OMAPL138 Software Developers Guide

Translate this page to Translate Show original

Contents

Welcome to the OMAP-L138 Software Developer's Guide

Thanks you for choosing the OMAP-L138 Evaluation Module (EVM) for your application. The purpose of this guide is to get you going with developing software for the OMAP-L138 on a Linux development host only.

Note! This Software Developer's Guide (SDG) supports version 4.xx of the OMAP-L138 DVSDK which is only for Linux host development.

Note! This guide assumes you have already followed the Quick Start Guide (QSG) for setting up your EVM and installing the Digital Video Software Development Kit (DVSDK). If you have not done this yet, please do so now before continuing. You can find a hard copy contained with your EVM. Alternatively you can find the QSG PDF and various other documentation in the 'docs' directory of the DVSDK installation directory.

Note! All instructions in this guide are for Ubuntu 10.04 LTS. At this time, it is the only supported Linux host distribution for development.

Note! In previous DVSDK releases there has been a *Getting Started Guide* explaining how to set up the DVSDK. This document replaces and extends the Getting Started Guide for DVSDK 4.xx.

Throughout this document there will be commands spelled out to execute. Some are to be executed on the Linux development host, some on the Linux target and some on the u-boot (bootloader) prompt. They are distinguished by different command prompts as follows:

```
host $ <this command is to be executed on the host>
target # <this command is to be executed on the target>
u-boot :> <this command is to be executed on the u-boot prompt>
```

Starting your software development

Your DVSDK should be installed before you continue. Throughout this document it will be assumed you have an environment variable *DVSDK* which points to where your DVSDK is installed. You can set it as follows (the following assumes that DVSDK was installed at default location):

```
host $ export DVSDK="${HOME}/ti-dvsdk_omapl138-evm_xx_xx_xx_xx"
```

Setting up the DVSDK

The DVSDK comes with a script for setting up your Ubuntu 10.04 LTS development host as well as your target boot environment. It is an interactive script, but if you accept the defaults by pressing return you will use the recommended settings. This is recommended for first time users. Note that this script requires internet access as it will update your Ubuntu Linux development host with the packages required to develop using the

DVSDK. Before executing the script make also sure that the SD card received with the EVM or an SD card prepared as described in the section below "How to create an SD card" is inserted in the EVM SD card reader.

Execute the script using:

```
host $ ${DVSDK}/setup.sh
```

If you accepted the defaults during the setup process, you will now have set up your development host and target to:

1. Boot the Linux kernel from your development host using TFTP. On your development host the Linux kernel is fetched from */tftpboot* by default.
2. Boot the Linux file system from your development host using a Network File System (NFS). On your development host the Linux target file system is located at *${HOME}/targetfs*
3. Minicom is set up to communicate with the target over RS-232. If you want to use a windows host for connecting to the target instead, see the #Setting_up_Tera_Term section.

If you start minicom on your Linux development host using *minicom -w* (or Tera Term on Windows) and power cycle the EVM, Linux will boot.

After Linux boots up, login into the target using **root** as the login name.

Note! The Matrix Application Launcher GUI is not launched automatically in the development filesystem. If you would like to start it, execute the following command on the target board:

```
target # /etc/init.d/matrix-gui-e start
```

If your kit includes an LCD display, the first time the Matrix GUI is executed from NFS, you'll go through a LCD touchscreen calibration process. The calibration process is important as other application in additional to the Matrix GUI require calibration to run successfully. You can also run the calibration manually without starting the Matrix GUI by executing the following command on the target board:

```
target # ts_calibrate
```

If the Matrix is running, make sure you have terminated the Matrix GUI before running any other applications from the command line:

```
target # /etc/init.d/matrix-gui-e stop
```

Note! if you select the "Primary display output" to DVI, then you might be required to have mouse support to work with graphical user interface, you must first clear these variables prior to running the demos to enable the mouse:

target # export QWS_MOUSE_PROTO=

target # export TSLIB_TSDEVICE=

Writing your own "Hello World!" application and executing it on the target

This section shows how to create/build an application on your host development PC and execute a basic Linux application on your booted target filesystem.

1. Create your own work directory on the host PC and enter it:

```
host $ mkdir ${HOME}/workdir
host $ cd ${HOME}/workdir
```

2. Create a new C source file:

```
host $ gedit helloworld.c
```

Enter the following source code:

```
#include <stdio.h>

int main()
{
    printf("Hello World!\n");
}
```

Save the file and exit.

3. Create a basic makefile:

```
host $ gedit Makefile
```

Enter the following:

```
# Import the variables from the DVSDK so that you can find the DVSDK components
include ${DVSDK}/Rules.make

helloworld:
# Make sure that you use a tab below
    $(CSTOOL_PREFIX)gcc -o helloworld helloworld.c
```

Save the file and exit. Note that the gap before $(CSTOOL_PREFIX)gcc corresponds to a tab. If it is filled with spaces instead you will get build errors.

4. Make sure the $DVSDK variable is still set using:

```
host $ echo $DVSDK
```

This command should print your DVSDK installation directory. If it doesn't, you will have to set it again as described in the beginning of this document. Compile the application:

```
host $ make helloworld
```

As a result, an executable called `helloworld` is generated in `${HOME}/workdir`

5. You now have your own application, but you need to create a directory and copy it to your NFS exported filesystem to make it visible by the target:

```
host $ mkdir ${HOME}/targetfs/home/root/omapl138
```

```
host $ cp helloworld ${HOME}/targetfs/home/root/omapl138
```

6. On your target this application will be accessible from `/home/root/omapl138/helloworld`. Execute it on your target:

```
target # /home/root/omapl138/helloworld
```

You should now see the following output:

```
Hello World!
```

Congratulations! You now have your own basic application running on the target.

Running the pre-installed applications on the target file system

The filesystem comes with a number of prebuilt applications (which can be rebuilt inside the DVSDK). This section shows how to execute those applications in the provided filesystem.

- A system wide loadmodule script is provided in the filesystem. This script loads the various TI kernel modules as part of the Linux init process which are needed by the various applications provided in the filesystem. The file can be found in the following location on the target:

```
target # vi /etc/init.d/loadmodule-rc
```

The script leverages a new feature of CMEM 2.0 to use general purpose heaps instead of specific pools. More information on this can found in the CMEM Overview on TI's embedded processor wiki page.

You can use the load/unload/restart parameter to load and unload the various TI kernel modules at any time by executing the following on the target:

```
target # /etc/init.d/loadmodule-rc start|stop|restart
```

Running the DSPLink examples

The DSPLink comes with a few sample application. To run them enter this directory on the target

```
target # cd /usr/share/ti/ti-dsplink-examples/
```

Execute the following script to run the example application

```
target # ./ti-dsplink-examples-run.sh
```

The target terminal window will output the results of the examples executed. The examples can be run individually (vi ti-dsplink-examples-run.sh for proper parameters to individual examples).

Running the C6Run Example Applications

The C6run package comes with example and test applications to demonstrate its functionality. Most applications exist as ARM version (ending in "_arm") and a DSP version (ending in "_dsp"), compiled from the same source. The intent of the two versions is to show that the same code running on the two different processors behaves the same. To run the applications, enter the following directory on the target:

```
target # cd /usr/share/ti/c6run-apps/
```

Setup

We'll unload and then re-load the necessary kernel modules needed by the application. If other applications haven't previous loaded any of the modules (CMEM, DSPLink, potentially LPM as well), unloading the modules will not be necessary (but it won't hurt either).

```
target # ./unloadmodules.sh
target # ./loadmodules.sh
```

C6RunApp Examples

Enter into the examples/c6runapp directory. This directory contains examples that use the c6runapp tool to move an entire app (starting with main() function) to the DSP. List the directory content to see all the example provided on the filesystem.

```
target # cd examples/c6runapp
target # ls
```

Run the following commands to execute a simple "hello world" example. The first one runs completely on the ARM, while the second one is executed on the DSP (but results are passed back to the ARM for display on the Linux console).

```
target # cd hello_world
target # ./hello_world_arm
target # ./hello_world_dsp
```

Move to the emqbit directory and run the applications there. These applications are floating point benchmarking examples. Note that on different platforms the ARM and DSP cores have differing support for floating point operations, so results may vary.

```
target # cd ../emqbit
target # ./bench_arm
target # ./bench_dsp
target # ./cfft_arm
target # ./cfft_dsp
```

C6RunLib Example

Next move to the c6runlib emqbit example directory and run the applications there. This example is different from the above emqbit example, in that the application, built using c6runlib and the ARM GCC compiler, is partitioned between the ARM core and the DSP core. Data buffers are allocated on the ARM core and passed to the DSP where they are operated on. This adds overhead to the execution process compared to the c6runapp case above, where allocation and operation happened on the same core.

```
target # cd ../../c6runlib/emqbit
target # ./bench_arm
target # ./bench_dsp
target # ./cfft_arm
target # ./cfft_dsp
```

Note: The c6run application required a larger CMEM block thus using the components provided loadmodule script was required to successfully run the application. To revert back to using the system-wide TI CMEM kernel module configuration settings run the following command on the target. This is required for successful run of the various other applications:

```
target # /etc/init.d/loadmodule-rc restart
```

For More Info

If you are interested in how these example are built and what the source code looks like, please look in the 'examples' directory of the C6Run source package. There are also a collection of test cases that can be examined in the 'test' directory of the source package.

Details on the C6Run usage can be found on the TI embedded processor wiki.

Running the C6Accel apps

The C6Accel package comes with a small test application benchmarks all the DSP kernel APIs for fixed point and floating point calculations. To run the application, enter the following directory on the target:

```
target # cd /usr/share/ti/c6accel-apps/
```

Load the C6Accel specific kernel modules:

```
target # ./loadmodules_omapl138_c6accel.sh
```

Execute the following command to run the example application

```
target # ./c6accel_app
```

The application benchmarks all the DSP kernel API calls in C6Accel and writes the benchmark data to file (benchmarking.txt) in the /usr/share/ti/c6accel-apps directory. To view the file, execute

```
target # vi /usr/share/ti/c6accel-apps/benchmarking.txt
```

Running the DMAI apps

The *Davinci Multimedia Application Interface* (DMAI) comes with small sample applications (including source code). Refrain from using **Ctrl-C** to terminate applications as un-expected results may occur. To run them enter this directory on the target:

```
target # cd /usr/share/ti/ti-dmai-apps/
```

Then, load the kernel modules:

```
target # /etc/init.d/loadmodule-rc restart
```

The DVSDK comes with DMAI, and the following example invocations are known to work. For more information on how to run DMAI applications, refer to the DMAI user guide shipped with the DMAI installation in the DVSDK.

