DM816x EZ Software Developers Guide

From MediaWiki

Translate this page to Translate Show original

Contents

Welcome to the DM816x EZSDK Software Developer's Guide

Thank you for choosing the DM816x Evaluation Module (EVM) for your application. The purpose of this guide is to get you going with developing software for the DM816x on a Linux development host only.

Note! This Software Developer's Guide (SDG) supports version 5.05 of the DM816x EZSDK which is only for Linux host development.

Note! All instructions in this guide are for Ubuntu 10.04 LTS and Ubuntu 11.10. At this time, these are the only supported Linux host distributions 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 3.xx and is a new document in the EZSDK superseding the *Getting Started Guide*.

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

```
host $ export EZSDK="${HOME}/ti-ezsdk_dm816x-evm_xx_xx_xx_xx"
```

Setting up the EZ SDK

You will need an ARM Linux development environment, in case you do not have one please refer to this link to see how to set one up.

Configuration of ARM Linux development Environment

The EZ SDK 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 ethernet access as it will update your Ubuntu Linux development host with the packages required to develop using the

EZ SDK. Execute the script using:

```
host $  ${EZSDK}/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 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 USB-UART (J6). If you want to use a windows host for connecting to the target instead, see the #Setting_up_Tera_Term section.

Note! To boot the board from NFS, you may need to change the boot switch settings on your EVM. Please refer the UBoot user guide in the board-support/docs folder for more information on the switch settings.

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 is launched automatically. If you exit from Matrix and if you would like to start it again, execute the following command on the target board:

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

Make sure you have terminated the Matrix 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 EZSDK so that you can find the EZSDK components
include ${EZSDK}/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 $EZSDK variable is still set using:

```
host $ echo $EZSDK
```

This command should print your EZSDK 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/dm816x
host $ cp helloworld ${HOME}/targetfs/home/root/dm816x
```

6. On your target this application will be accessible from `/home/root/dm816x/helloworld`. Execute it on your target:

```
target # /home/root/dm816x/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 EZSDK). This section shows how to execute those applications in the provided filesystem.

Before running these ensure that Matrix application is not running. This can be done by executing the following command in the serial terminal.

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

If you wish to restart the Matrix application at a later time, you can execute the following command.

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

Running Audio applications

1. Refer to <EZSDK installed root>/component-sources/omx_xx_xx_xx_xx/src/ti/omx/docs/OMX_xx_xx_xx_xx_UserGuide.pdf in "Audio Decode example" section's "Running the application" on how to run audio decode example

2. Refer to <EZSDK installed root>/component-sources/omx_xx_xx_xx_xx/src/ti/omx/docs/OMX_xx_xx_xx_xx_UserGuide.pdf in "Audio Encode example" section's "Running the application" on how to run audio encode example

Running the SysLink examples

The SysLink comes with a few sample applications. To run the sample applications such as "MessageQ" use the below set of commands.

Note! The syslink samples use a different memory map from the default EZSDK installation. In order to run syslink examples, you must boot with a different memory for linux. When booting, ensure that the linux bootargs is changed from the default values to **MEM=169M**

Note! The syslink samples cannot be run out with graphics or firmware loaded. Please execute the following steps to teardown the graphics plane and ensure that no firmware is running.

```
target # /etc/init.d/pvr-init stop
```

```
target # /etc/init.d/matrix-gui-e stop
target # /etc/init.d/load-hd-firmware.sh stop
```

Now the system is ready to run all syslink samples.

```
target # modprobe syslink
target # cd /usr/share/ti/syslink-examples/TI816X
target # cd helloworld
```

Execute the following script to run the example application

```
target # ./run.sh
```

The target terminal window will output the results of the examples executed.

Note! To run other examples, go to corresponding example folder and execute ./run.sh.

Please refer to the syslink documentation in component-sources/syslink_x_xx_xx_xx/docs to experiment on these examples and for further information on how to change the memory map.

Running the Codec Engine examples

The Codec Engine package comes with a small set of examples.

Note! The syslink samples use a different memory map from the default EZSDK installation. In order to run syslink examples, you must boot with a different memory for linux. When booting, ensure that the linux bootargs is changed from the default values to **MEM=169M**

Note! The Codec Engine examples cannot be run out with graphics. Please execute the following steps to teardown the graphics plane and ensure that no firmware is running.

