Contents

Chapter 1  Introduction

1.1  OMAP5912 Application Development
1.2  OMAP5912 c55xx DSP Application Development on Linux Hosts
1.3  Linux Hosted build, Printf debug using low impact LOG_printf

Chapter 2  Installing Linux DSP Tools

2.1  Dependency Checklist (supported platforms, required features)
2.2  Additional Requirements
2.3  Installation
   2.3.1  Linux DSP Tools
   2.3.2  Codegen – compiler, linker for 55xx DSP
   2.3.3  DSP/BIOS – real time OS for 55xx DSP
   2.3.4  DSP/BIOS Link – inter-processor communications
   2.3.5  RTA Tool Kit – Real Time Analysis software development kit
   2.3.6  Reference Framework – RF6
   2.3.7  Collateral Pack - xDAIS, PSL, IMGLIB, DSPLIB, RTDX from CCS 2.21
   2.3.8  Messaging (MSGQ support)

Chapter 3  Setting up for development

3.1  Build DSP/BIOS Link
3.2  Build and Run 'Hello World' from DSP
3.3  Build and Run RF6
3.4  Modify RF6 to add LOG_printf to syslog
3.5  Build and Run Loop example

Addenda

Support
Useful Tips
Chapter 1

Introduction

1.1 OMAP5912 Application development

The OMAP5912 System on a Chip (SoC) provides an ARM9 + 55xx DSP connected via shared memory with a boatload of peripherals. The OMAP5912 Starter Kit, OSK5912, provides an OMAP5912 evaluation board with Ethernet, USB and serial interfaces. The ARM development environment supplied with the OSK5912 is GNU embedded Linux using MontaVista PreView edition. The developer has two choices for DSP development: 1) Purchase Code Composer Studio + JTAG emulator, or 2) Use Linux DSP Tools for OSK5912. The first option, CCS, provides a complete integrated development environment (single step JTAG debugger, codegen, etc.). The second option, Linux DSP Tools, provides Linux hosted tools (compiler, Gnu make, Tconf) to build DSP applications and the foundational target content so that the developer can start from an existing base of robust working software on the DSP. This document addresses installation and setup to get started using the Linux DSP Tools option.
1.2 OMAP5912 c55xx DSP Application Development on Linux Hosts

The Linux DSP Tools distribution bundles together five standalone products: 1) c55xx codegen, 2) DSP/BIOS, 3) DSP/BIOS Link with MSGQ, 4) RF6, and 5) Real Time Analysis Software Developer’s kit (RTA SDK), and supplies a Getting Started Guide to step user’s through install, setup, and the start of application development.

Linux DSP Tools provides the following x86 PC Linux hosted tools for building DSP applications:

<table>
<thead>
<tr>
<th>Product</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C55xx Codegen</td>
<td>Compiler, Linker, Assembler, runtime library</td>
</tr>
<tr>
<td>DSP/BIOS Tconf</td>
<td>Configuration tool for DSP/BIOS real time kernel.</td>
</tr>
<tr>
<td>Gnu Make (gmake)</td>
<td>Make utility, comes with DSP/BIOS package</td>
</tr>
</tbody>
</table>

Linux DSP Tools provides the following foundational target content for DSP development.

<table>
<thead>
<tr>
<th>Product</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSP/BIOS</td>
<td>Real time kernel. Configurable, scalable, deterministic task scheduling with API’s for real time analysis.</td>
</tr>
<tr>
<td>DSP/BIOS Link</td>
<td>Program load, memory read write, shared memory channel driver for inter-processor communications between OMAP5912 ARM9 and c55xx DSP</td>
</tr>
<tr>
<td>DSP/BIOS RTA SDK</td>
<td>Real time analysis software developers kit. Provides transport and rendering of LOG_printf and STS data from DSP to ARM side applications.</td>
</tr>
<tr>
<td>Reference Frameworks (RF6)</td>
<td>RF6 is application starter-ware. Built on DSP/BIOS and Link provides, codec driver, algorithm socket, data streaming between audio codec, DSP and ARM.</td>
</tr>
<tr>
<td>Messaging (MSGQ)</td>
<td>The Windows PC buildable distribution of Messaging is supplied. This provides the libraries and headers to support using MSGQ, but no support for building with makefiles is provided in this release.</td>
</tr>
<tr>
<td>Target Content from CCS 2.21</td>
<td>Libraries and headers from CCS 2.21 for c5500 targets are supplied for Real Time Data Exchange (RTDX), Power Scaling Library (PSL), DSP Algorithm Interface Standard (xDAIS), Digital Signal Processing Function Library (DSPLIB) and Image Processing Library (IMGLIB).</td>
</tr>
</tbody>
</table>
1.3 Linux Hosted build, Printf debug using low impact LOG_printf

OMAP Applications consist of ARM and DSP executables built on a Linux Host. The ARM side development uses GNU tool chain provided by MontaVista. This guide is written assuming use of MontaVista 3.1 Preview Kit as provided with the OSK5912 package. There are dependencies between the content provided. In particular between DSP/BIOS Link and MontaVista Linux. User's are able to readily rebuild the MontaVista Embedded Linux kernel (MV Linux) with varying configuration options. The recommended practice is to rebuild DSP/BIOS Link after changing the configuration of MV Linux. Another dependency is with Reference Framework, RF6, which requires pthreads to be enabled in MontaVista, an option provided as default with the OSK5912. The other dependency to note is that DSP side executables must be built with the same options configured for ARM side DSP/BIOS Link, recommendation is to use PROC+CHAN+MSGQ.

To share files built on the on the Linux Host, an NFS mounted ‘share area’ is created. From the perspective of this guide, this are is known as the ‘run’ area since files are copied to this location to be run on the OSK5912 embedded Linux device.

Running an application consists of launching an ARM side executable from a minicom serial or ethernet terminal OSK session. The ARM side executable typically takes as a command line parameter the name of the DSP executable (coff file) and will load and run the DSP executable through DSP/BIOS Link API's. Examples of this type are provided with the RTA SDK, Reference Frameworks RF6 and DSP/BIOS Link example LoopGpp.

DSP executables are able to provide LOG_printf 4 word tokens to an ARM side utility, rtaTrace, that reads and renders them into formatted strings in the syslog. In the event the DSP executable crashes, a 'post mortem' capability is provided that reads the residual LOG_printf data (see RTA SDK docs for details or running kill -3 <rtaTrace process id>). LOG_printf is optimized for the hard real time environment of the DSP, typical impact is around 30 instruction cyrcles as compared to more than 30,000 for traditional printf.

The key to developing with LOG_printf is to make small incremental changes starting with a known good base of working software. Reference Framework RF6 is provided as starter-ware to provide this base.
1.4 Notes

This doc is written in a deterministic style, i.e., it documents the exact steps to integrate the various software products making a recommendation about the directory names and structure to enable future upgrades of individual components. Advanced users should be able to read the instructions and make their own decisions about where to place the software packages.
Chapter 2

Installing Linux DSP Tools

2.1 Dependency Checklist - supported platforms, required features

Required:
1) OSK5912 from Spectrum Digital – Beta kit or newer.
2) x86 PC Linux Development host (Red Hat v7.3 or Red Hat v9) with root access.