Audio

To decode an AAC file to Line out (connect headphone to listen to output) execute:

```
target # ./audio_decode1_omapl138.x470MV -c aachedec -e decode \
-i  /usr/share/ti/data/sounds/davincieffect.aac -n 1000
```

To decode an AAC file to a PCM file execute:

```
target # ./audio_decode_io1_omapl138.x470MV -c aachedec -e decode \
-i  /usr/share/ti/data/sounds/davincieffect.aac -n 1000 -o output.pcm
```

Display

To display a test pattern on the touchscreen LCD without using any codecs execute:

```
target # ./video_display_omapl138.x470MV -y 16 -O lcd --display_buffer 2
```

Video

To decode 30 frames from an H.264 encoded video to a YUV file execute:

```
target # ./video_decode_io2_omapl138.x470MV -c h264dec -e decode \
-i /usr/share/ti/data/videos/davincieffect_480x272.264 -n 30 -o output.yuv
```

To encode 30 frames of resolution 480x272 from a YUV file to an H.264 BP encoded file execute:

```
target # ./video_encode_io1_omapl138.x470MV -c h264enc \
-i output.yuv -o output.264 -r 480x272 -n 30
```

Speech

To decode a G.711 speech file to a PCM file execute:

```
target # ./speech_decode_io1_omapl138.x470MV -c g711dec -e decode \
-i /usr/share/ti/data/sounds/davincieffect.g711 -o output.pcm
```

To encode a G.711 speech file from the previously generated PCM file execute:

```
target # ./speech_encode_io1_omapl138.x470MV -c g711enc -e encode \
-i output.pcm -o output.g711
```

Image

To decode a JPEG image to a yuv file execute:

```
target # ./image_decode_io1_omapl138.x470MV -c jpegdec -e decode \
-i /usr/share/ti/data/images/remi003_422i.jpg -o output.yuv
```

To encode a JPEG image from the previously generated yuv file execute:

```
target # ./image_encode_io1_omapl138.x470MV -c jpegenc -e encode \
-i output.yuv -o output.jpg -r 720x576 --iColorSpace 3 --oColorSpace 1
```

The input parameters depends on the configuration of the input YUV file. In this case, the input file color space format is YUV422 ILE.

To know more about the color space values as supported by the application, execute:

```
target # ./image_encode_io1_omapl138.x470MV -h
```

Running the Qt/Embedded examples

The Qt embedded comes with some examples applications. To see the examples that are available, check out this directory on the target:

```
target # cd /usr/bin/qtopia/examples
target # ls
```

If the LCD touchscreen hasn't been calibrated before, you can start the Matrix Launcher Application as described in the #Setting_up_the_DVSDK section or run the following command. Use a dull device to touch the various points on the screen as prompted to calibrate the LCD

```
target # ts_calibrate
```

Terminate the Matrix application by touching the EXIT button on the LCD if it was started above.

You also need to export a couple of variables to enable the LCD touchscreen for the QT applications as follows:

```
target # export TSLIB_TSDEVICE=/dev/input/touchscreen0
target # export QWS_MOUSE_PROTO=Tslib:/dev/input/touchscreen0
```

Execute the following command to run Qt/e calendar example application.

```
target #  cd /usr/bin/qtopia/examples/richtext/calendar
target # ./calendar -qws -geometry 480x240+0+0
```

After you see the calendar interface, hit **CTRL-C** to terminate it or click on the **X** on the top right hand corner of the calendar window.

Running GStreamer pipelines

The DVSDK comes with GStreamer, and the following pipelines are known to work. Refrain from using **Ctrl-C** to terminate gst pipelines as un-expected results may occur. Pipelines will terminate on their own.

Reload the kernel module before running gstreamer pipelines :

```
target # /etc/init.d/loadmodule-rc restart
```

Note: Before running these pipelines you'll need the LCD Touchscreen attached and speaker/headphone connected to the Line-out stereo 3.5 mm jack for audio streaming.

Some of the GStreamer pipelines below use a sample vidoetestsrc element as a source. This element is a color bar test pattern.

This pipeline encodes H.264 video generated from videoetestsrc element

```
target # gst-launch videotestsrc num-buffers=1000 ! TIVidenc1 \
codecName=h264enc engineName=codecServer ! filesink \
location=sample.264 -v
```

To playback the H.264 encoded videotestsrc element, run the following pipeline

```
target # gst-launch filesrc location=sample.264 ! typefind ! TIViddec2 ! \
queue ! TIC6xColorspace engineName=codecServer ! queue ! tidisplaysink2 -v
```

This pipeline encodes MPEG-4 video generated from videoetestsrc element

```
target # gst-launch videotestsrc num-buffers=1000 ! TIVidenc1 \
codecName=mpeg4enc engineName=codecServer ! filesink \
location=sample.m4v -v
```

To playback the MPEG-4 encoded videotestsrc element, run the following pipeline

```
target # gst-launch filesrc location=sample.m4v ! typefind ! TIViddec2 ! \
queue ! TIC6xColorspace engineName=codecServer ! queue ! tidisplaysink2 -v
```

This pipeline decodes a provided H.264 encoded video stream

```
target # gst-launch filesrc location=/usr/share/ti/data/videos/davincieffect_480x272.264 ! \
typefind ! TIViddec2 ! queue ! TIC6xColorspace engineName=codecServer ! queue ! \
tidisplaysink2 -v
```

This pipeline decodes a provided MPEG-4 video encoded video stream

```
target # gst-launch filesrc location=/usr/share/ti/data/videos/davincieffect_480x272.mpeg4 ! \
typefind ! TIViddec2 ! queue ! TIC6xColorspace engineName=codecServer ! queue ! \
tidisplaysink2 -v
```

This pipeline decodes a provided MPEG-2 video encoded video stream

```
target # gst-launch filesrc location=/usr/share/ti/data/videos/davincieffect_480x272.m2v ! \
TIViddec2 codecName=mpeg2dec engineName=codecServer ! queue ! TIC6xColorspace \
engineName=codecServer ! queue ! tidisplaysink2 -v
```

This pipeline decodes a provided AAC audio file for 100 buffers

```
target # gst-launch filesrc location=/usr/share/ti/data/sounds\
/davincieffect.aac num-buffers=100 ! typefind ! TIAuddec1 ! alsasink -v
```

Note: If you would like to listen to more of the clip, you can increase the *num-buffers=100* line from above command or remove it completely to hear the entire clip.

This pipeline decodes a provided MP4 (H.264 + AAC) file for 1000 buffers

```
target # gst-launch -v filesrc location=/usr/share/ti/data/videos\
/davincieffect_480x272.mp4 num-buffers=1000 ! qtdemux name=demux demux.audio_00 ! \
queue max-size-buffers=8000 max-size-time=0 max-size-bytes=0 ! TIAuddec1 ! \
 alsasink demux.video_00 ! queue ! TIViddec2 ! TIC6xColorspace \
engineName=codecServer ! queue ! tidisplaysink2
```

Note: If you would like to see more of the clip, you can increase the *num-buffers=1000* line from above command or remove it completely to see the entire clip.

Running the Audio SOC example

To run the Audio Soc example a different kernel image (with ALSA driver disabled) must be created and used to boot-up the board. You'll also need to connect headphones or speakers to the **Line Out** stereo 3.5 mm jack on the target EVM.

On the Linux Host development environment (where DVSDK was installed) rebuild the base components as described in the #Rebuilding_the_DVSDK_components. This step is not necessary if it has been previously performed.

```
host $ cd ${DVSDK}
host $ make clean
host $ make components
```

Once the re-build completes, we will re-build the Linux kernel specifically for the Audio SOC example and install it (using sudo privileges) on the target file-system as follows:

```
host $ make audio_soc_example_kernel
host $ sudo make audio_soc_example_kernel_install
```

Copy the newly built kernel from the target file-system installation directory to the host's tftpboot directory:

```
host $ cp ${HOME}/targetfs/boot/uImage_audioSoc /tftpboot/.
```

Then re-build the Audio Soc example as follows:

```
host $ make audio_soc_example
```

Finally install the application examples on the target file-system as follows (using sudo privileges):

```
host $ sudo make audio_soc_example_install
```

Start minicom or Tera Term and power-cycle the board. Hit any key to stop the boot process.

At the U-boot prompt on the the target board, type the following commands:

```
u-boot :> setenv bootfile_org ${bootfile}
u-boot :> setenv bootfile uImage_audioSoc
u-boot :> saveenv
u-boot :> boot
```

The board will boot using the new kernel image. Login into the target using **root** as the login name.

On the target board go to:

```
target # cd /usr/share/ti/audio_soc_example/Release
```

We need to unload the TI system modules first as follows:

```
target # /etc/init.d/loadmodule-rc stop
```

To run the application, DSPLink module has to be inserted into the kernel as follows:

```
target # mknod /dev/dsplink c 230 0
target # insmod ./dsplinkk.ko
```

Then execute:

```
target # ./audioSoc_gpp audioSoc_dsp.out ../davincieffect_clip.pcm
```

To listen to the raw audio file clip, connect speakers/headphones to the Line-out stereo 3.5 mm jack on the board. The application will stream the PCM file from the ARM processor to the Line-out audio jack via the DSP driver peripherals.

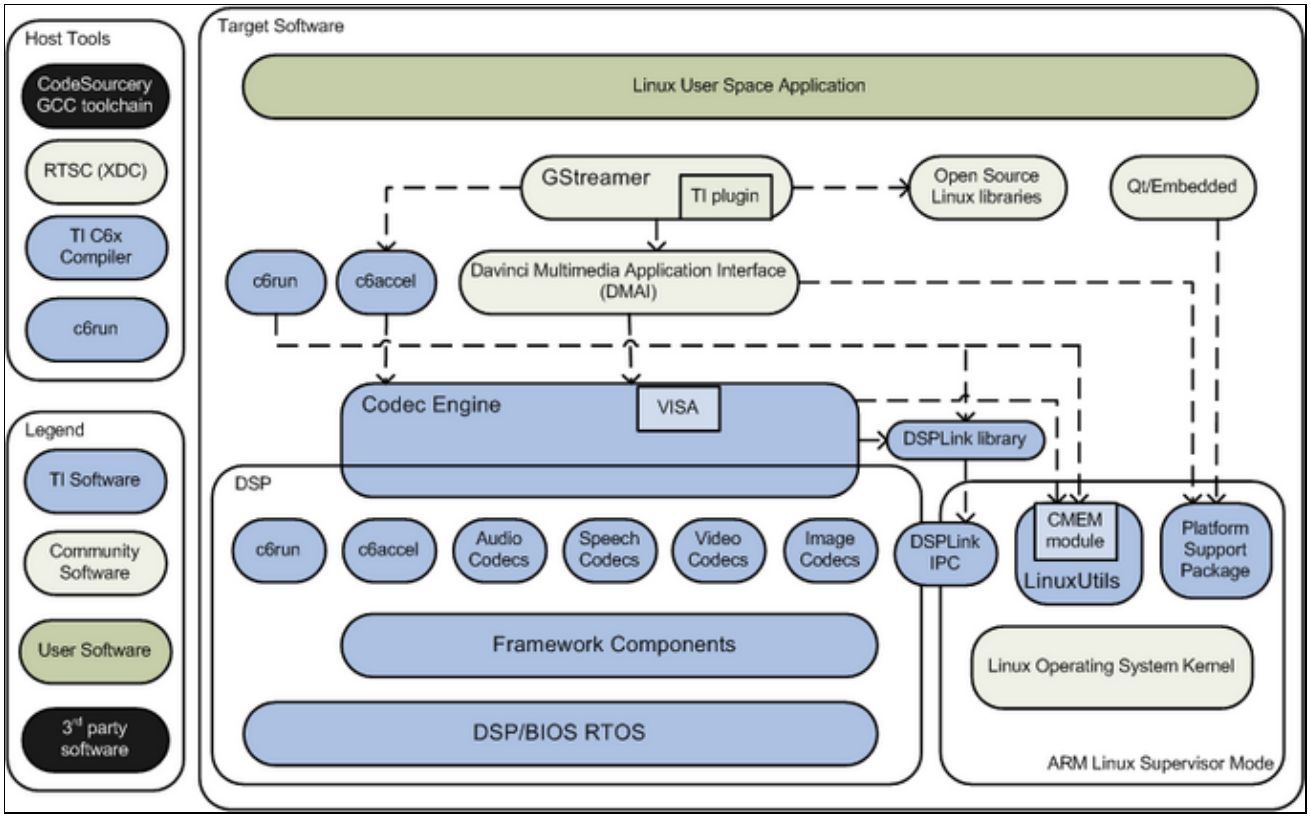
Note: The kernel used for the Audio SOC example has the Advanced Linux Sound Architecture (ALSA) driver disabled since audio is being processed on the DSP. If you would like to go back to the previous running Linux kernel, you'll need to start minicom or Tera Term and power-cycle the board. Hit any key to stop the boot process.