```
target # /etc/init.d/pvr-init stop
```

```
target # /etc/init.d/matrix-gui-e stop
target # /etc/init.d/load-hd-firmware.sh stop
```

To run the application, enter the following set of commands on the target:

```
target # cd /usr/share/ti/ti-codec-engine-examples
```

Ensure that syslink and cmem modules are installed with memory configuration as below

```
target # modprobe syslink
target # modprobe cmemk phys_start=0x94000000 phys_end=0x947fffff \
pools=20x4096,10x131072,2x1048576
```

To run the audio1_copy example, you will need to run the following commands.

```
target # cd audio1_copy
```

```
target # ./app_remote.xv5T
```

To run other examples, please refer the Codec Engine documentation.
http://ap-fpdsp-swapps.dal.design.ti.com/index.php?title=DM816x_EZ_Software_Developers_Guide&action=edit

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.

Note! - You should quit the Matrix GUI application before running Qt/Embedded examples.

```
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

QT Examples with SGX Acceleration please see QT Tips on TI Processors Wiki

Running the Graphics SDK examples

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

```
target # cd /usr/bin/SGX/demos/Raw
target # ls
```

Here is the list of apps you will see:

OGLES2ChameleonMan OGLESEvilSkull OGLESPolyBump

OGLES2Coverflow OGLESFilmTV OGLESShadowTechniques

OGLES2FilmTV OGLESFiveSpheres OGLESSkybox

OGLES2PhantomMask OGLESFur OGLESTrilinear

OGLES2Shaders OGLESLighting OGLESUserClipPlanes

OGLES2Skybox2 OGLESMouse OGLESVase

OGLES2Water OGLESOptimizeMesh

OGLESCoverflow OGLESParticles

Execute the following command to run 3D Graphics application, this particular example is for an album coverflow.

```
target # ./OGLES2Coverflow
```

After you see the output on the display interface, hit *q* to terminate it

Running GStreamer pipelines

The EZSDK comes with GStreamer, and you can construct your own pipelines, see [1].

Note: In order to see the video output, the graphics planes need to be turned off. By default, graphics plane 0 is tied to HDMI, graphics plane 1 is tied to HD DAC and graphics plane 2 is tied to SD. For more information on the graphics planes and their sysfs entries, please read the VPSS guide in PSP documentation.

Turn off the Graphics Plane 0 by running the following command.

```
target # echo 0 > /sys/devices/platform/vpss/graphics0/enabled
```

In case Graphics Planes 1 and 2 are currently open, then they need to be disabled as well. This is only required if the video output needs to be directed to the HD-DAC or SD displays.

```
target # echo 0 > /sys/devices/platform/vpss/graphics1/enabled
target # echo 0 > /sys/devices/platform/vpss/graphics2/enabled
```

The following pipeline decodes a H.264 Elementary Stream and displays it to HDMI:

```
target #     gst-launch -v filesrc location=/usr/share/ti/data/videos/dm816x_1080p_demo.264 \
   ! 'video/x-h264' ! h264parse access-unit=true ! omx_h264dec !  omx_scaler \
   ! omx_ctrl display-mode=OMX_DC_MODE_1080P_60 ! omx_videosink sync=false
```

Running the RPE examples

To run the RPE example, follow the following steps:

1. Go to the directory, where the RPE example and firmware binary are present.

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

2. Load the DSP firmware binary with the codec configurations using the firmware loader.

```
target # firmware_loader 0 dm81xx_c6xdsp_debug.xe674 start
```

3. Run the AAC-LC decoder example with the sample media and configuration file.

```
target # ./aacdec_a8host_debug.xv5T -i aacdectest.cfg
```

Using the devkits

At the top level directory of the EZSDK you will find one or more devkits, typically *linux-devkit* and *dsp-devkit*. The devkits are:

1. The tools, libraries and headers to develop applications for a specific hardware subsystem (e.g. the arm or the dsp).
2. The devkits are relocatable, meaning you can move them to another location on your filesystem and they will still work (see #Moving the devkits below).
3. The devkits do **not** contain source code or build files. If you want to change components, or make a

change to a component, the devkit will need to be regenerated, see #Regenerating the devkits below.
4. The devkits contain the documentation of the TI components in one location.