Checklist (with notes where appropriate) to complete before continuing:

- X86 Linux Development host – RH7.3 or RH9 required for MontaVista Linux install.
  - Root (or pseudo) access
  - System services
    - TELNETD
    - FTPD
    - NFSD (verify by running /sbin/service nfs status)
- GNU Developer tools (GCC, etc.)
  - The default set is sufficient for development
  - Recommend installing ‘Expect’ as this will assist in installation of Linux DSP Tools. An ‘Expect’ script is provided as an alternate to the detailed instructions to automate the installation.
- Korn Shell is required for BIOS config (Tconf) – not installed by default on Red Hat Linux (ACTION REQUIRED).
  - To locate korn shell for Linux, two options: a) go to www.rpmfind.net, b) Disk2 of Red Hat9
  - On disk2 of the Red Hat install disks, cd to /RedHat/RPMS disk 2
  - As root, run the command: rpm -Uvh pdksh-5.2.14-16.i386.rpm
- HTML browser and PDF viewer to read documentation.
- Minicom – serial communications program. Either run as root, or make sure user read /write for the ttys0 device (from com1). Minicom comes in the default Workstation Install of Red Hat 9.
- TFTP – trivial file transfer protocol. This is recommended for transferring Linux kernel image to uBoot for flashing the Linux kernel.
OSK5912 from Spectrum Digital
- OMAP5912 OSK development board with power supply, serial cable, Ethernet cable
- CDROM: MontaVista Linux Professional Edition 3.1 – Preview Kit for OMAP5912 OSK Professional
- CDROM: Spectrum Digital CD (schematics, app notes from TI, etc.)
- OMAP5912 Starter Kit (OSK5912) User’s Guide

Connection between OSK5912 and Linux Host
- Linux Host PC COM1 (or 2) connected to serial port on OSK5912
- Ethernet connection between OSK5912 and Linux Host, recommend using a router.

Other connections
- Recommended: Mini-jack stereo audio cable to supply audio input to the OSK5912
- Recommended: Headphones or speakers

Read and follow the instructions for Sections 1 and 2 of the OMAP5912 Starter Kit User’s Guide (printed copy comes in the OSK5912 box). This will insure the OSK5912 board is working and ARM side MontaVista embedded Linux is installed, setup, built and ready for development.

Section 1 – Quick Start
- 1.1 Booting the OSK5912
  - Strongly recommended step on Linux PC is to run 'minicom –s' to setup. Serial port settings 115200, 8, N, 1, No hardware or software flow control. Also, change Modem and dialing parameters: delete ‘Init String’, ‘Reset String’ and ‘Hangup’ String. Save to default and exit. (in Terminal window: ‘man minicom’ or select Help->Index->minicom for more info).

- 1.2 Running the Audio Demo
  - This section walks you through running the RF6 application burnt into the flash file system of the OMAP5912 OSK board – a good sanity check and prep for later running RF6 supplied with the Linux DSP Tools installation.

Section 2 – Developing with Linux.
- 2.1 Overview
- 2.2 Installing MontaVista Preview Kit
- 2.3 Installing a local copy of the Preview Kit
- 2.4 Building the Linux Kernel
- 2.5 Booting the OSK with a target file system
  - Step 4 indicates to start Hyperterminal and use Kermit to transfer the ulmage file from a Windows PC. For the Linux PC, ‘minicom’ doesn’t work due to an incompatibility with uBoot and kermit. The alternative for Linux PC is to use TFTP. The Spectrum Digital website gives instructions for this at http://omap.spectrumdigital.com/osk5912/factoryconfig.html
- 2.6 Building and Running a “Hello World” application.
2.2 Additional Requirements

Refer to release notes (doc/release_notes_linuxdsptools.html) for additional requirements.
2.3 Installation

**Overview:** Install Linux DSP Tools (LDT) by copying the installation package into a temporary location. This installation requires acceptance of a Eula, then provides the sub-package installs for Codegen, DSP/BIOS, DSP/BIOS Link, RF6, and RTA SDK. Run the install and direct the resulting sub-packages to another temporary location as each of these software packages will then be installed or untar'd into the desired location. The instructions below will guide you step by step presuming root access, though the packages can be installed into a 'user' area just the same (see table below for recommendations). Once installed, Chapter 3 will provide instructions on making development areas and setting up 'starter ware' to begin DSP app development.

Suggested install locations:

<table>
<thead>
<tr>
<th>Action</th>
<th>Logged in as ‘root’</th>
<th>Logged in as ‘user’</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temporary location to copy LDT install package</td>
<td>/tmp</td>
<td>/home/&lt;user&gt;/tmp</td>
</tr>
<tr>
<td>Destination directory when running LDT install package</td>
<td>/tmp/ldt</td>
<td>/home/&lt;user&gt;/tmp/ldt</td>
</tr>
<tr>
<td>Destination directory when installing sub-package installs</td>
<td>/opt/ti_dsptools</td>
<td>/home/&lt;user&gt;/ti_dsptools</td>
</tr>
</tbody>
</table>

Notes on instructions:

Note1: Logged in as root in a text based terminal.

Note 2: /opt chosen as it’s the standard location for ‘Add-on software packages’

Note 3: /tmp chosen for base of install as these will be deleted once installed /opt
2.3.1 Install Linux DSP Tools

These instructions assume logged in as root in a terminal session. The Linux DSP Tools can be installed as user, substitute the destination directory in a user account area. The times listed for each section are approximate.

- 1) Copy the installation package into a temporary location. (Note: XX is used to de-couple this document from the actual version proved, ex. linuxdsptools_v1_00_00_02.bin).
  
  o \textit{cp} linuxdsptools\_v1\_00\_00\_XX.bin /tmp

- 2) Change permissions (chmod), if not already executable, on the installation file.
  
  o cd /tmp
  o chmod +x linuxdsptools\_v1\_00\_00\_XX.bin

- 3) Run the install from \texttt{/tmp} and supply a destination directory, recommend \texttt{/tmp/ldt}. Note: omit the option \texttt{-console} for graphical install.
  
  o ./linuxdsptools\_v1\_00\_00\_XX.bin --console

- 4) During the install you will be prompted to accept the EULA, after having done this, you will be prompted to supply a destination directory, for the purpose of these step by step instructions, \texttt{/tmp/ldt} is recommended.
  
  o <Enter destination> /tmp/ldt

- 5) After the installation finishes, change directory to the destination and check the listing for directories containing sub-package installs for bios, cgtools, collateral\_pack, doc, dsplink, msgq, rf, and rdatools.
  
  o cd /tmp/ldt
  o ls

- 6) Create a destination directory where the sub-packages will be installed.
  
  o mkdir /opt/ti_dsptools

- 7) **Important Note:** there is an ‘Expect’ script included in the root of the install (\texttt{/tmp/ldt}) that automates steps 2 thru 7. This requires ‘Expect’ to be installed (or wish to install it) then run this script as this is automation for Steps 2 thru 7. Otherwise go to Step 2. The instructions provided assume root privileges.
  
