TMS320DM368 Software Developers Guide

Translate this page to Translate Show original

Contents

Welcome to the TMS320DM368 Software Developer's Guide

Thanks you for choosing the TMS320DM368 Evaluation Module (EVM) for your application. The purpose of this guide is to get you going with developing software for the TMS320DM368 on a Linux development host only.

Note! This Software Developer's Guide (SDG) supports version 4.xx of the TMS320DM368 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_dm368-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
```

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/dm368
host $ cp helloworld ${HOME}/targetfs/home/root/dm368
```

6. On your target this application will be accessible from `/home/root/dm368/helloworld`. Execute it on your target:

```
target # /home/root/dm368/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 DVSDK demos from the command line

The DVSDK multimedia demos can be launched from the Matrix application launcher. There is a copy installed in your NFS mounted target filesystem as well. First enter the demo directory on your target:

```
target # cd /usr/share/ti/dvsdk-demos/
```

Then, load the kernel modules:

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

You can quickly view information on options available to the demos using the -h option. For example:

```
target # ./encode -h
target # ./decode -h
```

Alternatively you can refer to even more detail in the documentation in these files:

```
target # cat encode.txt
target # cat decode.txt
target # cat encodedecode.txt
```

After reading up on the options, you can execute the demos from the command line like this:

```
target # ./encode <options>
target # ./decode <options>
target # ./encodedecode <options>
```

Note that there are multimedia clips installed under `/usr/share/ti/data` on the target.

If you prefer to run the demos using a graphical user interface (instead of the textual one), you can pass in the -o option to do so.

When using the 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=
```

NOTE: ***To capture from component/composite input, the kernel bootargs must be configured to enable these inputs. Bootargs can be generated by running the setup script. See the section #Setting_up_the_DVSDK for details on running the setup script.***

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/
```

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.

Video

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

```
target # ./video_decode_io2_dm368.x470MV -c h264dec -e decode -i /usr/share\
/ti/data/videos/davincieffect.264 -n 30 -o output.yuv
```

To decode 30 frames from an MPEG4 encoded video to a YUV file execute:

```
target # ./video_decode_io2_dm368.x470MV -c mpeg4dec_hdvicp -i /usr/share\
/ti/data/videos/davincieffect.mpeg4 -n 30 -o output.yuv
```

We use the 'mpeg4dec_hdvicp' codec in this case which corresponds to the universal MPEG4 decoder. There is also a codec named 'mpeg4dec' which can be used specifically to decode streams encoded by TI's MPEG4 encoder. It has better performance but it is limited in terms of the streams it can decode.

To encode 25 frames of resolution 1280x720 from a YUV file to an H.264 encoded file execute:

```
target # ./video_encode_io1_dm368.x470MV -c h264enc -i output.yuv -o output.264 \
-n 25 -e encode -r 1280x720
```

To do video loopback from 1080P component input to 1080I component output execute:

```
target # ./video_loopback_dm368.x470MV -O component -I camera -y5
```

Audio

To encode 1000 frames of AAC 48KHz @ 256kbps audio from microphone input execute:

```
target # ./audio_encode1_dm368.x470MV -c aaclc_enc -o output.aac \
--soundinput mic -n 1000 --samplerate 48000 --bitrate 256000
```

To decode 1000 frames of AAC audio to headphone output execute:

```
target # ./audio_decode1_dm368.x470MV -c aac_dec \
-n 1000 -i /usr/share/ti/data/sounds/davincieffect.aac
```

Image

To decode a JPEG encoded file to a YUV file execute:

```
target # ./image_decode_io1_dm368.x470MV -c jpegdec \
-i /usr/share/ti/data/images/remi003_422i.jpg -o output.yuv
```

To encode the resulting YUV file to a JPEG encoded file execute:

```
target # ./image_encode_io1_dm368.x470MV -c jpegenc -e encode \
-i output.yuv -o output.jpeg -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_dm368.x470MV -h
```

Running the Digital Video Test Bench (DVTB)

The DVTB allows you to tweak codec parameters using an interactive text based user interface and observe the result. The DVTB is particularly useful because it allows the user to modify not only codec base parameters but also codec extended parameters.

To run it enter this directory on the target:

```
target # cd /usr/share/ti/dvtb/
```

Depending on your test, you may have different CMEM requirements, the default system wide */etc/init.d/loadmodule-rc* works with the following sanity test:

First launch the dvtb

```
target # ./dvtb-r
```

At the dvtb prompt, enter the following commands to decode the first 30 frames in the davincieffect.264 file and write them to file output.yuv. This example illustrates how to modify codec base parameters.

```
<DVTB> $ setp engine name encodedecode
<DVTB> $ setp viddec2 maxWidth 1280
<DVTB> $ setp viddec2 maxHeight 720
<DVTB> $ setp viddec2 numFrames 30
<DVTB> $ getp viddec2
<DVTB> $ func viddec2 -s /usr/share/ti/data/videos/davincieffect.264 -t output.yuv
```

After completion of the decode process type "enter" to return to the "<DVTB> $" prompt. Then, enter the following command to exit DVTB application

```
<DVTB> $ exit
```

There are scripts provided for some commonly used DVTB tests. These are under scripts folder under /usr/share/ti/dvtb. The scripts which have a codec specific name set up extended parameters specific to that codec. For example scripts/xh264enc1.dvs sets up extended parameters for the h264 encoder codec.

To run the scripts execute

```
target # ./dvtb-r -s scripts/<script-name>.dvs
```

Please modify the scripts as needed with the correct path to the multimedia clip used.

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
```

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

```
target #  cd /usr/bin/qtopia/examples/richtext/calendar
target # ./calendar -qws -geometry 320x200+50+20
```

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

Reload kernel modules and enable video plane before running below gstreamer pipelines:

```
target # /etc/init.d/loadmodule-rc restart
target # cat /dev/zero > /dev/fb2 2> /dev/null
```

The DVSDK comes with GStreamer, and the following pipelines are known to work. You can construct your own pipelines, see Gstreamer pipeline.