The devkits were introduced to provide a more unified view of what is available for each hardware subsystem and present a system view of the software in the EZSDK as opposed to a component view. Since they are relocatable, they are also easier for a user to check in to version control.

Note! The components themselves are still available from the *${EZSDK}/component-sources* directory, and the *${EZSDK}/Rules.make* file still points to all the right component directories. If you do not wish to build against the devkits, but directly against the components, this is still possible.

Regenerating the devkits

You may need to regenerate the devkit because you changed a component version, in which case you (Syslink example):

1. Download the new Syslink release from the web.
2. Read the release notes to make sure all dependencies are satisfied, or you may have to update more components.
3. Extract the downloaded release on your target filesystem, and update the *SYSLINK_INSTALL_DIR* variable in *${EZSDK}/Rules.make* to point to the new location.
4. Enter the *${EZSDK}* directory.
5. Clean the EZSDK by executing *make clean* so that files not relevant to your target (linux, dsp etc.) don't get copied.
6. Make sure the components are compiled for Linux by executing *make components_linux*.
7. Execute *make linux-devkit* to populate the linux-devkit with the TI components.
8. Clean the EZSDK by executing *make clean*.
9. Make sure the components are compiled for the DSP by executing *make components_dsp*.
10. Execute *make dsp-devkit* to populate the dsp-devkit with the TI components.

If you have modified a component, in which case the support TI will be able to provide is limited, you can regenerate the devkits using only the last 7 steps above.

Note that not all components contribute to all devkits. You may only have to regenerate e.g. the dsp-devkit if you update or change sysbios.

Verifying the devkit integrity

When the devkits are created, two files are generated at the devkit's top level directory:

1. *install.log* contains the TI components and versions used in the devkit.
2. *md5sums* contains the md5sums of all files in the devkit.

In addition, the *${EZSDK}/docs* directory contains the md5sums of the devkits at the time of release.

If a file has been changed, or a component updated, the md5sums will have changed. To verify whether this is the case for e.g. the dsp-devkit, enter the dsp-devkit directory and execute:

```
host $ md5sum -c ${EZSDK}/docs/dsp-devkit.md5sums | grep -v OK$
```

If there is no output from this command, your integrity with the devkit released by TI is ok. If there is an error, the offending files will be printed.

Moving the devkits

The devkits are relocatable, whereas the rest of the EZSDK is not. This means that you can put the devkits in any directory on your Linux filesystem, as long as you do the following (dsp-devkit example):

1. If you want to be able to regenerate the dsp-devkit (see #Regenerating the devkits, you'll need to update the *DSP_DEVKIT_DIR* variable in *${EZSDK}/Rules.make*.
2. Before building against the dsp-devkit from the command line, you need to "source" the environment-setup script (don't forget the **.**):

```
host $ . /path/to/dsp-devkit/environment-setup
```