  o ./subpkg\_install\_linuxdsptools.exp /opt/ti_dsptools

- **Note:** If ‘Expect’ script was used, the install is complete. **go to Chapter 3**
2.3.2 Install 55xx Codegen tools

- 1) Create the destination directory for codegen
   - mkdir /opt/ti_dsptools/cgtools-c5500-v2.56

- 2) Unzip and untar the 55xx codegen tools into the development directory.
   - cd /tmp/ldt/cgtools
   - tar zxf c55256-linux.tar.gz -C /opt/ti_dsptools/cgtools-c5500-v2.56

- 3) The C55xx DSP code generation tools are now copied into the directory from which they will be used. There will be a top level README.txt along with bin, include and lib directories. To verify, list the directory.
   - ls /opt/ti_dsptools/cgtools-c5500-v2.56

- Step 2 complete

2.3.3 Install DSP/BIOS

- 1) Run the installshield executable ‘setuplinux.bin’ as a console install (to run GUI install, drop the –console option).
   - cd /tmp/ldt/bios
   - ./setuplinux –console

- 2) The installer will ask you to accept the end user license agreement (EULA) and proceed through until the ‘Install Location’ is requested.

- 3) Specify the destination directory name for ‘Install Location’:
   - /opt/ti_dsptools/bios_5_03

- 4) Complete the install of DSP/BIOS 5.03. For more info on bios, at /opt/ti_dsptools/bios_5_03, there is the DSP/BIOS GettingStartedGuide.html, an index to DSP/BIOS manuals, and release_notes.html at the BIOS base directory.
   - ls /opt/ti_dsptools/bios_5_03
2.3.4 Install DSP/BIOS Link

Install DSP/BIOS Link.
- 1) Unzip the DSP/BIOS Link distribution (this installs both ARM and DSP side).
   - cd /tmp/ldt/dsplink
   - tar -zxvf dsplink_5912osk_montavista_v112.01.tar.gz -C /opt/ti_dsptools
- 2) Verify install by looking for the directory named dsplink
   - ls /opt/ti_dsptools
- 3) Rename to the install destination directory to give it a version.
   - mv /opt/ti_dsptools/dsplink /opt/ti_dsptools/dsplink_1_12_01
- For more info on DSP/BIOS link, refer to the User's Guide, Reference and Release Notes in the doc subdirectory.
   - ls /opt/ti_dsptools/dsplink_1_12_01/doc
- Step 4 complete. For more info on DSP/BIOS link

2.3.5 Install RTA Tool Kit

Install RTA Tool Kit.
- 1) Change directories to the RTA SDK tools directory. Run the installshield executable ex. 'dsp_bios_rta_setuplinux_5_00_00_XX.bin' as a console install.
   - cd /tmp/ldt/rtatools
   - ./dsp_bios_rta_setuplinux_5_00_00_XX.bin –console
- 2) The install will ask you to accept the end user license agreement (EULA) and proceed through until the ‘Install Location’ is requested.
- During the install, you’ll be asked to specify the directory name, recommend:
  - /opt/ti_dsptools/rtatools/rtatools_5_00
- 3) Verify the install by listing the contents of the destination directory. The base should contain a 'ti' sub-directory containing the content with the User's Guide at the top level, spru819DspBiosRtaSdkUsersGuide.pdf and release_notes.html. In addition, there is a file that lists the versions of the Real Time Analysis software developer's kit subcomponents of the form DSP_BIOS_RTA_SDK_5.00.00.xx_Components
  - ls /opt/ti_dsptools/rta_sdk_5_00

2.3.6 Install Reference Framework RF6 [Time: 5min]

- 1) RF6 consists of three pieces, an application note (spra796a.pdf), an ARM side package and a DSP side package.
- 2) For future reference copy the doc directory to the development area. Note spra796a.pdf is the Reference Framework 6 Application Note.
  - mkdir /opt/ti_dsptools/doc
  - cp /tmp/ldt/doc /opt/ti_dsptools/doc

- 3) Install the ARM side of RF6. This will result in a directory named rf_gpp under the /opt/ti_dsptools directory. To be consistent with the other packages and pave the way for update management, this will be renamed with a version identifier.
  - cd /tmp/ldt/rf
  - tar xzf rf_gpp_v3_11_00_00.tar.gz -C /opt/ti_dsptools
  - mv /opt/ti_dsptools/rf_gpp /opt/ti_dsptools/rf_3_11_gpp

- 4) Verify the install by listing the directory /opt/ti_dsptools/rf_3_11_gpp. The listing will show directories named apps, bin, include, lib, src and Rules.mak
  - ls /opt/ti_dsptools/rf_3_11_gpp

- 5) Install the DSP side of RF6.
  - unzip rf_v3_11_00_00.zip -d /opt/ti_dsptools

- 6) The DSP side of RF6 will then be located in /opt/ti_dsptools/referenceframeworks.
- 7) Rename the top level DSP side directory of RF6.
  - mv /opt/ti_dsptools/referenceframeworks /opt/ti_dsptools/rf_3_11_dsp
8) Verify the contents by listing the directory /opt/ti_dsptools/rf_3_11_dsp. The listing will show directories named apps, include, lib, src with a collection of Windows PC batch files and a top level gnu makefile.

   o ls /opt/ti_dsptools/rf_3_11_dsp

- Step 6 complete.

2.3.7 Install Collateral Pack

- 1) Install target content from Code Composer Studio, CCS 2.21. This provides xDAIS, rtdx, power scaling library, dsp lib, and image lib. Copy the bash script which demonstrates setting environment variables. Note: docs were previously copied over when installing Reference Frameworks.

   o cd /tmp/ldt/collateral_pack
   o unzip c5500.zip –d /opt/ti_dsptools/ccs221_content
   o cp linuxdsptools_env.bash /opt/ti_dsptools

- Step 7 complete

2.3.8 Install MSGQ

- 1) Install Messaging support for DSP/BIOS and DSP/BIOS Link. This provides source, libraries and headers from the PC distribution are provided to support message passing. The examples provided with MSGQ are not supported with makefiles for Linux build, we'll be just using the libraries and headers in this release for the support of DSP/BIOS Link. DSP/BIOS Link requires MSGQ if certain configuration options are selected, specifically PROC + CHNL + MSGQ (more on this in Chapter 3).

   o cd /tmp/ldt/msgq
   o unzip msgq_v1_01_00_01.zip –d /opt/ti_dsptools/
   o mv /opt/ti_dsptools/messaging /opt/ti_dsptools/msgq_1_01

- Step 8 complete
Chapter 3
Setting up Linux DSP Tools for development

Overview: This section assumes that you have followed the instructions in the OSK5912 User’s Guide (specifically sections 2.4, 2.5 and 2.6) and have installed and built MontaVista Linux on the host Linux PC and then programmed the kernel into the flash on the OSK5912. Here’s an overview of the steps detailed in the next section to setup for DSP development:

**Build DSP/BIOS Link:** First step is to build the ARM (gpp) and DSP side of DSP/BIOS Link, the conduit between the ARM and the DSP, so that it is in sync with the version and configuration of MontaVista Linux programmed into the flash on the OSK5912 and is configured correctly with respect to data executables built for the DSP. For more info on DSP/BIOS Link, see the doc directory off the top of the install, ex. /opt/ti_dsptools/dsplink_1_12_01/doc.