Note! Before running these pipelines you need to connect 720P component output display and/or 720P component input video source.

This pipeline decodes MPEG-4 video:

```
target # gst-launch filesrc location=/usr/share/ti/data/videos\
/davincieffect.mpeg4 ! typefind ! TIViddec2 ! queue ! tidisplaysink2 \
video-standard=720p display-output=component -v
```

This pipeline decodes H.264 video:

```
target # gst-launch filesrc location=/usr/share/ti/data/videos/davincieffect.264 \
```

```
! typefind ! TIViddec2 ! queue ! tidisplaysink2 video-standard=720p \
display-output=component -v
```

This pipeline decodes MPEG-2 video:

```
target # gst-launch filesrc location=/usr/share/ti/data/videos/davincieffect.m2v \
! TIViddec2 codecName=mpeg2dec engineName=codecServer ! queue ! tidisplaysink2 \
video-standard=720p display-output=component -v
```

This pipeline decodes a .MP4 file with H264 video and AAC audio content:

```
target # gst-launch filesrc location=/usr/share/ti/data/videos\
/davincieffect_h264_aac.mp4 ! 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 padAllocOutbufs=TRUE ! queue ! tidisplaysink2 \
video-standard=720p display-output=component -v
```

NOTE: ***To run component/composite capture, pipelines bootargs must be configured to enable these inputs. Bootargs can be generated by running the setup script.***

This pipeline encodes H.264@1Mbps video captured from a component 720P input source:

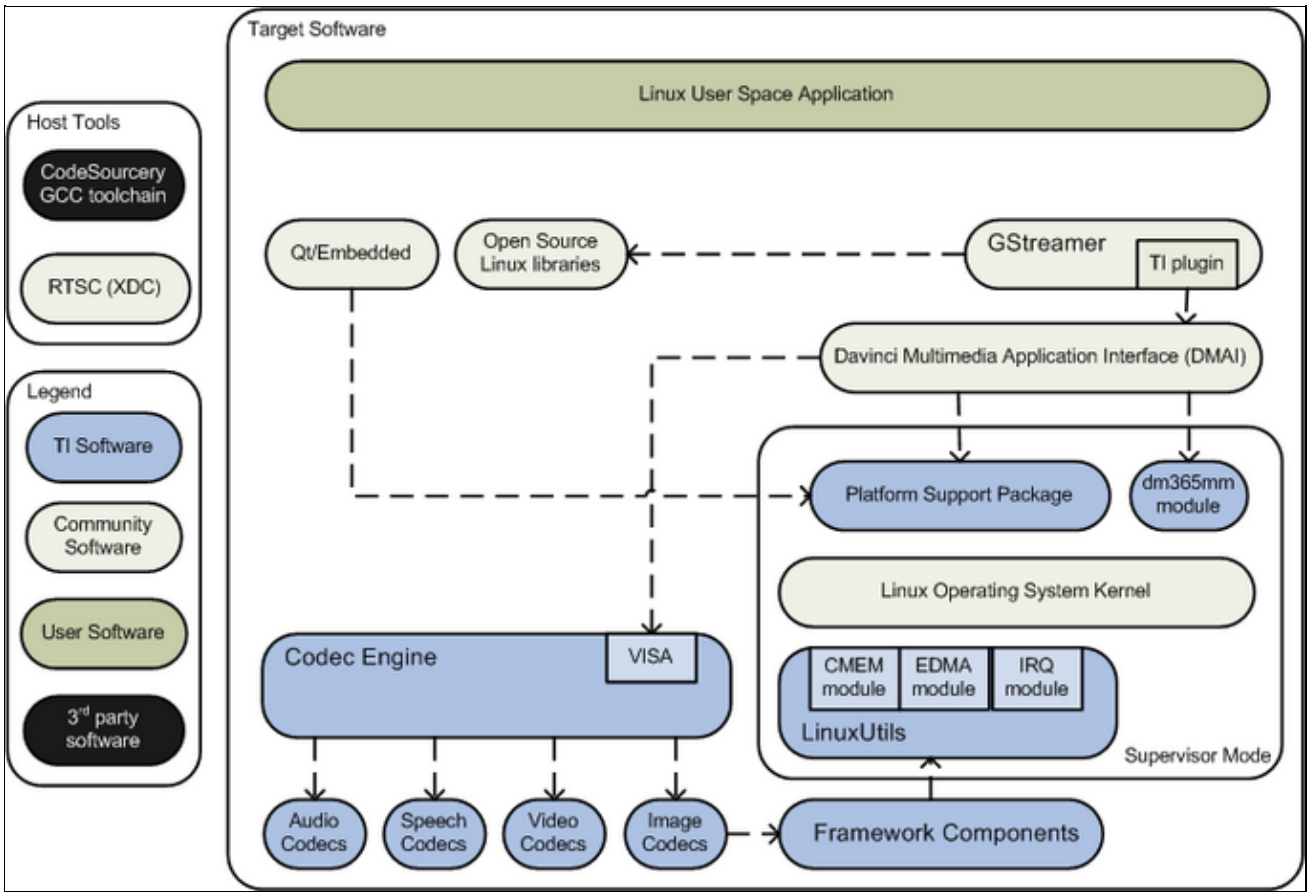
```
target # gst-launch v4l2src always-copy=FALSE num-buffers=500 ! \
'video/x-raw-yuv, format=(fourcc)NV12, framerate=(fraction)30/1, \
width=(int)1280, height=(int)720' ! queue ! TIVidenc1 codecName=h264enc \
engineName=codecServer contiguousInputFrame=TRUE bitRate=1000000 ! queue ! \
filesink location=sample.264 -v
```

