heap EXTERNALHEAP: size = 4096
33     used: 124 (3%)
34 Application started.
35 Changing volume on channel 0 to 100
36 Changing volume on channel 1 to 100
37 Changing coeffs on channel 0 to 0
38 Changing coeffs on channel 1 to 0
```

STS	Count	Total	Max	Average
tskPostProcessCodec	0	0 inst	-2147483648 inst	0.00
tskProcess0	0	0 inst	-2147483648 inst	0.00
tskProcess1	0	0 inst	-2147483648 inst	0.00
IDL_busyObj	1.64493e+...	-2.27336e+008	-99	-138.204
stsTime0	2985	5412.76 ms	1.82 ms	1.81
stsTime1	5970	257.55 ms	0.06 ms	0.04
stsTime2	2985	5412.77 ms	1.83 ms	1.81
stsTime3	5970	103.57 ms	0.04 ms	0.02
stsTime4	0	0 inst	-2147483648 inst	0.00

CPU HALTED For Help, press F1 Ln 470, Col 1 NUM SCRL

Start 4:05 PM

Figure 5. Debugging RF6 in Code Composer Studio

For a detailed description of the architecture of the RF6 or DSP/BIOS Link, please refer to the DSP/BIOS Link and RF6 user guides that are located in the c:\ti\boards\osk5912\dsplink\doc and c:\ti\boards\osk5912\referenceframeworks directories respectively.