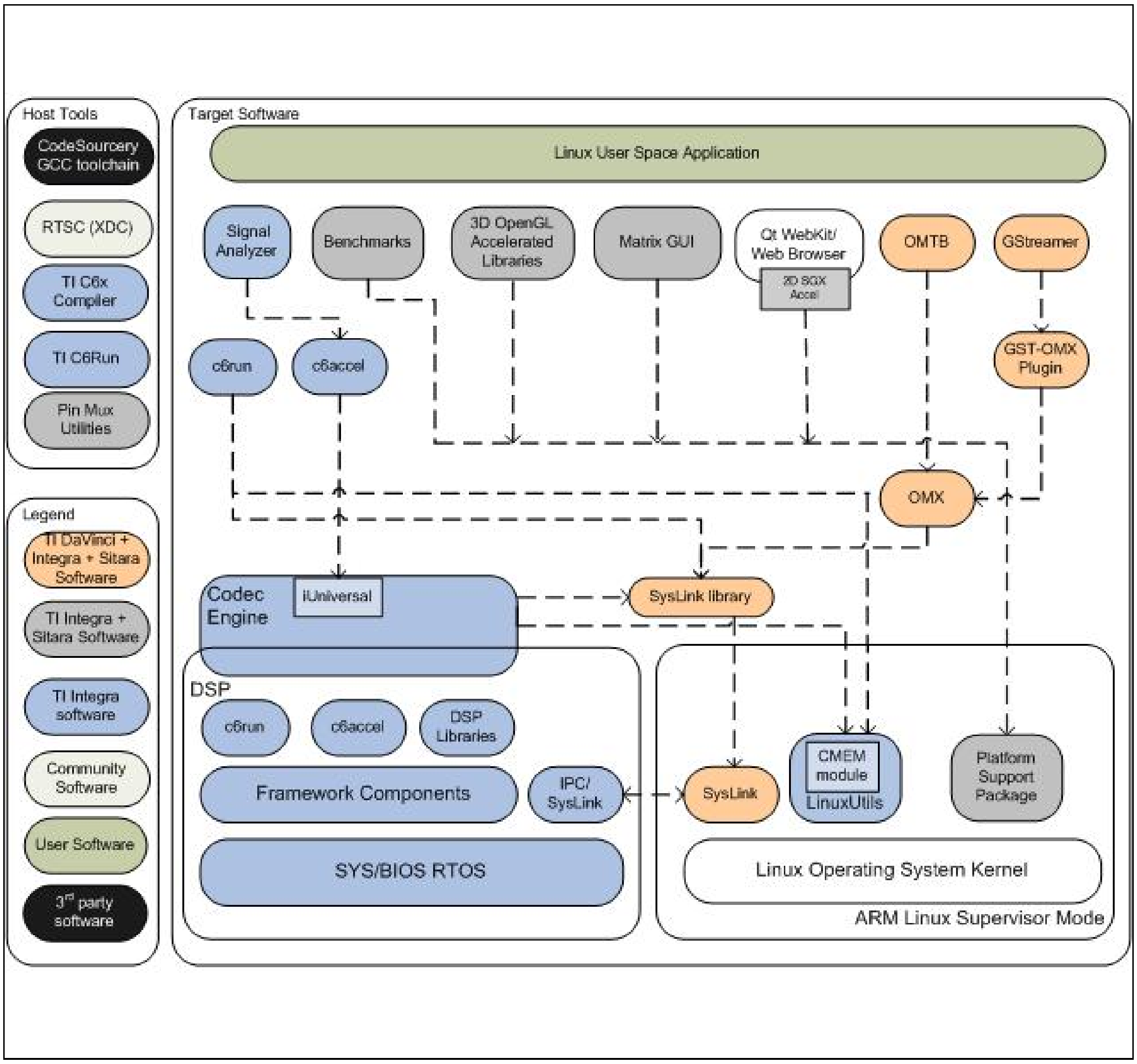
Note! For the linux-devkit you will currently have to edit the first line of *${EZSDK}/linux-devkit/environment-setup* to change the *SDK_PATH* variable to point to your new location. You can get your new location by executing the following in the linux-devkit directory:

```
host $ pwd
```

Note! The dsp-devkit does not contain xdctools. If you need to relocate the devkit, the path to xdctools needs to be updated in dsp-devkit/environment-setup.

Note! After relocating the devkits, Rules.Make must be updated. Please update CODEGEN_INSTALL_DIR, LINUX_DEVKIT_DIR and DSP_DEVKIT_DIR according to their new locations on your filesystem.