This pipeline encodes MPEG-4@9Mbps video captured from component 720P input source:

```
target # gst-launch v4l2src always-copy=FALSE num-buffers=500 ! \
'video/x-raw-yuv, format=(fourcc)NV12, framerate=(fraction)30/1, \
width=(int)1280, height=(int)720' ! queue ! TIVidenc1 codecName=mpeg4enc \
engineName=codecServer contiguousInputFrame=TRUE bitRate=9000000 ! queue ! \
filesink location=sample.m4v -v
```

NOTE: tidisplaysink2 is an experimental sink and intended to replace TIDmaiVideoSink. For more information run gst-inspect tidisplaysink2 on target to see various properties.

DVSDK software overview and support



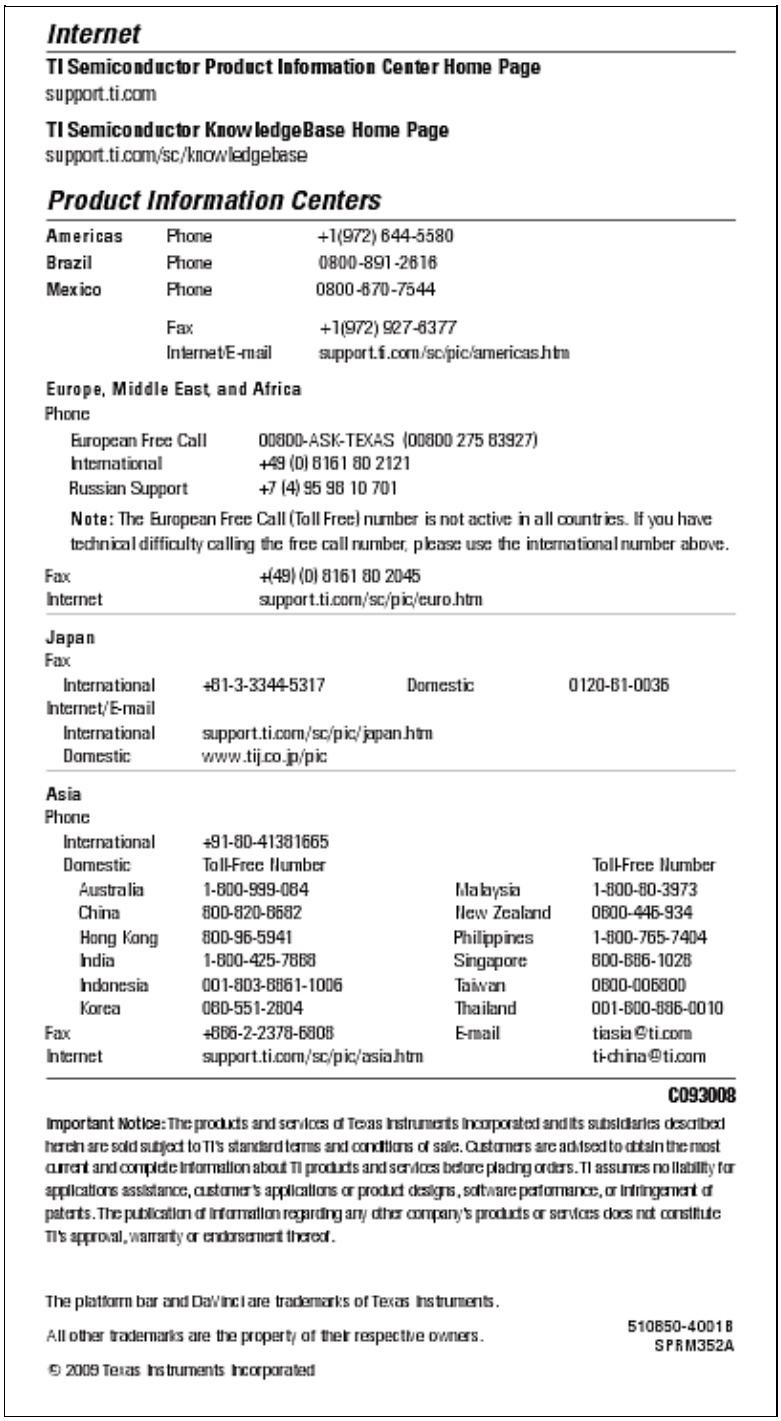
Overview of the DVSDK Software stack

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 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
CodeSourcery GCC toolchain	3rd party software	The 'lite' version of the CodeSourcery GCC toolchain used to develop this software is provided by CodeSourcery without support. However, there are commercially supported versions of the CodeSourcery tools available at their website.