At the U-boot prompt on the the target board, type the following commands:

```
u-boot :> setenv bootfile ${bootfile_org}
u-boot :> saveenv
u-boot :> boot
```

You are now back to booting the Linux kernel prior to running the Audio SOC example.

DVSDK software overview and support



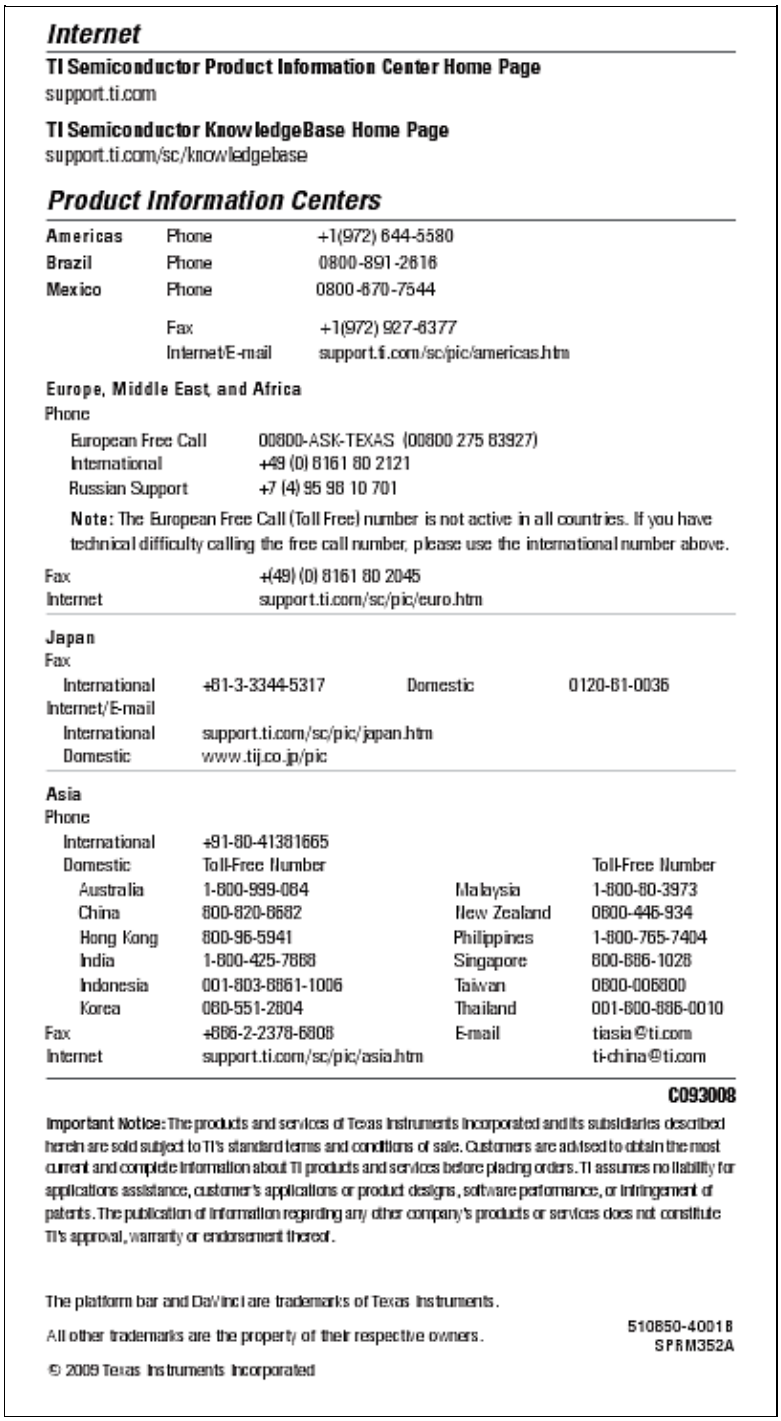
Overview of the DVSDK Software stack

The DVSDK contains many software components. Some are developed by Texas Instruments (*blue* in the diagram above) and some are developed in and by the open source community (*grey* in the diagram above). TI contributes, and sometimes even maintains, some of these open source community projects, but the support model is different from a project developed solely by TI. The table below lists how to get support for each component.

Component(s)	Type	Support
DVSDK demos, Codec Engine, Multimedia Codecs, Platform Support Package, Framework Components, DSP/BIOS, DSPLink, c6accel etc.	Texas Instruments Developed Software	In addition to the support channels listed at #TI Worldwide Technical Support you can use the following support channels: The community Linux forum is monitored by TI support and engineering teams and can be used for asking questions about development using this SDK. The multimedia codecs forum is a separate forum for discussing the multimedia codecs. If you have a bug tracking number (starting with *SDOCM*) you can track the issue using the SDOWP bug tracking web access

Davinci Multimedia Application Interface	Open Source Project	DMAI community project
GStreamer	Open Source Project	GStreamer community project
GStreamer plugin for accelerated multimedia	Open Source Project	gst-ti community project
Qt/Embedded	Open Source Project	Qt/Embedded community project
RTSC (XDC)	Open Source Project	RTSC community project
Linux kernel	Open Source Project	Linux kernel community project

TI Worldwide Technical Support



Internet

TI Semiconductor Product Information Center Home Page
support.ti.com

TI Semiconductor KnowledgeBase Home Page
support.ti.com/sc/knowledgebase

Product Information Centers

Americas	Phone	+1(972) 644-5580
Brazil	Phone	0800-891-2616
Mexico	Phone	0800-670-7544
	Fax	+1(972) 927-6377
	Internet/E-mail	support.ti.com/sc/pic/americas.htm

Europe, Middle East, and Africa
Phone

European Free Call	00800-ASK-TEXAS (00800 275 83927)
International	+49 (0) 8161 80 2121
Russian Support	+7 (4) 95 98 10 701

Note: The European Free Call (Toll Free) number is not active in all countries. If you have technical difficulty calling the free call number, please use the international number above.

Fax	+(49) (0) 8161 80 2045
Internet	support.ti.com/sc/pic/euro.htm

Japan
Fax

International	+81-3-3344-5317	Domestic	0120-81-0036

Internet/E-mail

International	support.ti.com/sc/pic/japan.htm
Domestic	www.tij.co.jp/pic

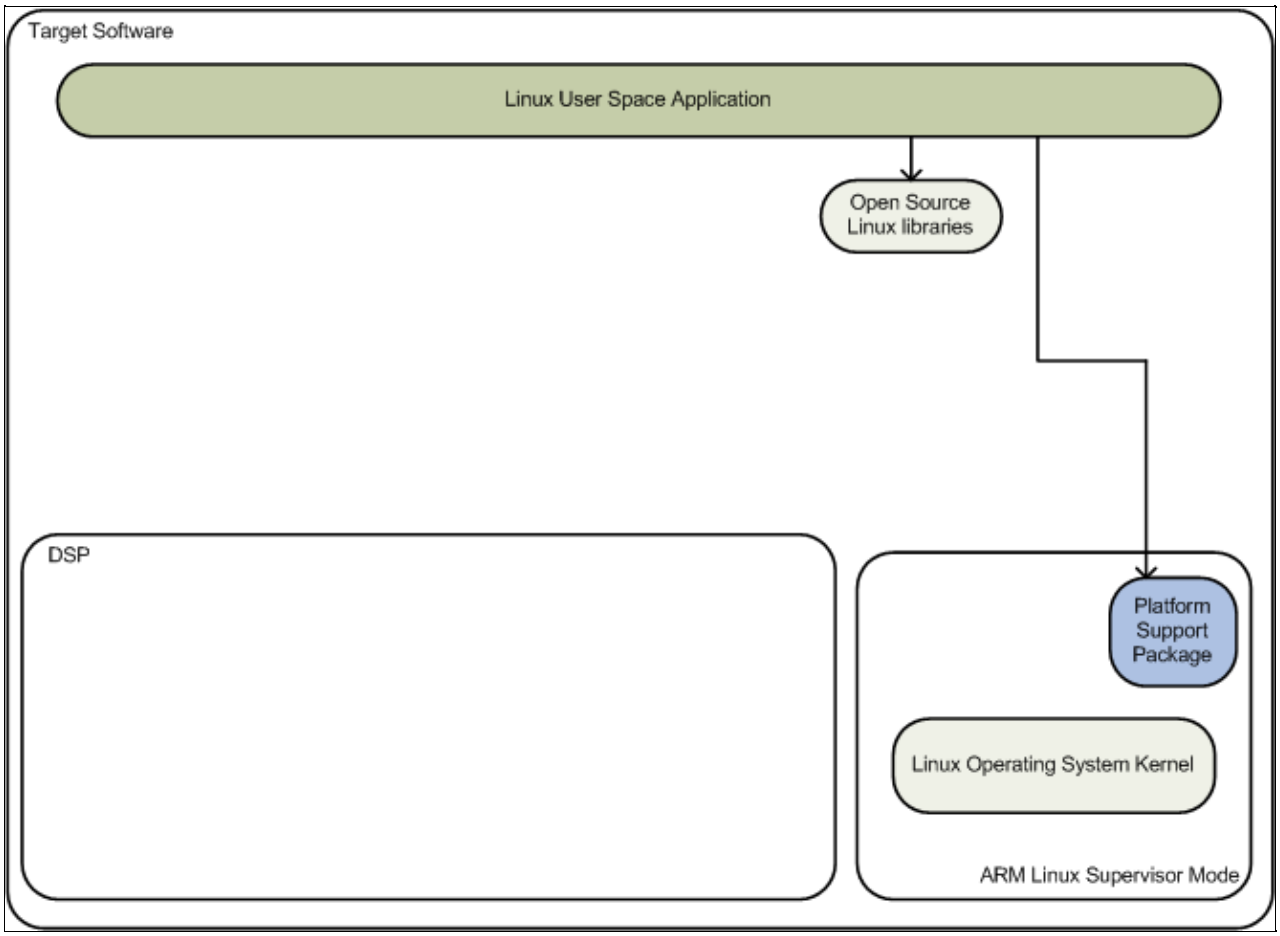
Asia
Phone

International	+91-80-41381665		
Domestic	Toll-Free Number		Toll-Free Number
Australia	1-800-999-084	Malaysia	1-800-80-3973
China	800-820-8682	New Zealand	0800-446-934
Hong Kong	800-96-5941	Philippines	1-800-765-7404
India	1-800-425-7888	Singapore	800-886-1028
Indonesia	001-803-8861-1006	Taiwan	0800-006800
Korea	080-551-2804	Thailand	001-800-886-0010
Fax	+886-2-2378-6808	E-mail	tiasia@ti.com
Internet	support.ti.com/sc/pic/asia.htm		ti-china@ti.com