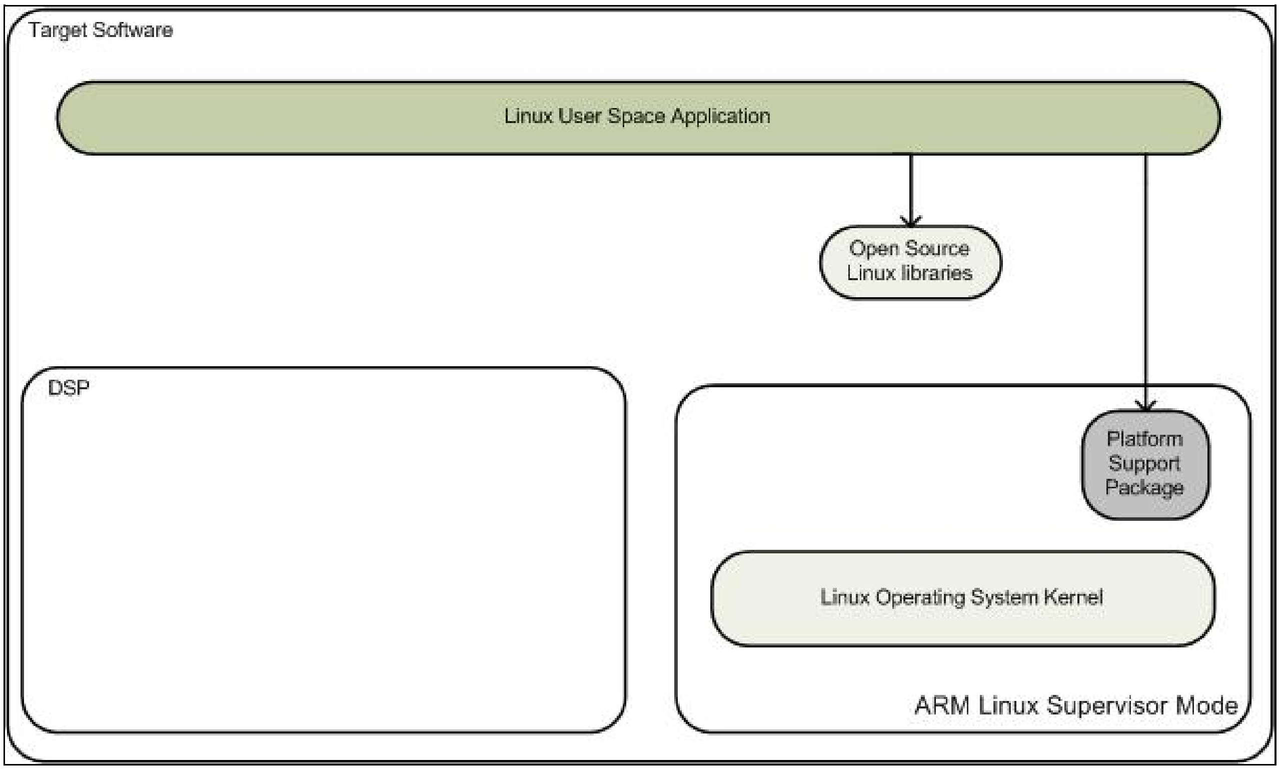
EZSDK software overview



Overview of the EZ SDK Software stack

The EZ SDK contains many software components. Some are developed by Texas Instruments and some are developed in and by the open source community(White). 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.

Creating a Linux application


Target Software
Linux User Space Application
Open Source Linux libraries
DSP
Platform Support Package
Linux Operating System Kernel
ARM Linux Supervisor Mode