**Run ‘Hello World’ on DSP:** Next step is to load and run a pre-built ‘hello world’ DSP app provided with the RTA SDK. This will give insight into the ‘printf style’ debug support provided by the RTA SDK and workflow steps necessary to move development software from Linux host to embedded Linux platform, start and run DSP/BIOS Link, then load and run a DSP executable. For more info on the RTA SDK, see the doc index at the top of the install, ex. /opt/ti_dsptools/rtasdk_5_00/index_bios_rta_mauals.html.

**Modify, build and run RF6.** The OSK5912 User’s Guide walks through running RF6, this section describes how to modify, build and run RF6. Specifically, modify the DSP-side of RF6 to send LOG_printf data to ARM by changing a configuration parameter in a DSP/BIOS Tconf script and leveraging the RTA SDK library. This results in the LOG_printf data from RF6 being available from the ARM. As a note, RF6 is recommended as starter-ware for developing multi-tasking DSP applications and comes with drivers and infrastructure for streaming stereo CD quality audio data sampled from the audio codec (LINE IN) through DSP/BIOS Link to the ARM and back to DSP and out through audio codec (HEADPHONE). Adding the LOG_printf debugging capability paves the way for app development. For more info on RF6 see the appnote in the install area: /opt/ti_dsptools/doc/spra796a.pdf.

**Build and run Loop GPP example.** Loop GPP is a simpler alternative to RF6 as starter-ware. This example streams data from the ARM to the DSP which then copies it back the ARM.
3.1 Build DSP/BIOS Link

DSP/BIOS Link provides a multi-channel inter-processor communications driver for communicating between ARM and DSP. DSP/BIOS Link is provided as a source code distribution, in the spirit same spirit as open source Linux. This section describes making a development copy of DSP/BIOS Link, setting configuration parameters and building first the ARM side libraries and modules for compatible configuration with Montavista Linux (ARM side), then the DSP side libraries with the same configuration options.

1) First, let’s create a set of environment variables that will be used throughout the remaining steps when configuring makefiles and also provides some useful directory shortcuts. The file `linuxdsptools_env.bash` is provided for this purpose. This file has been copied to `/opt/ti_dsptools` from actions in the previous section. Open this file and make note of the contents, some user’s may want to replace `$HOME` with the absolute path. Add this script to the ‘.rc’ file associated with your ‘user’ shell, (ex. `.bashrc`) as this will provide the environment variables and PATH necessary for developing with the various components, ie, add the line ‘source `/opt/ti_dsptools/linuxdsptools_env.bash’` to the ‘.rc’ file. It’s important to note that these will apply to the shell used for development, so it’s necessary to switch to the ‘user’ account that will be used for development. Observe that some variables are based in `$HOME` while others are in `/opt/ti_dsptools` (Note: `$HOME` applies to the ‘user’ account not to root). The reasoning is components such as DSP/BIOS and 55xx Codegen are used, while other components such as DSP/BIOS Link and RF6 are built (copied to `$HOME`). The following sections in chapter 3 will guide your through copying each of the components to your user area. The contents of the bash script are shown below and the remainder of this document will presume these variables are set. Root privileges are assumed to begin, then a change to user to edit .rc file.

- o cd `/opt/ti_dsptools`
- o `<edit> linuxdsptools_env.bash` insure all environment variables are correct. Final step is to source the `linuxdsptools_env.bash` script so the variables are available in this session. To insure correctness, cd to each variable after source the script, for example type ‘cd $DSPLINK’ to verify DSPLINK.
  - export DSPLINK=$HOME/dsplink_1_12_01
  - export MSGQ=/opt/ti_dsptools/msgq_1_01
  - export BIOS=/opt/ti_dsptools/bios_5_03
  - export CGTOOLS=/opt/ti_dsptools/cgtools-c5500-v2.56
  - export RF_GPP=$HOME/rf_3_11_gpp
  - export RF_DSP=$HOME/rf_3_11_dsp
  - export RTASDK=/opt/ti_dsptools/rta_sdk_5_00
- export OSK_SHARE=$HOME/montavista/filesys
- export OSK_WORK=$OSK_SHARE/opt
- export PATH=$PATH:$DSPLINK/etc/host/scripts/Linux:
  /opt/montavista/previewkit/arm/v4t_le/bin:/opt/montavista/previewkit/host/bin

  - <save and exit>
  - su <user>
  - cd /home/<user>
  - <edit> .bashrc (assuming bash as shell) add the following line:
    - source /opt/ti_dsptools/linuxdsptools_env.bash
  - <save and exit>
  - source /opt/ti_dsptools/linuxdsptools_env.bash

- 2) Logged in as 'user', make a development copy of the dsplink_1_12_01 directory by copying from the install area to the user development area. Next, go to the directory that contains the DSP/BIOS Link config scripts.

  - cp -r /opt/ti_dsptools/dsplink_1_12_01 $HOME
  - cd $DSPLINK/etc/host/scripts/Linux

- 3) Change the permissions on the file ‘dsplinkcfg’ to execute and run it. This launches a perl script that configures DSP/BIOS Link.

  - chmod a+x dsplinkcfg
  - ./dsplinkcfg

- 4) The first menu confirms the environment variable DSPLINK is set correctly, if this is not correct return to the step where the gppenv.bash file is edited and make sure the DSPLINK variable is pointing to the location of development copy of DSP/BIOS Link. Note: if DSP/BIOS Link configurations are changed, use ‘gmake –s clean clobber’ before making the config change in $DSPLINK/gpp/src and $DSPLINK/dsp/src directories.

  - If DSPLINK is correct, select ‘1’ to continue
  - Next, select ‘1’ the OMAP5912 OSK GPP platform.
  - Select ‘1’, OMAP as the DSP platform
  - Choose option ‘3’ to select MontaVista Linux Pro 3.1 Preview Kit.
  - Choose option ‘2’ to select c55xx BIOS v5.00 for Linux
  - Choose option ‘1’ to select PROC+CHNL+MSGQ build configuration. DSP/BIOS Link is scalable and this is one of the configuration options that affects size of DSP/BIOS Link. As a side effect, this also affects the layout of data structures that need to be the same between
ARM and DSP. The DSP examples that are pre-built have been built with the PROC + CHNL + MSGQ option which is the reason for selection option 1, as you gain experience configure (optimize) link for the particular needs of your application.

- **Select ‘0’** – trace not enabled. Recommend reading DSP/BIOS Link User's Guide in $DSPLINK/doc to explore this and other features.
- **Select ‘0’** – profiling not enabled.
- Finally, **select ‘0’** – probe not enabled
- Config is complete – configuration generated into $DSPLINK/config/CURRENTCFG.MK

5) Next, edit the $(DSPLINK)/make/Linux/osk5912_mvlpreviewkit3.1.mk file located in $DSPLINK/make/Linux to specify the correct base locations of Montavista components, this file and settings are for Montavista Preview kit, refer to DSP/BIOS Link User's Guide for details for Pro version of Montavista. Note: development builds should be performed logged in as user, not root.