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

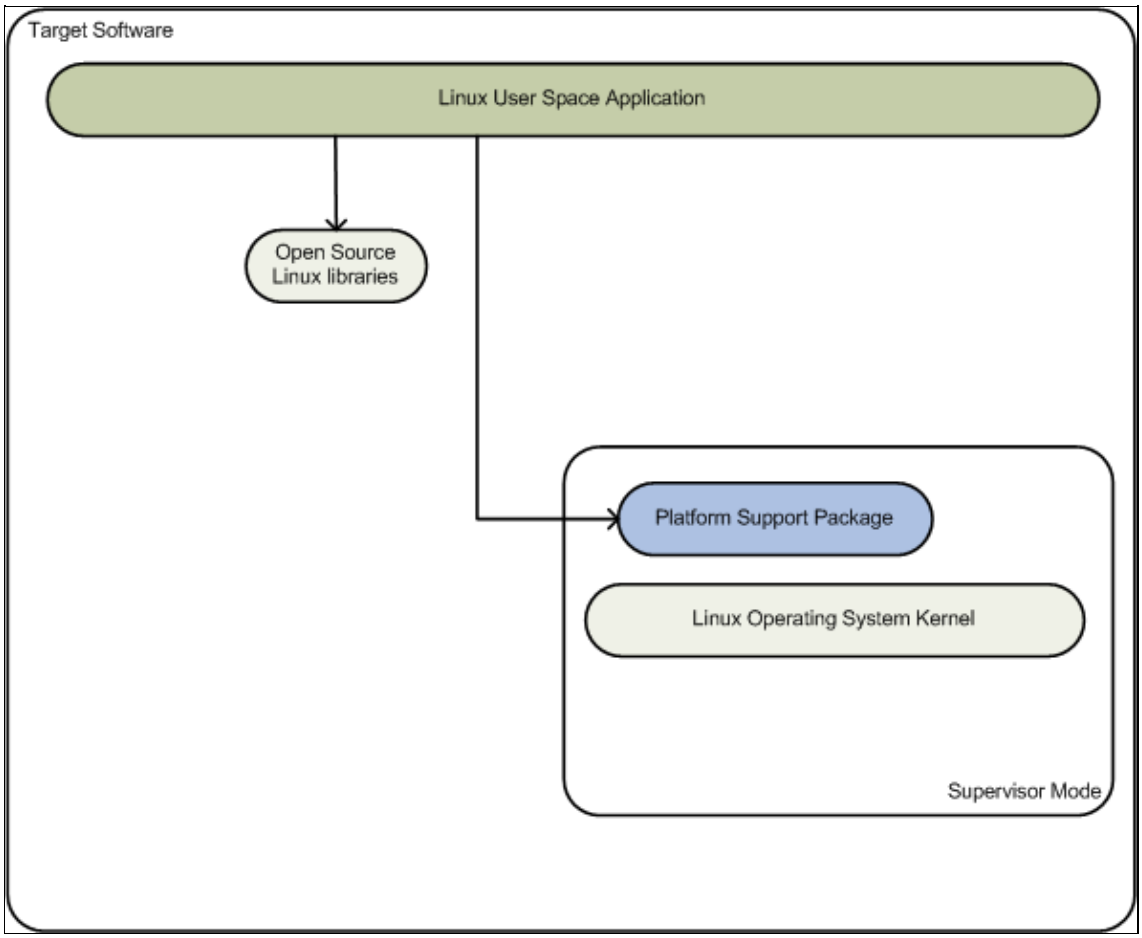
Asia
Phone

International	+91-80-41381665		
Domestic	Toll-Free Number		Toll-Free Number
Australia	1-800-999-064	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