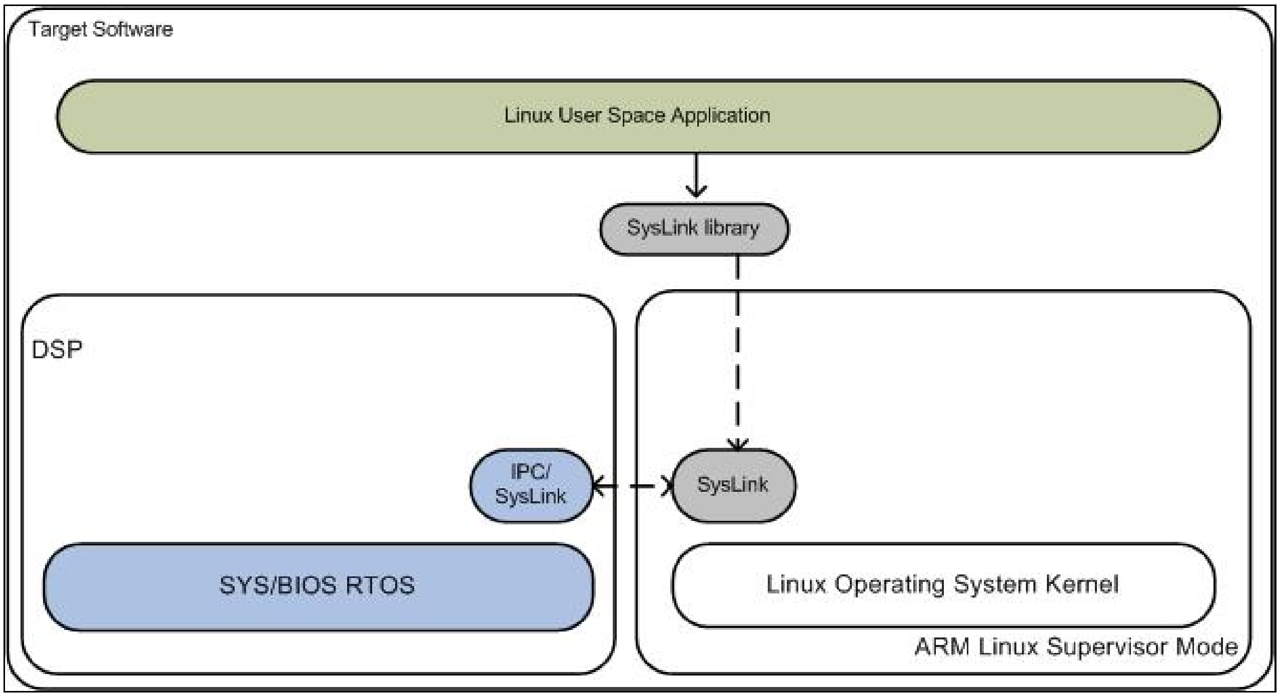
Overview of a basic Linux application component usage

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 EZSDK
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.	board-support
Linux kernel	The Linux kernel with the PSP device drivers	board-support/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 SYS/Link application


Target Software
Linux User Space Application
SysLink library
DSP
IPC/ SysLink
SysLink
SYS/BIOS RTOS
Linux Operating System Kernel
ARM Linux Supervisor Mode

Overview of a SYSlink application component usage

SYS/Link(SysLink) is foundation software for the inter-processor communication across the HLOS-RTOS boundary. It provides a generic API that abstracts the characteristics of the physical link connecting HLOS and RTOS from the applications. It eliminates the need for customers to develop such link from scratch and allows them to focus more on application development.

SysLink 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 HLOS operating system specific details from applications, otherwise needed for talking to the hardware (e.g. interrupt services)
- Applications written on SysLink 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 SysLink.