- **cd** $DSPLINK/make/Linux -
- **<edit> osk5912_mvlpreviewkit3.1.mk** - set the following variables:
  - BASE_BUILDSOS := $(HOME)/montavista/kernel/linux
  - BASE_CGTOOLS := /opt/montavista/previewkit/arm/v4t_le/bin
  - BASE_OSLIB := /opt/montavista/previewkit/arm/v4t_le/lib
  - OSLIB_PLATFORM := $(HOME)/montavista/filesys/lib
  - STD_LD_FLAGS := -lc $(HOME)/montavista/filesys/lib/libpthread.so.0

6) Change directories to the gpp/src directory. Clean and Build the ARM side of DSP/BIOS Link. (This assumes the DSPLINK and PATH environment variable are set as described above). Note the '-s' silent option can be passed to gmake and depending on the version of gmake the output will be easier to read, leave off the –s for verbose output. Note: gmake ver 3.80 is provided by DSP/BIOS in /opt/ti_dsptools/bios_5_03/tconf/bin.

- **cd** $DSPLINK/gpp/src
- **gmake** –s clean
- **gmake** –s

7) The release versions of the kernel module (dsplinkk.o) and user library (dsplink.lib) will be created in $DSPLINK /gpp/export/BIN/Linux/OMAP/RELEASE upon successful completion of the build.

- **ls** $DSPLINK/gpp/export/BIN/Linux/OMAP/RELEASE

8) It's now time to build the DSP side of DSP/BIOS Link now that the ARM side kernel module and library of have been built. As noted above, DSP/BIOS Link has configuration options, which must be the same between ARM and DSP side – notably the selection of PROC + CHNL + MSGQ as this affects a
shared data structure among other things. Since the edit of the c55xx_5.00_linux.mk makefile located in $DSPLINK/make/DspBios is prone to typo's, here's a helpful Makefile debug tips: – add the statement: $(warning $CGTOOLS)) to the makefile to help debug if CGTOOLS (for example) is properly resolved. Also, insure that file permissions are set correctly, build errors can occur that are tough to pinpoint if files are inadvertently owned by root (by editing or building with root permissions) in the user area.

- Change directories to $DSPLINK/dsp/src, clean and build. A successful build will produce the dsplink.lib library in the directory $DSPLINK/dsp/export/BIN/DspBios/OMAP/RELEASE. Note: there are some remarks generated during the build that can be ignored, see section 12 “Build DSP side sources” in the DSP/BIOS Link User's Guide in $DSPLINK/doc.

- Step 3.1 is complete – ARM and DSP sides of DSP/BIOS Link has been built.
3.2 Build and Run ‘Hello World’ from DSP

[Time: 30 min]

The RTA SDK provides a pre-built hello world application that runs on the DSP and demonstrates LOG_printf from DSP to ARM Linux application. The sequence will be to:

1) load the DSP/BIOS Link drivers on the embedded Linux running on OSK,
2) use the ‘loadandrun’ utility that utilizes DSP/BIOS Link to load and run the ‘hello’ DSP executable,
3) then start the ARM side Linux application ‘rtatrace’ to read and the LOG_printf buffers from the DSP.

Make note of the workflow – development is done on the Linux PC ‘Host’ for both ARM and DSP side applications. Executables (and data) are copied to an area shared between host and embedded target (NFS mounted filesystem), then executed on embedded OSK5912 ARM Linux target. If development involves a program for the DSP, a utility is executed on the embedded Linux target to load and run the DSP executable via DSP/BIOS Link. These steps presume the NFS mounted ‘share’ area has been setup per the instructions in the OSK5912 User’s Guide – section 2.5, step 12.

- 1) Create a ‘user’ copy of the RTA SDK for development (logged in as ‘user’), then create an execution area for the rta example in the shared file system between the Linux host and the embedded Linux OSK board – we’ll create ‘/work’ on the OSK5912, equivalent to $HOME/montavista/filesys/work on Linux host. Hereafter in this guide the shared execution area on the embedded Linux device will be referenced by $OSK_WORK – make sure the /opt/ti_dsptools/linuxdsptools_env.bash is properly setup if something other than /work is chosen for the execution directory. Note: The instructions in section 2.5 of the OSK5912 Starter guide, specifically step 12, created a development host directory $HOME/montavista/filesys that is mounted as the root of the embedded linux device. In a user terminal window on the host linux machine, make a copy of the rta tools:
   - cp –r /opt/ti_dsptools/rta_sdk_5_00  $HOME
   - cd $OSK_SHARE
   - su root  (login as root)
   - mkdir work
   - chown –R <user>:<user> work  (ex. user is tm, chown –R tm:tm work)
   - exit    (logout as root, return to <user>)
   - cd $OSK_WORK
   - mkdir rta
   - cd rta
2) Copy the link drivers from the install location to the Linux host development area (shared with embedded Linux).
   (Note: having root permission on Linux host works around file permission problems in the shared area – refer to OSK User’s Guide for details on file permission setup.
   - cp $DSPLINK/gpp/export/BIN/Linux/OMAP/RELEASE/dsplinkk.o
   - cp $RTASDK/ti/bios/rta/sdk/lib/pmr_drv.o

3) From a terminal window on the Linux development host, hereafter referred to as user ‘Host Session’
   list the directory contents of the directory $HOME/monatavista/filesys/home/work/rta (also known as $OSK_WORK/rta).  This listing should show dsplinkk.o and pmr_drv.o. Copy the file ‘load4rta’ as this will be used in a following step.
   - ls $OSK_WORK/rta
   - cp /opt/ti_dsptools/rta_sdk_5_00/load4rta $OSK_WORK/rta

4) If you don’t already have a serial OSK session, open a serial port terminal session with the OSK board
   (open a terminal window and type minicom – settings are 115200,n,1 no flow control).  A second root session is opened to insure proper nfs mount during OSK boot.  The minicom terminal window connected via the serial port to the embedded OSK5912 Linux device will hereafter be referred to as ‘OSK Serial Session’. List the directory contents of /work/rta (to re-inforce the concept that this is the $OSK_WORK directory).
   - <open two new terminal session> – both logged in as root, in the first session, start minicom
     (refer to the OSK5912 User’s Guide for detail on starting a serial OSK session).
     - minicom
       - CTRL-A Z for help
       - 115200,n,1, no flow control
   - <in the second root session, stop and start the nfs service>
     - /sbin/service nfs stop
     - /sbin/service nfs start
   - hit the reset button on the OSK5912, this should cause the minicom session to display
     OSK5912 boot sequence.
   - Login to OSK5912 as root
   - cd /work/rta
   - ls

5) In the OSK Serial Session, load the link drivers by executing the following commands.  The easiest way is to run the executable file ‘load4rta’ that was copied in from a previous step.  Alternative is to run the commands one after another.  Note: insmod –f can be used to override kernel version checks, etc.)
   - ./load4rta
alternatively, commands for loading link and rta drivers can be run individually:

- mknod /dev/dsp c 14 3
- insmod dsplinkk.o
- mknod /dev/dsplink c 230 0
- insmod pmr_drv.o
- mknod /dev/bios_pmr c 253 0

- 6) Verify the modules are loaded by running the command on the OSK Serial Session. Results should show both dsplinkk and pmr_drv with 0 for ‘Used’.