C093008

510850-40018
SPRM352A

Creating a Linux application



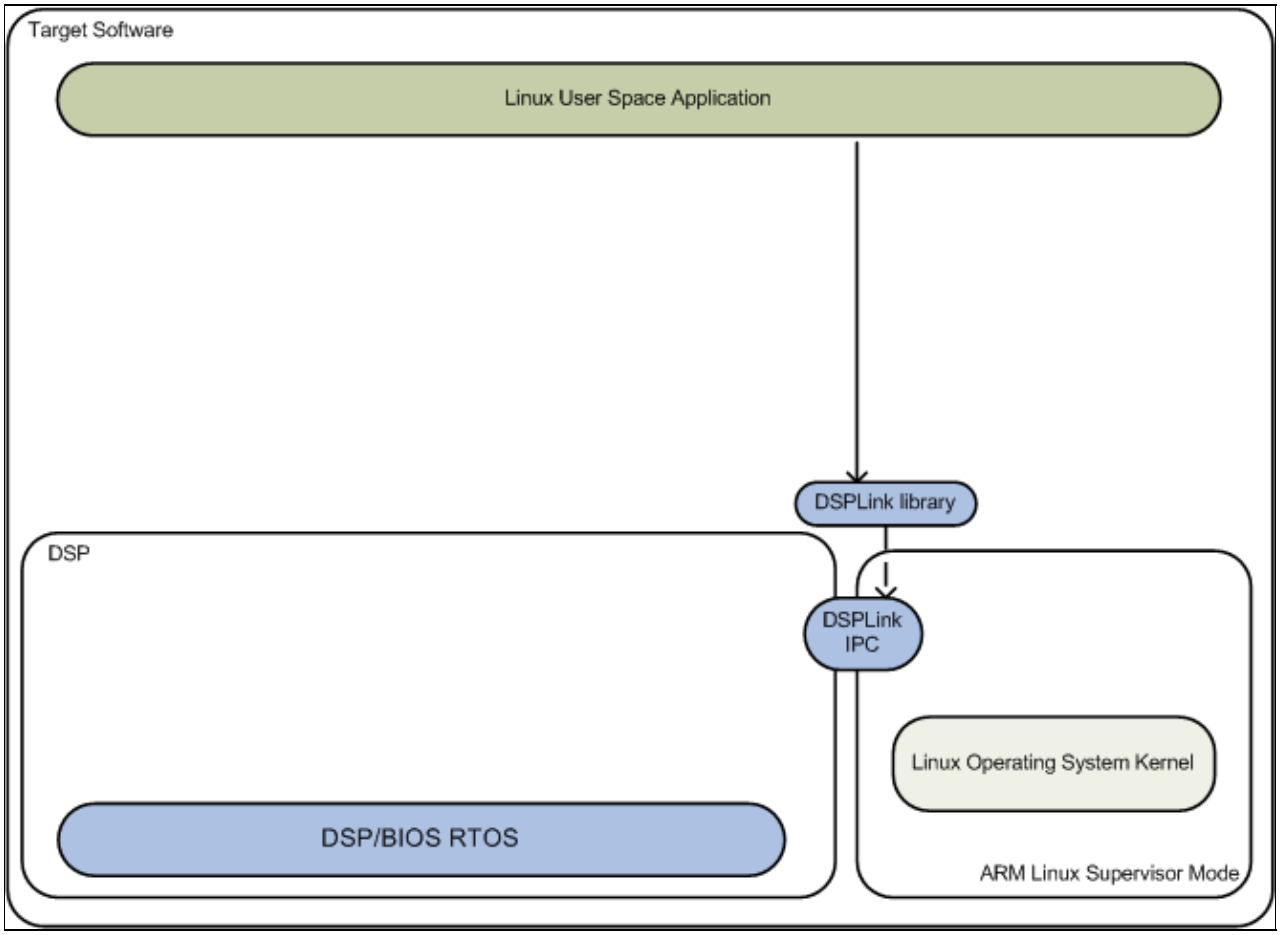
Overview of a basic Linux application component usage

While creating a basic Linux application you are typically using the following components of the stack (the rest are greyed out above):

Component	Purpose in this application	Location in the DVSDK
GCC toolchain	Cross compiler for generating ARM Linux binaries.	linux-devkit directory under the DVSDK
Open Source Linux libraries	Provides libraries such as libpng, libusb, libz, libcurl etc.	linux-devkit/arm-arago-linux-gnueabi/lib and linux-devkit/arm-arago-linux-gnueabi/usr/lib/
Platform Support Package	Provides device drivers for the EVM and documentation and examples to support them.	psp
Linux kernel	The Linux kernel with the PSP device drivers	psp/linux-kernel-source

You can find examples all over the web on how to write this type of application. The PSP examples are a good reference on how to access the peripheral drivers specific to this platform.

Creating a DSPLink application



Overview of a DSPlink application component usage

DSPLink is foundation software for the inter-processor communication across the GPP-DSP boundary. It provides a generic API that abstracts the characteristics of the physical link connecting GPP and DSP from the applications. It eliminates the need for customers to develop such link from scratch and allows them to focus more on application development. This software can be used across platforms:

- Using SoC (System on Chip) with GPP and one DSP.
- With discrete GPP and DSP.

DSPLink provides several features and capabilities that make it easier and more convenient for developers using a multi-core system:

- Provides a generic API interface to applications
- Hides platform/hardware specific details from applications
- Hides GPP operating system specific details from applications, otherwise needed for talking to the hardware (e.g. interrupt services)
- Applications written on DSPLink for one platform can directly work on other platforms/OS combinations requiring no or minor changes in application code
- Makes applications portable
- Allows flexibility to applications of choosing and using the most appropriate high/low level protocol
- Provides scalability to the applications in choosing only required modules from DSPLink.

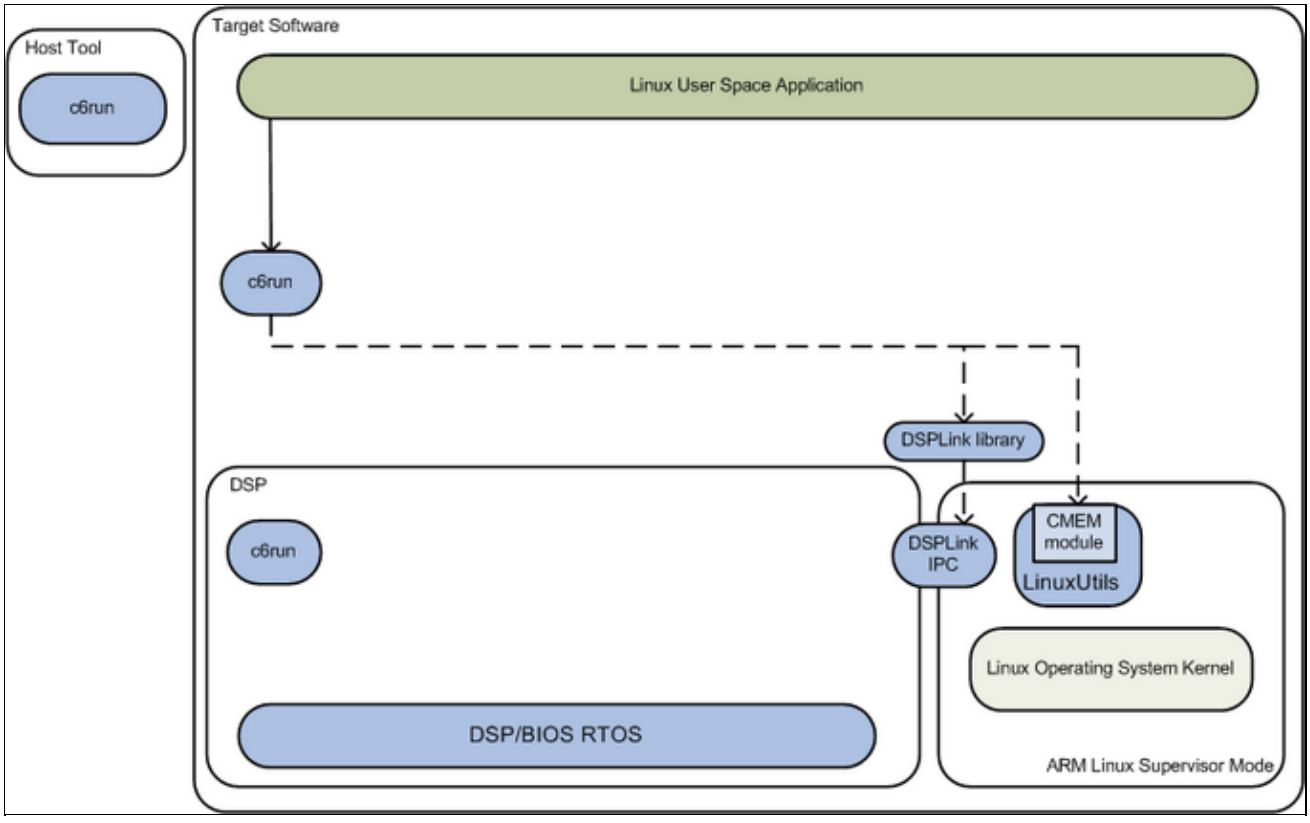
In addition to the components used for the basic Linux app, these are used (and the rest is greyed out in the diagram above):

Component	Purpose in this application	Location in the DVSDK
DSP/BIOS	Real-Time Operation System for TI DSPs	dspbios_x_xx_xx_xx
DSPLink	GPP to DSP processor communication link for passing messages and data in multiprocessor systems	dsplink_x_xx_xx_xx
C6000 Code Generation Tools	TI DSP code generation tools	cgt6x_x_x_xx

Good application examples to start from include:

- The sample applications (dsplink_x_xx_xx_xx/dsplink/gpp/src/samples and dsplink_x_xx_xx_xx/dsplink/dsp/src/samples) provide simpler and smaller examples on how to use DSPLink.
- On some platforms, an Audio SOC example application (audio_soc_example_x_xx_xx_xx) is delivered that leverages DSPLink to perform audio processing and output the data via the DSP peripheral drivers. This example illustrates how DSP-side peripheral drivers run in conjunction with the Linux kernel application on a ARM processor. More information on this example can be found at http://processors.wiki.ti.com/index.php/Audio_Soc_example

Creating a C6Run application



Overview of a basic C6Run application component usage

The C6Run package is to ease initial development and loading of DSP code for ARM developers who are familiar with building applications for the Linux OS using an ARM GCC cross-compiler. The project consists of two main components:

1. A build system to create back-end libraries from the various TI software technologies and the code of the C6Run project itself
2. Front-end scripts that wrap the TI C6000 code generation tools in a GCC-like interface and also make use of the back-end build system to create ARM-side executables or libraries that transparently make use of the DSP.

There are two uses of the C6Run project, exposed through two different front-end scripts. They are called **C6RunLib** and **C6RunApp**.

- **C6RunLib** works to build a static ARM library from C source files that can be linked with an ARM application and provide access to the DSP when library functions are called. This allows the user to keep portions of the application on the ARM and move other portions to the DSP.