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

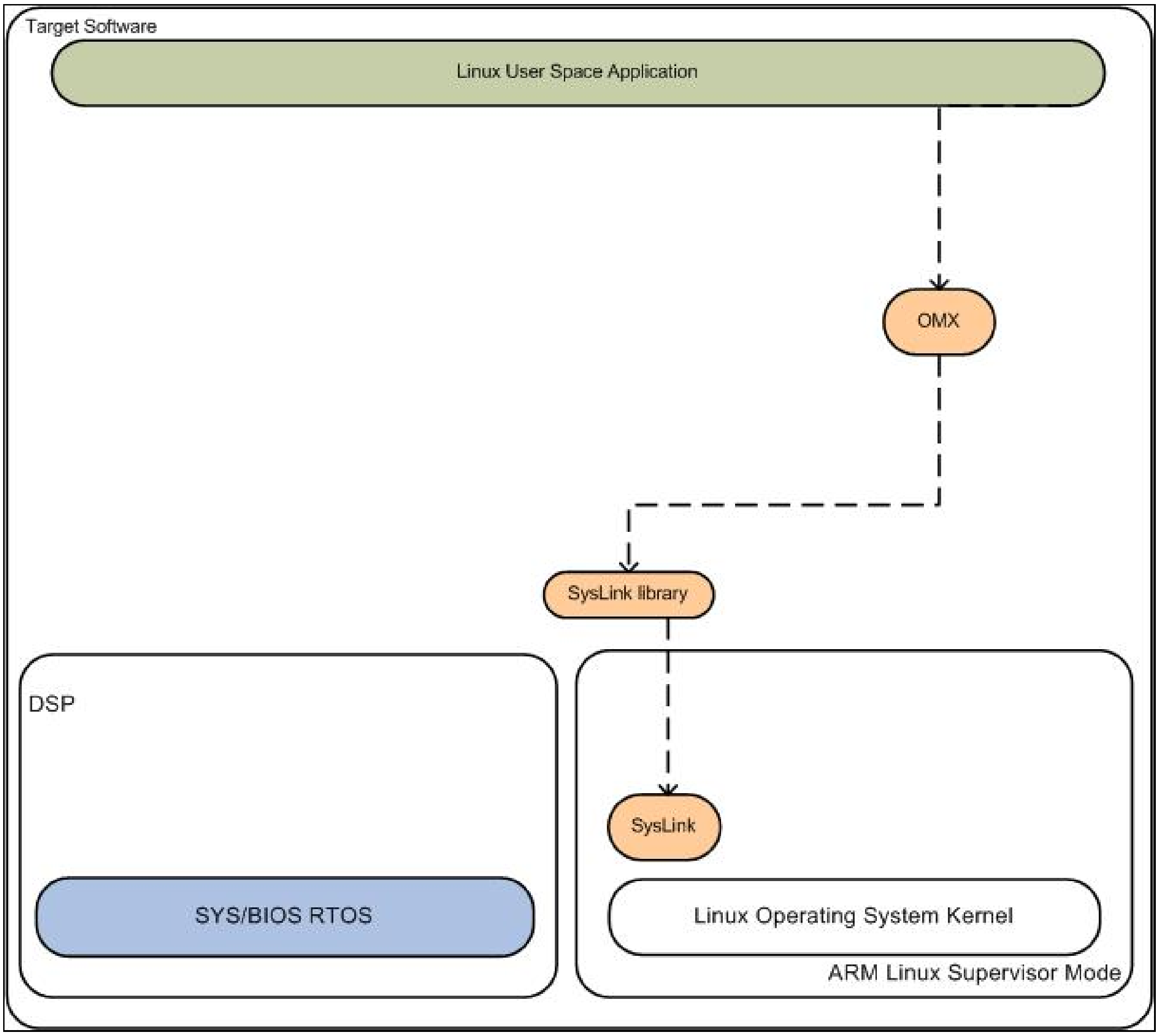
Component	Purpose in this application	Location in the EZSDK
SYS/BIOS	Real-Time Operation System for TI DSPs	component-sources/sysbios_x_xx_xx_xx
SysLink	HLOS to RTOS communication link for passing messages and data in	component-sources/syslink_x_xx_xx_xx

	multiprocessor systems	
IPC	RTOS communication link for passing messages and data communication	component-sources/ipc_x_xx_xx_xx
Platform Support Package	Provides device drivers for the EVM and documentation and examples to support them	board-support
C6000 Code Generation Tools	TI DSP code generation tools	dsp-devkit/cgt6x_x_x_xx

Good application examples to start from include:

- The sample applications (component-sources/syslink_x_xx_xx_xx/packages/ti/syslink/samples provide simpler and smaller examples on how to use SysLink.