  - lsmod

- 7) In the Host Session (user, not root) make sure your directory is the rta developer area, copy the pre-built hello world example to the development area $OSK_WORK/rta from /opt/ti_dsptools/rta_sdk_5_00. Then copy the loadandrun and rtaTrace utilities from their respective locations. Lastly, list the directory contents, verifying hello.out, channel driver library, loadandrun, and rtaTrace applications.

  - cd $OSK_WORK/rta
  - cp $RTASDK/ti/bios/rta/examples/dsp/basic/hello/osk5912/hello.out .
  - cp $RTASDK/ti/bios/rta/sdk/lib/libcd_bl.so .
  - cp $RTASDK/ti/bios/rta/examples/gpp/loadandrun/loadandrun .
  - cp $RTASDK/ti/bios/rta/examples/gpp/rtatrace/rtaTrace .
  - ls

- 8) In the OSK Serial Session, create an environment variable called LD_LIBRARY_PATH and point it to the location of the libcd_bl.so library, this will ensure that the rta utilities (ex. rtaTrace) can find the channel driver properly. Type ‘env’ to verify. Note: this is something that needs to be done once per session so we’ll create a ‘.profile’ then rather than reboot the OSK5912, we’ll set the LD_LIBRARY_PATH for this session. Note: it is important to leave the filesystem in the read-only state (remount,ro).

  - mount -o remount,ro /
  - <edit> /root/.profile (ex. vi /root/.profile) add the following line:

    - export LD_LIBRARY_PATH=/work/rta
  - mount -o remount,ro /
  - export LD_LIBRARY_PATH=/work/rta
  - env
9) In the **OSK Serial Session**, make sure you are in the directory `/work/rta`. Load and run the executable (using & to load in the background, hit ‘enter’ to get the prompt back). You should see several strings “calling….”. To see the application is running, type ‘jobs’ and you’ll see loadandrun listed with it’s job number. (Linux tidbit: typing ‘kill -2 <job number>’ will terminate, something useful to do if there is an error running rtaTrace which normally terminates loadandrun).

   - cd /work/rta
   - ./loadandrun hello.out &
   - <hit enter>
   - jobs

10) Start the ARM side Linux utility, rtaTrace, with options xosur (x says don’t write to sys log, o says write to stdout, s says statistics, u says cpu load, r says reset statistics). rtaTrace channels DSP-side LOG_printf data through DSP/BIOS Link and formats the output to sys log (or stdout) using the coff file for formatting. Consult the DSP/BIOS User’s Guide accessed thru html doc index located in /opt/ti_dsptools/bios_5_00/index_bios_manuals.html. The hello.out coff file needs to be supplied for the ARM side rendering by the rtaTrace utility of the 4 word LOG_printf message from the DSP. Note: the CPU busy number is unusually high (over 50%), due to a bug in the pre-built example configuration file – this will be fixed in a later step by rebuilding the dsp executable, hello.out, with the bios 5.03 provided.

   - ./rtaTrace -xosur hello.out
   - hit CTRL-C when you’ve seen enough output

11) For more info on running the rtaTrace command, line type ‘rtaTrace –help’. Alternately you can run rtaTrace and direct output to syslog, rtaTrace –osur. Use tail –f /var/log/messages to see printf.

   - ./loadandrun hello.out &
   - <hit enter>
   - ./rtaTrace –osur hello.out
   - hit CTRL-C
   - tail –f /var/log/messages

12) Switch to the **Host Session** (user). In order to build the samples the gpp and dsp makefiles must be edited. This is described in detail in the RTA SDK User’s Guide in the Installation and Setup chapter. What follows is a brief summary (using the environment variables setup above eg. $CGTOOLS, etc):

   - cd $RTASDK
   - Edit $RTASDK/ti/bios/rta/examples/gpp/common/config.mak to point to Montavista and DSP/BIOS Link. In the makefile,
     - DSPLINK_DIR = $(DSPLINK)
     - MVL_TOOLCHAIN = /opt/montavista/previewkit/arm/v4t_le.
- Edit $RTASDK/ti/bios/rta/examples/gpp/rtatrace/makefile
  - RTA_SDK_DIR = $(RTASDK)/ti/bios/rta/sdk

- Edit $RTASDK/ti/bios/rta/sdk/src/pmrdrv/makefile
  - KERNEL_INC_DIR = $(HOME)/montavista/kernel/linux/include

- Edit $RTASDK/ti/bios/rta/sdk/src/chnldrv/makefile
  - RTA_SDK_DIR = $(RTASDK)

- Edit $RTASDK/ti/bios/rta/examples/dsp/common/config.mak to setup the following (tip: make sure no trailing spaces, check for typo’s):
  - RTASDK_INSTALL_ROOT = $(RTASDK)
  - SABIOS_INSTALL_ROOT = $(BIOS)
  - COMPONENT_ROOT = $(CCS_DIR)/c5500
  - C55X_CODEGEN_ROOT = $(CGTOOLS)
  - BIOSLINK_DIR = $(DSPLINK)/dsp/export/BIN/DspBios/OMAP/RELEASE

- Edit $RTASDK/ti/bios/rta/examples/dsp/common/c55rules.mak to modify the assumption about the ccs221_content directory structure. Change the LD55OPTS include paths (around line 22):
  - LD55OPTS = ...
    - -I$(RTDX_ROOT)/lib
    - -I$(CSL_ROOT)/lib
    - -I$(PSL_ROOT)/lib

- 13) Build ARM side rtatrace utility. Change directories to $RTASDK/ti/bios/rta/examples/gpp, clean and build. Successful build produces the executable ‘rtatrace’
  - cd $RTASDK/ti/bios/rta/examples/gpp/rtatrace
  - gmake clean
  - gmake
  - cp rtatrace $OSK_WORK/rta

- 14) Build DSP side hello example.
  - cd $RTASDK/ti/bios/rta/examples/dsp/basic/hello/osk5912
  - edit the makefile, change the name of the dsplink.l55l library specified in the LD55FLAGS to -l$(BIOSLINK_DIR)/dsplink.lib
  - LD55FLAGS = -x -w -m"hello.map" -l$(BIOSLINK_DIR)/dsplink.lib
  - gmake clean
  - gmake
  - chmod a+w $OSK_WORK/rta/hello.out
- 15) Switch to **OSK Serial Session** and run hello world. After printing 'hello world', CPU busy should report something around 4%.
  
  o `cp hello.out $OSK_WORK/rta`

  o `.loadandrun hello.out &`
  o `<hit enter>`
  o `.rtatrace –xosur hello.out`
  o `hit CTRL-C`

- For more info on RTA SDK, see the html docs located at the top of the install directory, `/opt/ti_dsptools/rta_sdk_5.00`. The RTA SDK User’s Guide gives instructions on setting up, using, and building. The API reference is useful for developers but also is the place to go for diagnosing failures as the error codes are listed.