- **C6RunApp** tool acts as a cross-compiler for the DSP, allowing portable C applications to be rebuilt for the C6000 DSP core of various Texas Instruments heterogeneous (ARM+DSP) processors. The C6RunApp front-end consists of a single script, called c6runapp-cc. This use of this script matches, as much as possible, the use of GCC. It can compile C code to C6000 object files and link the C6000 object files into an application. When performing linking operations, the tool makes use of a number of steps (including linking using the C6000 code generation tools) to create an ARM-side executable from the DSP object files.

In addition to the components used for the Linux app, these are used (and the rest is greyed out in the diagram above):

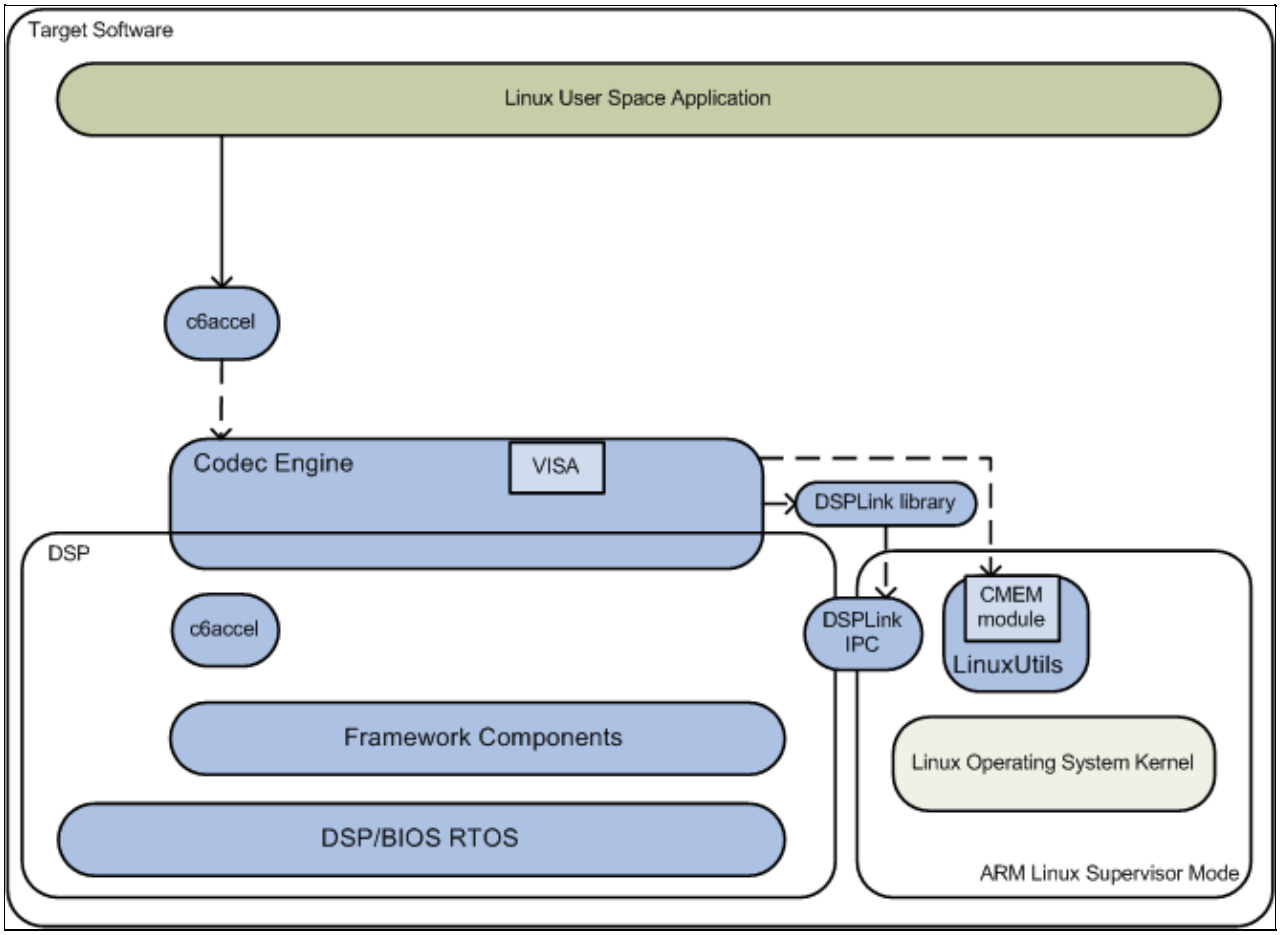
Component	Purpose in this application	Location in the DVSDK
LinuxUtils	Linux specific utilities for Framework Components, used for allocating physically contiguous memory (CMEM module, see this wiki topic for more information) for sharing data between the ARM and DSP.	linuxutils_xx_xx_xx_xx
RTSC (XDC)	Tool required to configure and build DSP/BIOS real-time kernel for the DSP.	xdctools_xx_xx_xx_xx
Local Power Manager	TI power management package (not required for all platforms)	local_power_manager_x_xx_xx_xx
DSP/BIOS	Real-Time Operation System for TI DSPs	dspbios_x_xx_xx_xx
DSPLink	GPP to DSP processor communication link for passing messages and data in multiprocessor systems	dsplink_x_xx_xx_xx
C6000 Code Generation Tools	TI DSP code generation tools	cgt6x_x_x_xx

Good application examples to start from:

- The C6Run package contains sample applications to test/validate the functionality. The applications are located in the c6run_xx_xx_xx_xx/examples and the c6run_xx_xx_xx_xx/test directories. Each example includes full source and standard makefiles.
- There is a QT-based fractal example that leverages C6Run to perform the fractal computation on the DSP. Information on how to build and run the example can be found at: C6Run QT Fractal Example

For more information on C6Run visit the TI Embedded Processors wiki, C6Run Project Page.

Creating a C6Accel application



Overview of a basic C6Accel application component usage

The C6Accel package wraps key DSP software kernels in an xDAIS algorithm which can be invoked from the ARM side using simple API calls. C6Accel can be used in a plug and play like any other codec used for encoding and decoding audio and video streams. C6Accel is built in the codec engine compliant IUniversal framework and can be used on various DSP only and ARM + DSP devices.

The purpose of C6Accel is to provide the ARM user with the compute power of the DSP on computational intense tasks like running Color Space Conversion, Filtering or Image/Signal Processing algorithm. The library of DSP kernels wrapped in C6Accel are optimized for performance on the DSP core and would allow the ARM user to use the DSP as an accelerator for their application. By using these routines, the ARM

developer can develop a more compelling application by achieve execution speeds considerably faster than equivalent C code written on ARM. In addition, by providing ready-to-use DSP kernels, C6Accel can significantly shorten the ARM application development time.

The benefits of using C6Accel include:

1. **Ready to use kernels**: Library of Optimized DSP kernels wrapped in a single package. Reduces learning curve and time to market.
2. **Easy to interface**: ARM side API library abstracts complexities while invoking DSP functionality from ARM application
3. **Easy Portability**: Fully compatible with most TI C6x devices
4. **Efficient multiple call execution**: Capabilty to chain kernel calls using single call to codec engine
5. **Easy Evaluation of DSP performance:** DSP kernel Benchmarks (cycle and code size) provided in C6Accel aid in evaluating performance that can be leveraged from the DSP and make informed decisions while developing applications
6. **Parallel processing:** Asynchronous calling mode enables parallel processing on DSP and ARM
7. **Simple Template to add functionality on DSP:** SoC developers can explore maximum flexibility by using C6Accel algorithm as a template to add custom compute intense functionality on the DSP that can be accessed from the ARM.

In addition to the components used for the Linux app, these are used (and the rest is greyed out in the diagram above):

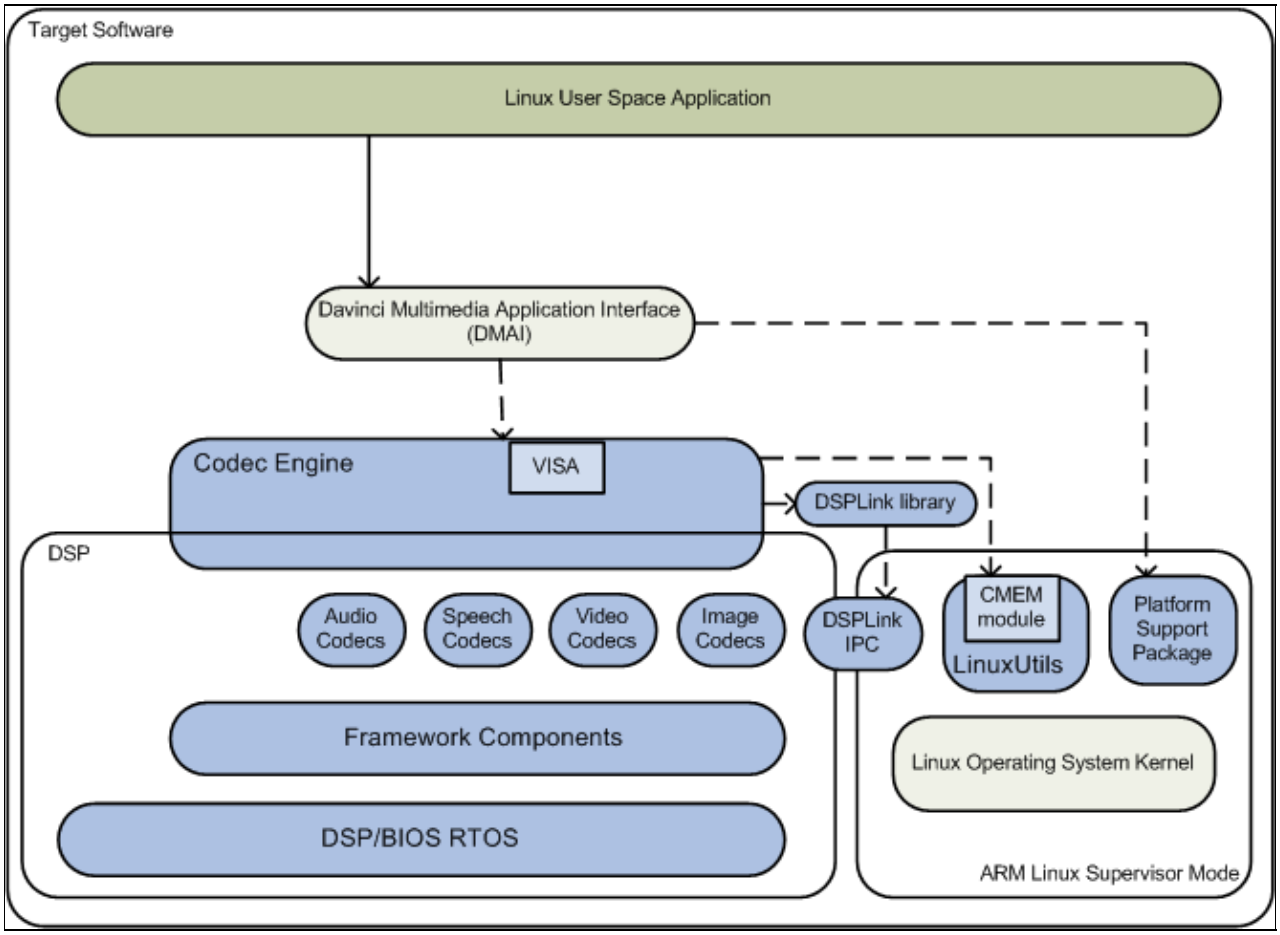
Component	Purpose in this application	Location in the DVSDK
Codec Engine	Cross platform framework for the applications invoking multimedia codecs and other algorithms.	codec_engine_xx_xx_xx_xx
LinuxUtils	Linux specific utilities for Framework Components assisting with resource allocation of DMA channels (EDMA module), physically contiguous memory (CMEM module, see this wiki topic for more information) and allows the codecs to receive completion interrupts of various coprocessor resources (IRQ module).	linuxutils_xx_xx_xx_xx
RTSC (XDC)	Tool used to configure Codec Engine, Framework Components and multimedia codecs for your application.	xdctools_xx_xx_xx_xx
XDAIS	TI Algorithm Interface Standard used for algorithm standardization which is used by various other components including Codec Engine	xdais_x_xx_xx_xx
DSPLINK	GPP to DSP processor communication link for passing messages and data in multiprocessor systems	dsplink_x_xx_xx_xx

Good application examples to start from include:

- The C6Accel contains a sample application to test/validate the functionality. The application is located in the c6accel_xx_xx_xx_xx/soc/app directory.