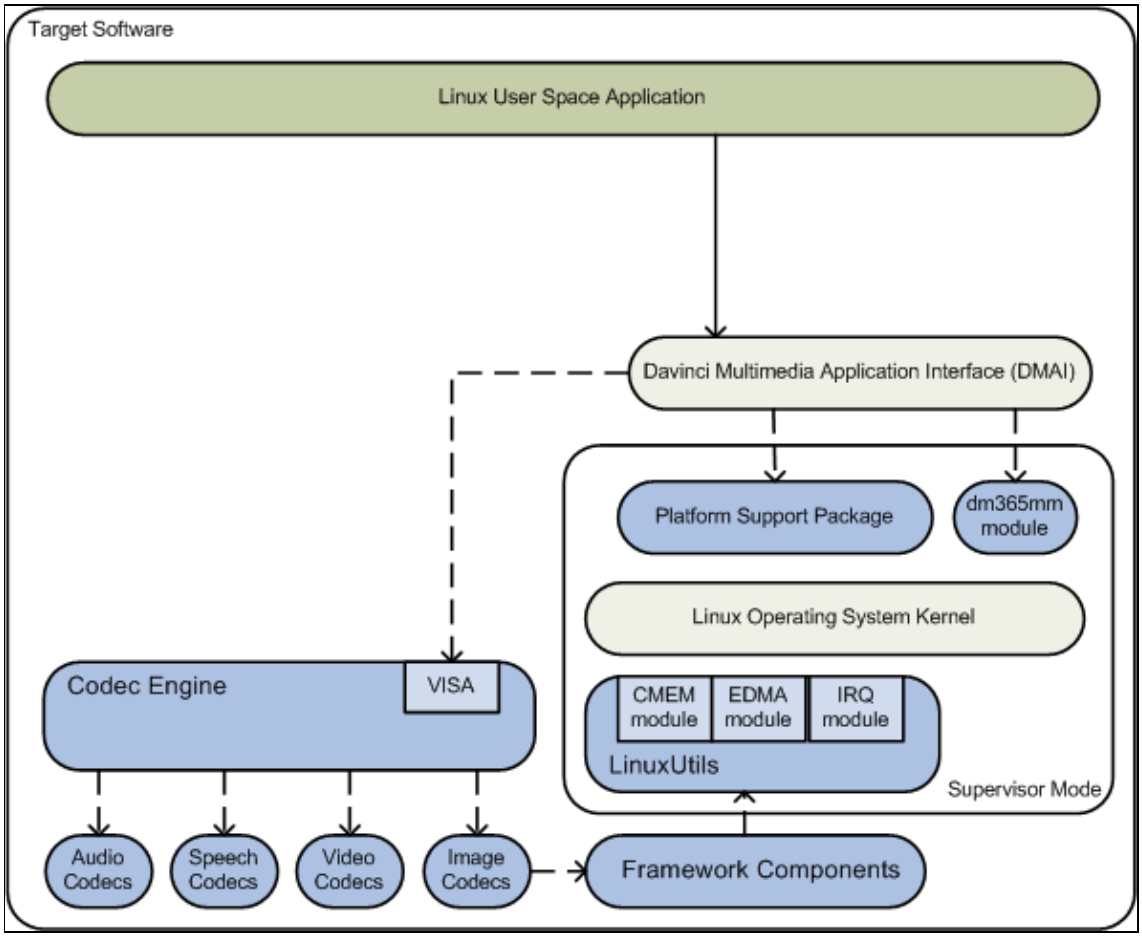
Overview of a basic Linux application component usage

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
CodeSourcery GCC toolchain	Cross compiler for generating ARM Linux binaries.	User specified location outside the DVSDK
Open Source Linux libraries	Provides libraries such as libpng, libusb, libz, libcurl etc.	linux-devkit/arm-none-linux-gnueabi/lib and linux-devkit/arm-none-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 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):

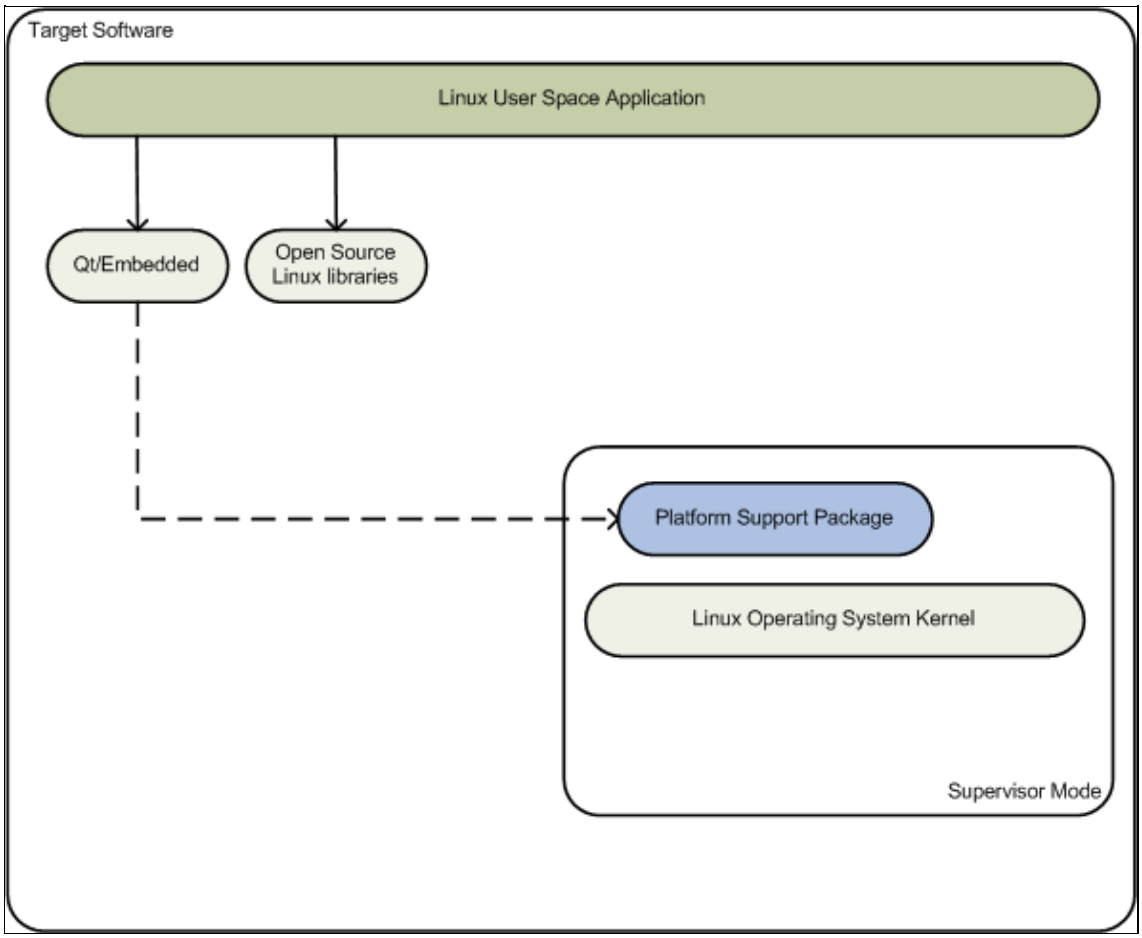
Component	Purpose in this application	Location in the DVSDK
Codec Engine	Cross platform framework for the applications invoking multimedia codecs and other algorithms.	codec_engine_xx_xx_xx_xx