- Step 3.2 complete.
3.3 Build and Run RF6

This section covers building and running RF6 and presumes that DSP/BIOS Link user library has been built (step 3.1 above).

- 1) In the Host session, make a development copy both of the rf_3_11 directories by copying from the golden copy in the install area to the user development area.
  - cp -r /opt/ti_dsptools/rf_3_11_gpp $HOME
  - cp -r /opt/ti_dsptools/rf_3_11_dsp $HOME

- 2) Change directory to the RF6 gpp development area. At the base of rf_3_11_gpp, edit the Rules.make file. Specify the DSPLINK_DIR and MVL_TOOLCHAIN variables. Save the Rules.make file and exit the text editor.
  - cd $RF_GPP
  - <edit> Rules.make
    - DSPLINK_DIR = $(DSPLINK)
    - MVL_TOOLCHAIN = /opt/montavista/previewkit/arm/v4t_le

- 3) Change directories to $RF_GPP/apps/rf6/Linux (/home/<user>/rf_3_11_gpp/apps/rf6/Linux/OMAP) and type gmake. This will build the gpp application. Failures are possible here if the dsplink library was not properly built or if the Rules.make was incorrectly edited.
  - cd $RF_GPP/apps/rf6/Linux/OMAP
  - gmake

- 4) Create an rf6 ‘run’ directory in the NFS mounted share work area. Copy the resulting executable, rf6_gpp, from it’s build location to the ‘run’ location.
  - mkdir $OSK_WORK/rf6
  - cp $RF_GPP/apps/rf6/Linux/OMAP/release/rf6_gpp $OSK_WORK/rf6

- 5) In the Host session, change directories to the development copy of the dsp side of rf6. Starting at the top of the directory, edit config.mak (note: there is a rules.mak, but DSP side is different, user edits config.mak).
  - cd $RF_DSP
  - <edit> config.mak
    - INSTALL_ROOT = $(BIOS)
    - RTDX_ROOT = $(CCS_DIR)/c5500/rtdx
- XDAIS_ROOT = $(CCS_DIR)/c5500/xdais
- CSL_ROOT = $(CCS_DIR)/c5500/csl
- C55X_CODEGEN_ROOT = $(CGTOOLS)

1. Save config.mak and exit editor

6) Since the CCS content is delivered in a structure that matches CCS 2.21 and the RF6 makefile assumes a different directory structure some changes are necessary. To make the CCS 2.21 content work with RF6 makefiles, edit rules.mak located in the same directory as config.mak.

   - cd $RF_DSP
   - <edit> rules.mak
     - modify LD55OPTS (around line 29) to eliminate c5500 extension for rtdx and csl entries, for example, change "–I$(CSL_ROOT)/lib/c5500 \" to "–I$(CSL_ROOT)/lib \"
     - -I$(RTDX_ROOT)/lib
     - -I$(CSL_ROOT)/lib
   - <save and exit>

7) Change directories to the rf6 app area for OSK5912. Clean then build the sources. One source of error on this step is if korn shell is not installed on Linux Host, Tconf will not be able to run successfully.

   - cd $RF_DSP/apps/rf6
   - gmake clean_osk5912
   - gmake build_osk5912

8) Copy the resulting DSP executable to the ‘run’ directory

   - cp $RF_DSP/apps/rf6/projects/osk5912/Debug/app.out $OSK_WORK/rf6

9) Switch to an OSK Serial Session and load the DSP/BIOS Link driver. Make sure the drivers (dsplinkk and pmr_drv) are loaded and running and if need be, load them again (won’t hurt if the load4rta command is run with drivers already loaded). Skip this step if you’ve not been interrupted since section 3.2.

   - cd /work/rta
   - lsmod
   - ./load4rta

10) Change directories in the OSK Serial Session to rf6 ‘run’ directory. Then run the rf6_gpp application which loads and runs the DSP side app.out and supply the codec sampling speed (44.1Khz). RF6 outputs a string “Reference Frameworks 6 audio application started” and offers commands for
changing the volume and filter coefficient set. It is recommended to have a sound source and speakers connected to the OSK5912 so you can hear the audio playing. For more info on RF6 see /opt/ti_tools/doc/spra796a.pdf.

- cd /work/RF6
- ./rf_gpp app.out 44100

- Step 3.3 complete.
3.4 Modify RF6 to add LOG_printf to syslog  [Time: 30 min]

RF6 comes with LOG_printf statements in the c source, however, it is configured to send the LOG_printf data through RTDX which requires a JTAG emulator and CCS attached to the JTAG port on the OMAP5912 to catch the ‘real time’ log data. This section describes the steps necessary to change the RTA transport layer and to configuration for BIOS to enable RTA data through DSP/BIOS Link to ARM Linux syslog.

- 1) Change directories to the rf6 for OSK5912 directory and edit the linker command file, link.cmd. After the line in the linker command file specifying the DSP/BIOS Link driver, add the ‘-l lnkbioslink.a55L’ to link in the library from the RTA SDK.
  - `cd $RF_DSP/apps/rf6/projects/osk5912`
  - `<edit> link.cmd. Around line 46 (just after the RF supplied DSP/BIOS Link driver, dsplink.l55l), add the rta link transport driver:`
    - `-l lnkbioslink.a55L`

- 2) Add rta link transport library to RF6. Note: Since we are using v2.56 of the c55xx linker, the preferred method of adding the search path to the RTA SDK library does not work as the linker has a limit of 7 ‘-I’ linker search paths. RF6 is already at the limit and adding one more would cause an error. There are a couple of options 1) remove the RTDX search path, unfortunately due to a defect, the BIOS generated linker command file still attempts to link with RTDX even when it is configured out. The approach is to replace the RTDX search path, and work around the defect by copying the rtdx library, rtdxx.lib, to the rf6 library directory which is already in the search path. Copy the ‘rtdxx.lib’ from ccs221_contents into the rf6 lib area and then edit the config.mak and rules.mak to replace the RTDX search path with the one for the RTA library. (Note the command detailed in the 3rd bullet is one line
  - `cd $RF_DSP`
  - `<edit config.mak> At the bottom of the file add`
    - `- LNKBIOSLINKLIB_DIR = $(RTASDK)/ti/bios/rtalnk/lib`
  - `<save and exit>`
  - `<edit rules.mak> Replace under LD55_OPTS the –I$(RTDX_ROOT)/lib \ with`
    - `-I$(LNKBIOSLINKLIB_DIR) \`
  - `<save and exit>`
  - `cp $CCS_DIR/c5500/rtdx/lib/rtdxx.lib $RF_DSP/lib`
3) Modify the RF6 configuration file for OSK5912 to configure DSP/BIOS Real Time Analysis to use the correct channel driver. The RTA SDK provides a Tconf include file (bioslink_common.tci) that contains a good starting point for adding the configuration parameters. Copy this file to the RF6 app config directory rename, and modify to suit. Then change to the OSK5912 RF6 directory and edit the application configuration file, appcfg.tcf, and import the Tconf include file. Tconf comes with a single step script debugger which can be invoked by adding "-g" flag to the tconf command line (see RF6 makefile for the Tconf command line). For more information on Tconf, see the DSP/BIOS Tconf User's Guide, spru007e.pdf, located in $BIOS/tconf.doc.

```bash
- cd $RF_DSP/apps/rf6/appConfig
- cp $RTASDK/ti/bios/rta/examples/dsp/common/bioslink_common.tci .
- mv bioslink_common.tci bioslink_rta.tci
- <edit bioslink_rta.tci>
  - Rename all bios.* to tibios.* to match RF6 Tconf object names, ex. tibios.GIO.ENABLEGIO = TRUE. Etc.
  - <Save and exit>
- cd $RF_DSP/apps/rf6/projects/osk5912
- <edit appcfg.tcf>
  - around line 92, just after utils.importFile("appGlobal.tci"), add the line:
    - • utils.importFile ( "bioslink_rta.tci" );
```