For more information on C6Accel visit C6Accel: ARM access to DSP software

Creating a DMAI multimedia application



Overview of a DMAI application component usage

The Davinci Multimedia Application Interface (DMAI) is a thin utility layer on top of Codec Engine and the Linux kernel. The benefits of using DMAI include:

1. DMAI and it's sample applications are written to adhere to the XDM 1.x semantics for multimedia codecs. Codec Engine facilitates the invocation of the codecs, but DMAI provides the semantics to make the codecs plug and play.
2. DMAI wraps the Linux device drivers in a multimedia function focused API, shielding you from the rapid progress of the Linux kernel, increasing your portability.
3. The DVSDK demos and gst-ti plugin are written on top of DMAI. If you can make a codec work with the DMAI sample applications, it will most likely work in these applications too.

In addition to the components used for the Linux app, these are used (and the rest is greyed out in the diagram above):

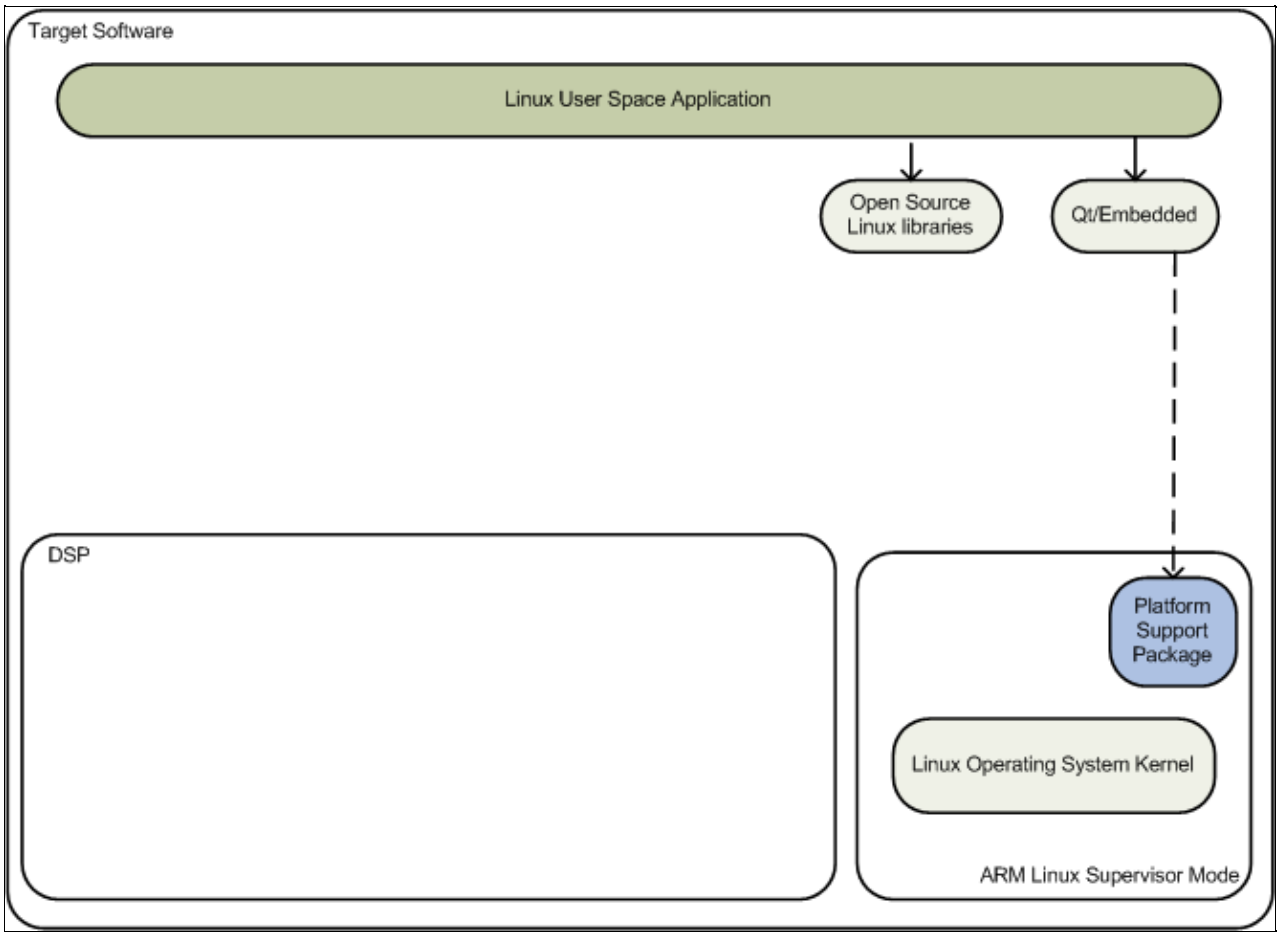
Component	Purpose in this application	Location in the DVSDK
Codec Engine	Cross platform framework for the applications invoking multimedia codecs and other algorithms.	codec_engine_xx_xx_xx_xx

Framework Components	Cross platform framework for servicing resources to algorithms.	framework_components_xx_xx_xx_xx
LinuxUtils	Linux specific utilities for Framework Components assisting with resource allocation of DMA channels (EDMA module), physically contiguous memory (CMEM module, see this wiki topic for more information) and allows the codecs to receive completion interrupts of various coprocessor resources (IRQ module).	linuxutils_xx_xx_xx_xx
Davinci Multimedia Application Interface	Multimedia application utility layer	dmai_xx_xx_xx_xx
Multimedia Codecs	Compression and decompression of multimedia data	codecs_<platform>_xx_xx_xx_xx
RTSC (XDC)	Tool used to configure Codec Engine, Framework Components and multimedia codecs for your application.	xdctools_xx_xx_xx_xx

Good application examples to start from include:

- The DMAI sample applications (dmai_xx_xx_xx_xx/packages/ti/sdo/dmai/apps) provide simpler and smaller examples on how to use DMAI to create a multimedia application.
- If applicable for your device, the DVSDK demos located in the dvsdk_demos_xx_xx_xx_xx of the $(DVSDK) installation directory. These use DMAI to provide a full multimedia application. However, the application does not support A/V sync; if this feature is required GStreamer is a better option.

Creating a Qt/Embedded application



Overview of a Qt/Embedded application component usage

Qt/Embedded is a Graphical User Interface toolkit for rendering graphics to the Linux framebuffer device, and is included in this kit. The base Qt toolkit on the other hand renders the graphics to the X11 graphical user interface instead of to the basic framebuffer.

In addition to the components used for the basic Linux app, these are used (and the rest is greyed out in the diagram above):

Component	Purpose in this application	Location in the DVSDK
Qt/Embedded	Provides a Graphical User Interface toolkit	linux-devkit/arm-arago-linux-gnueabi/usr/lib/libQt*

See the Qt Reference Documentation on various API's and its usages. You can also download some Qt/e example applications from Qt Examples web page.

Compiling an application

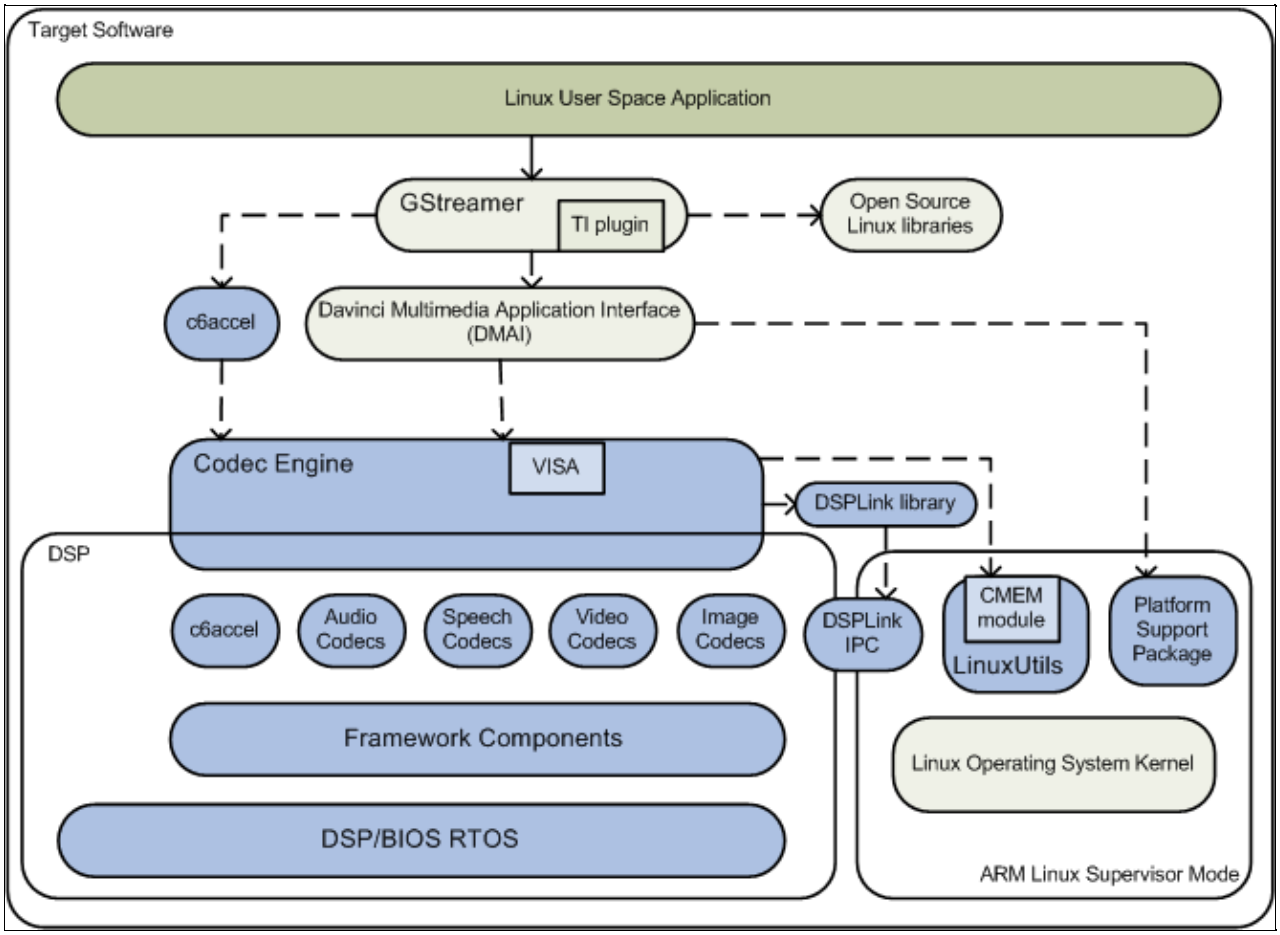
DVSDK Linux development kit includes the Qt/Emebedded host tools and development header and libraries.