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-none-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

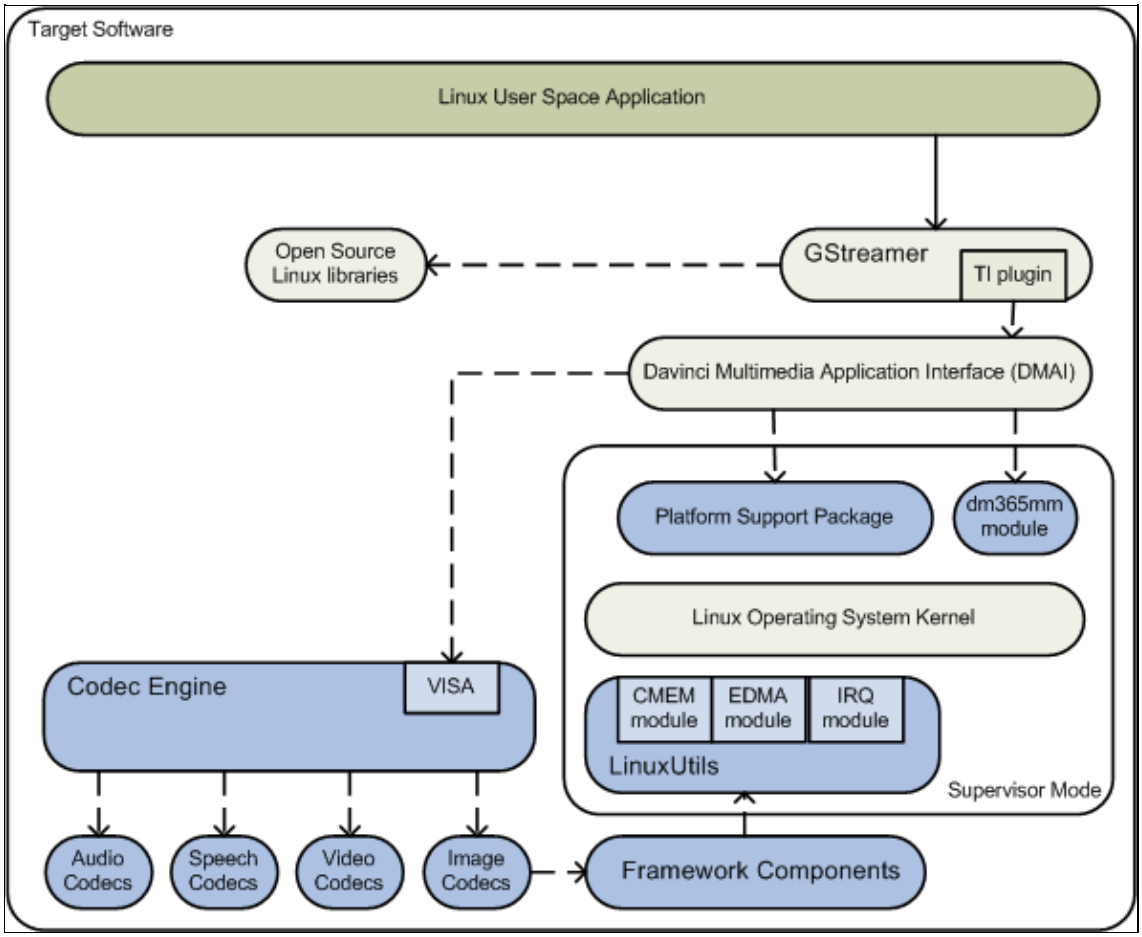
DVSDK Linux development kit includes the Qt/Emebedded host tools and development header and libraries.