4) Build dsp side of rf6 (/home/<user>/rf_3_11_dsp/apps/rf6). Upon completion of a successful build, copy the app.out to the RF6 work directory.

```bash
- cd $RF_DSP/apps/rf6
- gmake clean_osk5912
- gmake build_osk5912
- cp $RF_DSP/apps/rf6/projects/osk5912/Debug/app.out $OSK_WORK/rf6
```

5) To run the Tconf script debugger, look through the output of the build to find the tconf invocation. It starts with /opt/ti_dsptools/bios_5_03/tconf/tconf -Dconfig.platform .... Copy this line using the terminal, and paste into a command prompt. To run the debugger add ‘-g’ just after tconf (before –D).

```bash
- cd $RF_DSP/apps/rf6/projects/osk5912
- opt/ti_dsptools/bios_5_03/tconf/tconf -g -Dconfig.platform="Innovator1510" -Dconfig.importPath="/opt/ti_dsptools/bios_5_03;../../appConfig;../.." appcfg.tcf
- <try 'step over' 'step into', etc.>
```
- 6) Switch to the **OSK Serial Session** window and find out the IP address by running `ifconfig` and looking next to `inet addr` (for example: 192.168.1.106). Next open another Host terminal session and `telnet` to the OSK by supplying the IP address for your board (replace 192.168.1.106 with your OSK IP address). This session will be referred to as the ‘**OSK Telnet session’**.

  - `/sbin/ifconfig`  
    - Note the `inet addr` for the OSK
  - `<launch a new terminal session – this will be OSK Telnet Session>`
  - `telnet <osk inet address>`
  - `<login as root to the OSK>`

- 7) In the **OSK Serial Session** (where you last ran `/sbin/ifconfig`) run `rf6_gpp`. Note: if the board has been restarted, make sure link drivers are loaded and LD_LIBRARY_PATH is set and `load4rta` has installed drivers as in section 3.2. Audio should be heard playing if the OSK is connected to audio source and speakers after running the `rf_gpp` executable supplying `dsp app.out` and sampling frequency (44100).

  - `cd /work/rf6`
  - `./rf_gpp app.out 44100`

- 8) Switch to the **OSK Telnet session**, make sure LD_LIBRARY_PATH is set and run the `rtaTrace` utility. The LOG data will be dumped the screen, CPU Load with debug library and 44100 sampling frequency should be around 25%. Note: setting LD_LIBRARY_PATH is unnecessary if you’ve created the ‘.profile’ and have rebooted the OSK5912 since step 3.2 above. Running `rtaTrace` should output the LOG buffers from the rf6 application.

  - `cd /work/rta`
  - `./rtaTrace –xosur ../rf6/app.out`

- 9) In the OSK Serial Session, quite the rf6 application.

  - `<type q to quit rf6_gpp application>`

- Step 3.4 complete
3.5 Build and Run ‘Loop’ example  [Time: 30 min ]

The ‘Loop’ example supplied with DSP/BIOS Link illustrates basic streaming concepts using DSP/BIOS Link. Data is transferred between a task running on the ARM and another DSP/BIOS task running on the DSP. More detailed info is provided in section F (SAMPLE APPLICATIONS), part 17 (LOOP) in the DSP/BIOS Link User's Guide.

- Change directories to the DSP side of the 'loop' example, located in $DSPLINK/dsp/src/samples/loop. The build depends on the proper configuration of the makefile variables defined in the Section 3.1 above. Some remarks which can safely be ignored will be generated during the build – see DSP/BIOS Link User's Guide for more info. If there are errors building, it’s likely due to an improperly set makefile variable – use ‘gmake VERBOSEQ=1’ instead of ‘gmake –s’ to help diagnose issues with $DSPLINK/make/DspBios/c55xx_5.00_linux.mk.
  
  o cd $DSPLINK/dsp/src/samples/loop
  o gmake –s clean
  o gmake –s

- Create a work directory for 'loop', then copy the DSP executable to the shared filesystem for execution on the OSK5912.
  
  o mkdir $OSK_WORK/loop
  o cp $DSPLINK/dsp/export/BIN/DspBios/OMAP/RELEASE/loop.out $OSK_WORK/loop

- Change directories to the ARM side of the 'loop' example located in $DSPLINK/gpp/src/samples/loop. Clean and build. If errors are encountered, the $DSPLINK/make/Linux/osk5912_mvlpreviewkit3.1.mk should be checked (see section 3.1) and ‘gmake VERBOSEQ=1’ used as a diagnostic.
  
  o cd $DSPLINK/gpp/src/samples/loop
  o gmake –s clean
  o gmake –s

- Copy the release version of the ARM-side executable, loopgpp, to the share area.
  
  o cp $DSPLINK/gpp/export/BIN/Linux/OMAP/RELEASE/loopgpp $OSK_WORK/loop

- Switch to OSK Serial Session and change directories to the to the 'loop' working directory and run the loopgpp executable –supply the dsp executable name (loop.out) as an argument, and specify the buffer.
size (128) and number of transfers (10000). (This assumes /work/rta/load4rta has been run and 
LD_LIBRARY_PATH is set to /work/rta).
  o  cd /work/loop
  o  ./loopgpp loop.out 128 10000

- At this point, the example can now be modified to use the bioslink_rta.tci configuration script and include 
the rta link library as described in the previous section with RF6. This assignment is left for the user.

- Step 3.5 complete.
Addenda Section

Support

This product is not supported through TI Hotline, rather relies on the community support model. Some resources for support are:

Yahoo OMAP group:  http://groups.yahoo.com/group/omap

OMAP Linux User Group:  http://linux.omap.com

Spectrum Digital OMAP site:  http://omap.spectrumdigital.com/osk5912

Linux DSP Tools FAQ located at:  

Miscellaneous

Useful Tips:

Setting Environment variables on the OSK5912:  One way to set LD_LIBRARY_PATH at boot time is to provide a `.profile` file in the root subdirectory.  The instructions are shown below.  First is to change the flash file system to allow writes, then edit .ashrc, and IMPORTANT, change the flash file system back to read-only as this prevents inadvertent writes

- `mount -o remount,rw /`
- `cd /root`
- `<edit> .profile` add the following line:
- export LD_LIBRARY_PATH=/work/rta
  o <save and exit> .profile
  o mount -o remount,ro /
  o re-boot the OSK
  o env