1. First, configure your cross compilation environment #Setting_up_cross_compilation_environment.

2. Next, follow the typical Qt/e recommended method for cross compiling your application on host.

```
host $ cd <directory where your application is>
host $ qmake -project
host $ qmake
host $ make
```

Creating a GStreamer application



Overview of a GStreamer application component usage

GStreamer is an open source multimedia framework which allows you to construct pipelines of connected plugins to process multimedia content. There is a plugin which accelerates multimedia using DMAI and Codec Engine.

Compared to creating an application directly on top of DMAI you get the advantage of A/V sync and access to many useful open source plugins which e.g. allows you to demux avi-files or mp4-files. The downside is increased complexity and overhead.

In addition to the components used for the DMAI app, these are used (and the rest is greyed out in the diagram above):

Component	Purpose in this application	Location in the DVSDK
GStreamer	Multimedia Framework	linux-devkit/arm-arago-linux-gnueabi/usr/lib

To construct your own pipelines there are examples of how to use the open source plugins in various places

on the web including the GStreamer homepage. And to learn more about GStreamer-ti plugin architecture watch this online video and visit gstreamer.ti.com.

See the GStreamer Application Development Manual and the GStreamer 0.10 Core Reference Manual on how to write GStreamer applications.

Compiling an application

DVSDK Linux development kit includes the GStreamer development header files, libraries and package configs.

1. First, configure your cross compilation environment #Setting_up_cross_compilation_environment

2. Next, follow the typical GStreamer recommended method for compiling your application. e.g.

```
host $ cd <directory where your application is>
host $ gcc -o decode decode.c `pkg-config --libs --cflags gstreamer-0.10`
```

Additional Procedures

Setting up cross compilation environment

To enable your application development, DVSDK comes with linux-devkit which contains package header, libraries and other package dependent information needed during development. Execute the following commands to configure your cross compilation environment

```
host $ source ${DVSDK}/linux-devkit/environment-setup
```

The above command will export cross compilation specific environment variables.

You will notice that the command will add **[linux-devkit]** to your bash prompt to indicate that you have exported the required cross compiler variables.

Rebuilding the DVSDK components

The DVSDK has provided a top level Makefile to allow the re-building of the various components within the DVSDK.

Note: The DVSDK component build environment is self contained and doesn't require the #Setting_up_cross_compilation_environment thus should be avoided to prevent possible build failures.

Rebuild the DVSDK components by first entering the DVSDK directory using:

```
host $ cd ${DVSDK}
```

The DVSDK makefile has a number of build targets which allows you to rebuild the DVSDK components. For a complete list execute:

```
host $ make help
```

Some of the components delivered in the DVSDK are not pre-built. The provided 'make clean' & 'make components' build targets are designed to clean and build all components (e.g. Linux Kernel, U-boot, CMEM, DMAI, etc.) for which a build is compulsory to begin application development. These components must first be cleaned and then rebuilt by the user before the user attempts to rebuild anything else. To do this, simply run

```
host $ make clean
host $ make components
```

After that, each of the build targets listed by 'make help' can then be executed using:

```
host $ make <target>_clean
host $ make <target>
host $ sudo make <target>_install
```

In order to install the resulting binaries on your target, execute one of the "install" targets using "sudo" privileges. Where the binaries are copied is controlled by the `EXEC_DIR` variable in `${DVSDK}/Rules.make`. This variable is set up to point to your NFS mounted target file system when you executed the DVSDK setup (`setup.sh`) script, but can be manually changed to fit your needs.

You can remove all components (including demos and examples) generated files at any time using:

```
host $ make clean
```

And you can rebuild all components and demos/examples using:

```
host $ make all
```

You can then install all the resulting target files using:

```
host $ sudo make install
```

Note: By default make install will override existing files and this can be controlled via modifying EXEC_DIR variable in Rules.make file.

Note: Booting newly build kernel requires you to run "depmod -a" command on target to regenerate new module dependencies for modprobe to work properly.

Creating your own Linux kernel image

The pre-built Linux kernel image (uImage) provided with the DVSDK is compiled with a default configuration. You may want to change this configuration for your application, or even alter the kernel source itself. This section shows you how to recompile the Linux kernel provided with the DVSDK, and shows you how to boot it instead of the default Linux kernel image.

1. If you haven't already done so, follow the instructions in #Setting_up_the_DVSDK to setup your build environment.

2. Recompile the kernel provided with the DVSDK by executing the following:

```
host $ cd ${DVSDK}
host $ make linux_clean
host $ make linux
host $ sudo make linux_install
```

3. You will need a way for the boot loader (u-boot) to be able to reach your new uImage. TFTP server has been setup in the #Setting_up_the_DVSDK section.

4. Copy your new uImage from the EXEC_DIR specified in the file ${DVSDK}/Rules.make to the tftpserver:

```
host $ cp ${HOME}/targetfs/boot/uImage /tftpboot/new_uImage
```

5. Run the u-boot script and follow the instructions. Select TFTP as your Linux kernel location and the file 'new_uImage' as your kernel image.

```
host $ ${DVSDK}/bin/setup-uboot-env.sh
```

6. Note that when you change your kernel, it is important to rebuild all the kernel modules supplied by the DVSDK sub-components. You can find a list of these modules under the directory /lib/modules/<kernel-version>/kernel/drivers/dsp/ (replace <*kernel-version*> with the version of the kernel applicable to your platform)

```
host $ ls ${HOME}/targetfs/lib/modules/<kernel-version>/kernel/drivers/dsp/
```

For each module that you see listed, you should go back to the host, rebuild it, and replace the file with the one from your EXEC_DIR. E.g. for cmemk.ko

```
host $ cd ${DVSDK}
host $ make cmem_clean
host $ make cmem
host $ sudo make cmem_install
```

You can also opt to re-build all the TI provided kernel modules (CMEM, etc.) and examples applications including the Linux kernel modules by issuing a **make all**. Then running the **make install** as described in the #Rebuilding_the_DVSDK_components.

8. After updating all modules, start minicom or Tera Term and power-cycle the board. The new kernel will now be loaded over TFTP from your Linux host.

9. Re-generate the kernel module dependency file on target by running the below command.

```
target $ depmod -a
```

Setting up Tera Term

Tera Term is a commonly used terminal program on Windows. If you prefer to use it instead of Minicom, you can follow these steps to set it up.

1. Download Tera Term from this location, and start the application.

2. In the menu select *Setup->General...* and set:

```
Default port: COM1
```

3. In the menu select *Setup->Serial Port...* and set the following:

```
Port:         COM1
Baud rate:    115200
Data:         8 bits
Parity:       none
Stop:         1 bit
Flow control: none
```

Flashing boot loader using serial flash utility

Follow the below instructions to flash ubl binaries on the OMAP-L138 EVM from a Windows PC.

1. Power OFF the OMAP-L138 EVM.

2. Connect the RS232 serial cable from the host Windows PC to the OMAP-L138 EVM.

3. Set the boot pins to UART2 boot mode. This is done by setting switch S7 on the OMAP-L138 EVM according to the following table:

Pin#	1	2	3	4	5	6	7	8
Position	OFF	OFF	OFF	OFF	OFF	OFF	ON	ON

4. Copy these files from DVSDK installation directory to your host Windows PC directory of choice:

- ${DVSDK}/psp/board-utilities/serialflasher/sfh_OMAP-L138.exe
- ${DVSDK}/psp/board-utilities/images/boot-strap/arm-mmcsd-ais.bin

Optional

- ${DVSDK}/psp/prebuilt-images/u-boot-da850-omapl138-evm.bin

NOTE: The serial flash utility must be run from the command line under Windows with the Microsoft .NET Framework 4 installed. You can download it from here

5. Open a command prompt on the host Windows PC and run the following flashing utility that was copied, as follows:

```
sfh_OMAP-L138.exe -flash_noubl -p COM1 arm-mmcsd-ais.bin
```

NOTE: If you want to flash both ubl and u-boot in SPI flash then use: **sfh_OMAP-L138.exe -flash -p COM1 arm-mmcsd-ais.bin u-boot-da850-omapl138-evm.bin**

6. When asked, power ON the OMAP-L138 EVM.

7. After the UBL file have been flashed, power OFF the EVM. Restore S7 back to normal (SPI boot) mode and power ON the EVM. This is done by setting switch S7 on the OMAP-L138 EVM according to the following table:

Pin#	1	2	3	4	5	6	7	8
Position	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF

How to copy the kernel image to SPI Flash

The documented process uses TFTP to load the kernel image onto SDRAM and save to SPI flash. The chart shows the SPI flash partitions

Address Range	Size	Usage
0x000000 to 0x03FFFF	256K	uboot
0x040000 to 0x07FFFF	256K	uboot environment variables
0x080000 to 0xA7FFFF	2.5M	kernel
0xA80000 to 0x7FFFFF	5M	

The steps assumes that you have already flashed bootloaders (uboot,ubl) as described in #Flashing boot loader using serial flash utility, and that you have run the *setup.sh'* script to boot the kernel from TFTP and filesystem from NFS.

1. Start minicom or Tera Term and power ON the board. Press any key to stop the boot process.

2. At the u-boot prompt execute the following commands:

```
u-boot :> dhcp
u-boot :> tftpboot 0xC0700000 ${bootfile}
u-boot :> sf probe 0
u-boot :> sf erase 0x080000 0x280000
u-boot :> sf write 0xC0700000 0x080000 0x280000
```

3. Re-run the u-boot script and follow the instructions, but this time select that your Linux kernel is in flash.

```
host $ ${DVSDK}/bin/setup-uboot-env.sh
```

Note! If you have built your own Linux kernel image as described in #Creating your own Linux kernel image, copy the resulting image to the tftp root directory and run the setup-uboot-env.sh (as indicated above):

```
host $ cp ${HOME}/targetfs/home/root/omapl138/boot/uImage /tftpboot/uImage-mynewimage
```

When you run the setup script the first time as per the instructions above, make sure you select the new uImage file instead of the default one.

Integrating a new Codec in the OMAPL138 DVSDK

There are codecs available on the C64x+ Codec Page which are not included in the DVSDK because they require additional licenses. In order to use these codecs in the DVSDK environment they must be integrated in the OMAP-L138 Codec Server. The following steps describe how to integrate a new codec into the OMAP-L138 Codec Server package. The MP3 Decoder is used as an example but the information applies to any codec.

NOTE: That xx_xx_xx_xx should corresponded to the specific version associated with your DVSDK installation.