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-none-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/2.6.32-rc2-davinci1/kernel/drivers/dsp/ (replace *2.6.32-rc2-davinci1* with the version of the kernel applicable to your platform)

```
host $ ls ${HOME}/targetfs/lib/modules/2.6.32-rc2-davinci1/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

Serial flashing is supported on PG 1.2 silicon or newer, see the "Revision Identification" section in the TMS320DM368 Silicon Errata sprz316 to verify your silicon revision.

Follow the below instructions to flash u-boot and ubl binaries on the DM368 DVEVM from a Window host PC.

1. Power OFF the DM368 EVM.

2. Connect the RS232 cable between your PC and EVM.

3. Write down the configuration of switches SW4 and SW5.

4. Set switch SW5 to Vcore=1.35 Volts configuration as shown below on DM368 EVM:

S5:6 (NTSC/PAL)	S5:5 (VCORE ADJ)	S5:4 (EXTRA3)	S5:3 (EXTRA2)	S5:2 (EXTRA1)	S5:1 (NAND/ONE NAND)
OFF	ON	OFF	OFF	OFF	OFF

5. Set switch SW4 to UART boot mode and 8-bit AEMIF configuration as shown below on the DM368 EVM:

S4:6 (BITSEL2)	S4:5 (BITSEL1)	S4:4 (BITSEL0)	S4:3 (AECFG2)	S4:2 (AECFG1)	S4:1 (AECFG0)
OFF	OFF	OFF	ON	ON	OFF

6. Copy these files from DVSDK installation directory to your host PC :

- psp/board_utilities/serial_flash/dm365/sfh_DM36x.exe
- psp/board_utilities/serial_flash/dm365/UBL_DM36x_NAND_ARM432_DDR340_OSC24.bin
- psp/prebuilt-images/u-boot-dm368-evm.bin

7. Power ON the DM368 EVM.

8. Run the below command on your host PC. Due to an existing bug in the utility, you will set a repetitive BOOTME message at some point. At that point, hit CTRL-C and re-execute the same command. The second try will succeed. Do NOT power off the board between the two attempts.

```
c:\> sfh_DM36x.exe -nandflash UBL_DM36x_SDMMC_ARM432_DDR340_OSC24.bin u-boot-dm368-evm.bin
```

9. Power OFF the DM368 EVM.

10. Restore SW4 and SW5 setting back to normal mode. Or if you want to boot using the newly flashed bootloader, use these dip switch settings:

- Set the Boot Mode / Configuration Select Switch SW4 for the NAND boot mode and 8-bit AEMIF configuration -

S4:6 (BITSEL2)	S4:5 (BITSEL1)	S4:4 (BITSEL0)	S4:3 (AECFG2)	S4:2 (AECFG1)	S4:1 (AECFG0)
OFF	OFF	OFF	OFF	OFF	OFF

- Set the Board Configuration Select Switch SW5 for the Vcore=1.35 Volts configuration -

S5:6 (NTSC/PAL)	S5:5 (VCORE ADJ)	S5:4 (EXTRA3)	S5:3 (EXTRA2)	S5:2 (EXTRA1)	S5:1 (NAND/ONE NAND)
OFF	ON	OFF	OFF	OFF	OFF

Copying Linux Kernel image to NAND flash

If desired, follow the below steps to copy the pre-built Linux Kernel image on NAND flash.

The steps assumes that you have already flashed bootloaders (uboot,ubl) and your TFTP server is up and running.

1. Copy the kernel image to the directory exported by the TFTP server

```
host $ sudo cp ${DVSDK}/psp/prebuilt-images/uImage-dm368-evm.bin /tftpboot/.
```

2. Power ON the board. Hit a key to halt the boot process at the u-boot prompt.

3. Get an ip address from dhcp server using this command:

```
u-boot :> dhcp
```

4. Download kernel image on DM368 using TFTP as shown below. The default kernel image `uImage-dm368-evm.bin` must be available in the directory defined in the TFTP server configuration file **/etc/inetd.conf**.

```
u-boot :> tftp 0x80700000 uImage-dm368-evm.bin
 TFTP from server 192.168.1.25; our IP address is 192.168.1.197
 Filename 'uImage-dm368-evm.bin'.
 Load address: 0x80700000
 Loading: #################################################################
          #################################################################
          ############
 done
 Bytes transferred = 2117624 (204ff8 hex)
```

Make a note of "Byte transferred" hex decimal number.

5. Erase the nand for the required size. The first argument in "nand erase" command is the start offset address and second is the size (you noted in previous step, rounded up to the closest multiple of 0x1000).

```
u-boot :> nand erase 0x400000 0x205000
 NAND erase: device 0 offset 0x400000, size 0x205000
 Erasing at 0x400000 -- 100% complete.
 OK
```

6. Write the data in nand. The first argument in "nand write" command is tftp downloaded address from the step 1 and second argument is NAND offset address and third is the size you erased in above step.

```
u-boot :> nand write 0x80700000 0x400000 0x205000
 NAND write: device 0 offset 0x400000, size 0x205000
  2097152 bytes written: OK
```

7. The DVSDK comes with a script for setting up u-boot to boot the Linux kernel from flash. 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 make sure to choose flash as Linux kernel location.

8. Power cycle the board, it will boot the kernel from flash.

This completes the copying of the Linux Kernel image on NAND flash.

How to flash a filesystem

Assuming you have created a filesystem in a file `filesystem.tar.gz`, follow the below steps to flash the filesystem image on your NAND flash. The step assumes that boot loader is flashed on NAND and NFS based file system is up and running on EVM. See section #Setting up the DVSDK to set up NFS based file system if you haven't already done so. The kernel can be on flash or downloaded through tftpboot. Note that if the flash has been erased, for example during the u-boot flashing process, there is no more kernel on the flash and tftpboot must be used to boot the kernel.

1. Open a terminal emulator window and boot the EVM with default kernel image and NFS root filesystem.

2. On the Ubuntu host, copy filesystem.tar.gz in /home/root directory on your NFS mounted root filesystem (Root permission required).

3. Run the following commands on EVM to copy the default filesystem in NAND flash.