1. Download the MP3 Decoder package from the Linux Download section located on the C64x+ Audio Codecs Page and install it.

```
host $ ./c64xplus_mp3dec_x_xx_xxx_production.bin
```

2. Extract the tar file from the directory the MP3 codec was installed.

```
host $ tar -xf c64xplus_mp3dec_x_xx_xxx_production.tar
```

3. Make a copy of the OMAP-L138 codec server folder `$(DVSDK)/codecs-omapl138_x_xx_xx_xx` and rename it:

```
host $ cd $(DVSDK)
host $ cp -r codecs-omapl138_xx_xx_xx_xx codecs-omapl138_x_xx_xx_xx_test
```

4. Update in top level Rules.make file to point to the directory of the newly copied codec folder:

```
host $ gedit Rules.make
```

5. Modify to following lines in the Rules.make as shown in *italics* below:

```
# Where the codecs are installed.
# CODEC_INSTALL_DIR=$(SDK_INSTALL_DIR)/codecs-omapl138_x_xx_xx_xx
CODEC_INSTALL_DIR=$(SDK_INSTALL_DIR)/codecs-omapl138_x_xx_xx_xx_test
```

6. Save and exit editor.

7. Copy the folder mp3dec directory from the MP3 codec directory to the DVSDK codecs folder previously created:

```
host $ cp -r \
c64xplus_mp3dec_x_xx_xxx_production/packages/ti/sdo/codecs/mp3dec \
$(DVSDK)/codecs-omapl138_x_xx_xx_xx_test/packages/ti/sdo/codecs/.
```

8. A few files need to be update that were copied from the MP3 codec package so that a new codec server can be built for the C674 target:

- Replace the content of $(DVSDK)/codecs-omapl138_x_xx_xx_xx_test/packages/ti/sdo/codecs/mp3dec/package.xs with the following

```
/*
 *  ======== package.xs ========
 *
 */

/*
 *  ======== getLibs ========
 */
function getLibs(prog)
{
    var lib = null;

    if (prog.build.target.isa == "64P"||prog.build.target.isa == "674") {
        if ( this.MP3DEC.watermark == false ) {
                lib = "lib/mp3dec_tii_l1l2l3.l64P";
        }
        else {
                lib = null;
        }
        print("    will link with " + this.$name + ":" + lib);
    }
    return (lib);
}

/*
 *  ======== getSects ========
 */
function getSects()
{
    var template = null;

    if (Program.build.target.isa == "64P"||Program.build.target.isa == "674") {
        template = "ti/sdo/codecs/mp3dec/link.xdt";
    }

    return (template);
}
```

- Replace the content of $(DVSDK)/codecs-omapl138_x_xx_xx_xx_test/packages/ti/sdo/codecs/mp3dec/**ce/**package.xs with the following

```
/*
 *  ======== package.xs ========
 *
 */


function getLibs(prog)
{
    var lib = null;

    if (prog.build.target.isa == "64P"||prog.build.target.isa == "674") {

        lib = "lib/mp3auddecskel.a" + prog.build.target.isa;

        print("    will link with " + this.$name + ":" + lib);

    }
```

```
    return (lib);
}
```

- Replace the content of $(DVSDK)/codecs-omapl138_x_xx_xx_xx_test/packages/ti/sdo/codecs/mp3dec/**ce/**package.bld with the following

```
/*
 *  ======== package.bld ========
 */


Pkg.attrs.exportAll = true;
var SRCS = ["src/auddec1_skel.c", ];

for (var i = 0; i < Build.targets.length; i++) {
    var targ = Build.targets[i];

    if (targ.name == "C64P"||targ.name == "C674") {
        Pkg.addLibrary("lib/mp3auddecskel", targ, {
        }).addObjects(SRCS);
    }
}
```

9. Build the codecs package by running "make codecs_clean" followed by "make codecs" in the $(DVSDK) folder. All the codec packages must be built in order for the Codec Engine GenServer Wizard to recognize them. If "make components" hasn't been previously executed as described in the #Rebuilding_the_DVSDK_components section, do so before running the next commands.

```
host $ cd $(DVSDK)
host $ make codecs_clean
host $ make codecs
```

10. The Codec Engine GenServer Wizard is a tool that is used to generate server packages. For information on GenServer Wizard go to The Codec Engine GenServer Wizard FAQ

To Launch the server wizard:

```
host $ cd $(DVSDK)/codecs_omapl138_x_xx_xx_xx_test/
host $ make -f Makefile.ce.genserver
```

11. From the wizard GUI open the file (File -> Open) `codecs-omapl138_x_xx_xx_xx_test/ti_sdo_server_cs_wizard.svrwiz`. This file includes the configuration of the original OMAP-L138 codec server.

12. Set the Server Package Name: **ti.sdo.server.cs**

13. Set the Destination Directory: **$(DVSDK)/codecs-omapl138_x_xx_xx_xx_test/packages**

14. Set C6000 TI cgtools Directory to: **$(DVSDK)/cgt6x_x_x_xx**

15. The Search Path must be modified to pick up the correct codec packages and dependent components. Click on **Set Search Path** button and remove all the existing paths. Now add the following:

- $(DVSDK)/c6accel_x_xx_xx_xx/soc/packages
- $(DVSDK)/codecs_omapl138_x_xx_xx_xx_test/packages
- $(DVSDK)/codec-engine_x_xx_xx_xx/packages
- $(DVSDK)/xdais_x_xx_xx_xx/packages
- $(DVSDK)/framework-components_x_xx_xx_xx/packages

16. Refresh Codec list, MP3DEC should now be in the codec list. If MP3DEC is not in the list make sure that the Search Path to the codec packages has been updated properly in the previous step. Select all the codecs

17. Uncheck "Generate CCS Eclipse project" and Check "Don't check for building dependencies .."

18. Select Next and then Finish.

19. Select "Yes" to save the values entered in a new configuration file. Set the file name to:

```
ti_sdo_server_cs_wizard_test.svrwiz
```

Set the Save folder in to: $(DVSDK)/codecs_omapl138_x_xx_xx_xx_test

20. Replace to following files from `$(DVSDK)/codecs_omapl138_x_xx_xx_xx_test/packages/ti/sdo/server/cs/package.bld, codec.cfg, server.cfg, server.tcf` by copying them from the original Codec Server directory `$(DVSDK)/codecs_omapl138_x_xx_xx_xx/packages/ti/sdo/server/cs`

21. Add the following to the codec.cfg file copied above:

```
var MP3DEC = xdc.useModule('ti.sdo.codecs.mp3dec.ce.MP3DEC');

    MP3DEC.serverFxns = "MP3DEC_INBUFCACHEFLUSH";
    MP3DEC.stubFxns = "AUDDEC1_STUBS";
    MP3DEC.alg.watermark = false;
    MP3DEC.alg.codeSection = codeSection;
    MP3DEC.alg.udataSection = udataSection;
    MP3DEC.alg.dataSection = dataSection;
```

Add the following to codec.cfg in the Server.algs array:

```
{name: "mp3dec", mod: MP3DEC , threadAttrs: {
    stackMemId: 0, priority: Server.MINPRI + 3},
    groupId : 2,
},
```

22. Remove config.bld from the `server/cs` directory

23. Build the codec server package by running "make codecs" from the top on the $(DVSDK) directory.

```
host $ cd $(DVSDK)
host $ make codecs
```

24. Test the new server with one of the DMAI sample apps.

How to create an SD card

This section describes how to create an SD card for the filesystem and has been tested with a 4GB SD card. It is useful if you have downloaded the DVSDK from the web and want to recreate the SD card image provided in the box, or if your existing SD card has been corrupted.

1. Make sure you have flashed the UBL bootloaders as described in #Flashing boot loader using serial flash utility.

2. Plug an SD card on Linux host machine.

3. Run dmesg command to check the device node. Triple check this to ensure you do not damage your HDD contents!

```
host $ dmesg
   [14365.272631] sd 6:0:0:1: [sdc] 3862528 512-byte logical blocks: (1.97 GB/1.84 GiB)
   [14365.310602] sd 6:0:0:1: [sdc] Assuming drive cache: write through
   [14365.325542] sd 6:0:0:1: [sdc] Assuming drive cache: write through
   [14365.325571]  sdc: sdc1 sdc2
```

In this example, SD card is detected on /dev/sdc.

4. Run mksdboot script installed in DVSDK as show below using the correct device detected for the SD card above

```
host $ sudo ${DVSDK}/bin/mksdboot.sh --device /dev/sdc --sdk ${DVSDK}
```

Wait for script to complete. On successful completion, remove the SD card from the host PC.

NOTE: When creating a bootable SD card, the mksdboot script uses the pre-built root filesystem, kernel and bootloader images provided in the DVSDK installation

5. The DVSDK comes with a script for setting up u-boot to boot the filesystem and the Linux kernel from SD card. Enter the DVSDK directory and execute:

```
host $ ${DVSDK}/bin/setup-uboot-env.sh
```

Follow the instructions (accept the defaults for host ip address and filesystem directory) and choose SD card for both Linux kernel and filesystem location. Make sure to plug the RS-232 serial cable from your Linux host PC to the OMAP-L138 board so that the script properly sets up the u-boot variables.

6. Insert the SD card into the OMAP-L138 EVM. Power cycle the board, it will boot from SD card.

Note! If you want to recreate the full SD card with the DVSDK installer execute the following:

```
host $ sudo ${DVSDK}/bin/mksdboot.sh --device /dev/sdc --sdk ${DVSDK} \
/path/to/dvsdk_omapl138-evm_4_xx_xx_xx_xx_setuplinux
```

This takes significant extra time so it's not part of the default instructions.

Setting up Bluetooth and Wireless LAN demo

For more information on how to enable WL1271 daughtercard and run Wireless LAN and Bluetooth demo see AM18x_Wireless_Connectivity_Demo.

And refer release note for additional information.

GPLv3 Disclaimer GPLv3 Disclaimer GPLv3 Disclaimer GPLv3 Disclaimer

There are GPLv3 licensed software components contained within the this SDK on host side. The software manifest (software_manifest.htm) is located in the docs/ directory of the installed SDK. All GPLv3 components are contained in the SDK directory.

These GPLv3 components are provided for development purposes only and are intended to be removed before installing the application(s) code in your final product.

Additional Resources

PSP Documentation

http://processors.wiki.ti.com/index.php/DaVinci_PSP_Releases

http://processors.wiki.ti.com/index.php/DaVinci_PSP_03.20.00.14_Release_Notes

http://processors.wiki.ti.com/index.php/DaVinci_PSP_03.20.00.14_Device_Driver_Features_and_Performance_Guide

http://processors.wiki.ti.com/index.php/Community_Linux_PSP_for_DA8x/OMAP-L1/AM1x

Wireless LAN, Bluetooth and Crypto

Note: To run WiFi demos from Matrix-GUI click on right arrow at the top of the GUI.

Wireless connectivity hardware installation:

http://processors.wiki.ti.com/index.php/AM18x_Wireless_Connectivity_Hardware_installation_guide

WLAN station mode demo: This include WLAN ping test demo from EVM & web browsing from EVM GUI

http://processors.wiki.ti.com/index.php/Open_Source_Wireless_Connectivity_Guide#WLAN_Station

WLAN SoftAP mode demo:

http://processors.wiki.ti.com/index.php/Open_Source_Wireless_Connectivity_Guide#WLAN_SoftAP

More details on WLAN & Bluetooth :

http://processors.wiki.ti.com/index.php/ARM_Processor_Open_Source_Wireless_Connectivity

Cryptography Users Guide:

http://processors.wiki.ti.com/index.php/Cryptography_Users_Guide

Matrix Application launcher

http://processors.wiki.ti.com/index.php/Matrix_Users_Guide

Miscellaneous

http://processors.wiki.ti.com/index.php/DVSDK_4.x_FAQ

http://processors.wiki.ti.com/index.php/Category:OMAPL1