```
target # flash_eraseall /dev/mtd4
 target # mkdir -p /mnt/nand
 target # mount /dev/mtdblock4 /mnt/nand -t jffs2
 target # cd /mnt/nand
 target # tar zxf /home/root/filesystem.tar.gz
 target # sync
 target # cd /home/root
 target # umount /mnt/nand
```

4. The DVSDK comes with a script for setting up u-boot to boot the filesystem from flash. 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 make sure to choose flash as filesystem location.

5. Power cycle the board, it will boot using the filesystem in flash.

Integrating a new Codec in the DM368 DVSDK

There are codecs available on the DM365 Codec Page which are not included in the DVSDK because they require additional licenses. The following steps describe how to integrate a new codec in the DM368 DVSDK. The MP3 Decoder is used as an example but the information applies to any codec.

Replace all version numbers in the procedure as appropriate since the ones shown may be outdated.

1. Download the codec from the DM365 Codec Page. The DM368 can use DM365 codecs given it is simply a DM365 running at a higher clock speed.

2. Run the installer

```
host $ ./dm365_mp3dec_4_8_01_production.bin
```

3. Copy the folder `mp3_dec` to the DVSDK codec folder `${DVSDK}/codecs-dm365_04_00_00_00/packages/ittiam/codecs>`

```
host $ export MP3DIR1=${HOME}/dm365_mp3dec_4_8_01_production

host $ export MP3DIR2=${MP3DIR1}/mp3_dec_4_8_01_production_dm365_mvl

host $ cp -r ${MP3DIR2}/packages-production/ittiam/codecs/mp3_dec
${DVSDK}/codecs-dm365_04_00_00_00/packages/ittiam/codecs
```

4. Test the codec with the audio_decode_io1 application: update the configuration file `audio_decode_io1_dm368.cfg` in ${DVSDK}/dmai_2_20_00_14/packages/ti/sdo/dmai/apps/audio_decode_io1/linux/

```
utils.importFile("../../fc_common.cfg");
utils.importFile("../../app_common.cfg");


/*
 *  ======== Resource Manager Configuration ========
 */

var RMAN = xdc.useModule('ti.sdo.fc.rman.RMAN');

/* No DSKT2 on Linux, so need to use the non-default config */
RMAN.useDSKT2 = false;
RMAN.persistentAllocFxn = "__ALG_allocMemory";
RMAN.persistentFreeFxn = "__ALG_freeMemory";
```

```
/* Implements the ALG_create/delete activate/deactivate interfaces.
   If using another framework (such as Codec Engine), ALG interfaces
   will be provided by it */
xdc.loadPackage("ti.sdo.fc.utils.api");


/*
 *  ======== Engine Configuration ========
 */
var AAC_DEC  = xdc.useModule('ittiam.codecs.aac_dec.ce.AAC_DEC');
var MP3_DEC  = xdc.useModule('ittiam.codecs.mp3_dec.ce.MP3_DEC');

var Engine = xdc.useModule('ti.sdo.ce.Engine');
var myEngine = Engine.create("decode", [
    {name: "aac_dec", mod: AAC_DEC, local: true},
    {name: "mp3_dec", mod: MP3_DEC, local: true},
]);

algSettings = xdc.useModule('ti.sdo.ce.alg.Settings');
algSettings.useCache = true;
```

5. Build the application: `make dmai` (this assumes that `make all` was invoked previously)

6. Export the application: `make dmai_install`

7. Run the application on the target. In the following example we generate an output pcm file that consists of decoding 1000 frames from the mp3 file.

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

```
target # /usr/share/ti/ti-dmai-apps/audio_decode_io1_dm368.x470MV -c
mp3_dec -e decode -i music.mp3 -n 1000 -o output.pcm
```

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 and 8GB SDHC 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. Plug an SD card on Linux host machine.

2. 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.

3. The uflash utility provided with the u-boot component has to be built prior to running the script. See #Rebuilding_the_DVSDK_components to determine how to build u-boot if not previously built.

4. Run mksdboot script installed in DVSDK as shown 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 and insert into EVM board.

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. Power OFF the board and change SW4 switch on EVM to boot from SD card.

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

6. 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 machine to the DM368 board so that the script properly sets up the u-boot variables.

7. Power cycle the board, it will boot from SD card.

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

```
host $ sudo ${DVSDK}/bin/mksdboot.sh --device /dev/sdc --sdk ${DVSDK} \
 /path/to/dvsdk_dm368-evm_4_xx_xx_xx_setuplinux
```

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

Reference

http://processors.wiki.ti.com/index.php/Category:DM36x

http://processors.wiki.ti.com/index.php/H.264_DM36x_Ver_2.0_Codec

http://processors.wiki.ti.com/index.php/DM36x_H.264_encoder_FAQ

http://processors.wiki.ti.com/index.php/DM36x_Rate_Control_Modes

http://processors.wiki.ti.com/index.php/H.264_DM36x_Ver_2.1_Codec

http://processors.wiki.ti.com/index.php/DM36x_Schematic_Review_Checklist

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

http://processors.wiki.ti.com/index.php/DaVinci_PSP_03.01_GA_(r38)_Release_Notes

http://processors.wiki.ti.com/index.php/DaVinci_PSP_03.01_Linux_Installation_User_Guide

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