Creating an OpenMax IL application

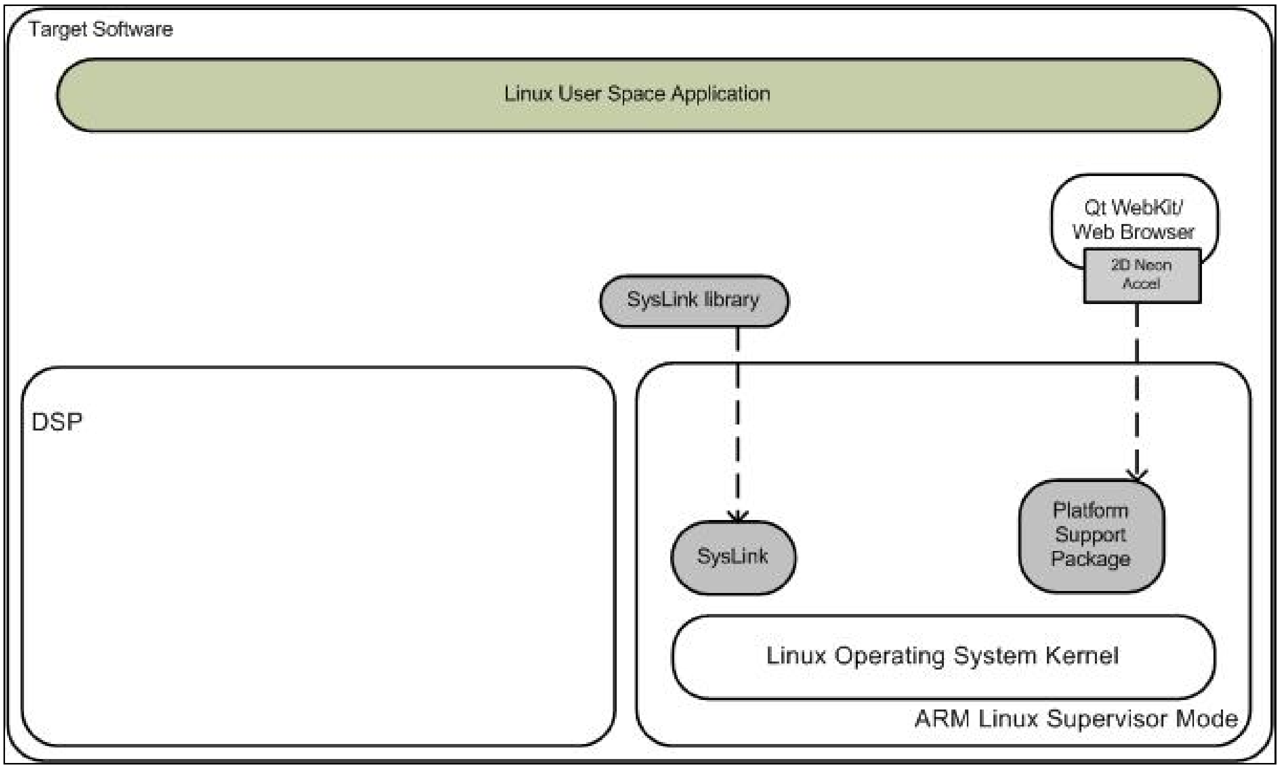

Target Software
Linux User Space Application
OMX
SysLink library
DSP
SYS/BIOS RTOS
SysLink
Linux Operating System Kernel
ARM Linux Supervisor Mode

Overview of a basic OMX application component usage

The OpenMax IL package wraps key Multimedia functions which can be invoked from the ARM side using simple API calls. 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 EZSDK
OpenMax	OpenMax IL multimedia framework for the applications invoking multimedia codecs and other algorithms.	component-sources/omx_xx_xx_xx_xx
SysLink	HLOS to RTOS communication link for passing messages and data in multiprocessor systems	component-sources/syslink_x_xx_xx_xx

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 EZ SDK
Qt/Embedded	Provides a Graphical User Interface toolkit	linux-devkit/arm-none-linux-gnueabi/usr/lib/libQt*

SysLink	HLOS communication link for passing messages and data in multiprocessor systems	component-sources/syslink_x_xx_xx_xx
Platform Support Package	Provides device drivers for the EVM and documentation and examples to support them.	board-support

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

EZ SDK Linux development kit includes the Qt/Embedded host tools and development header and libraries.

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

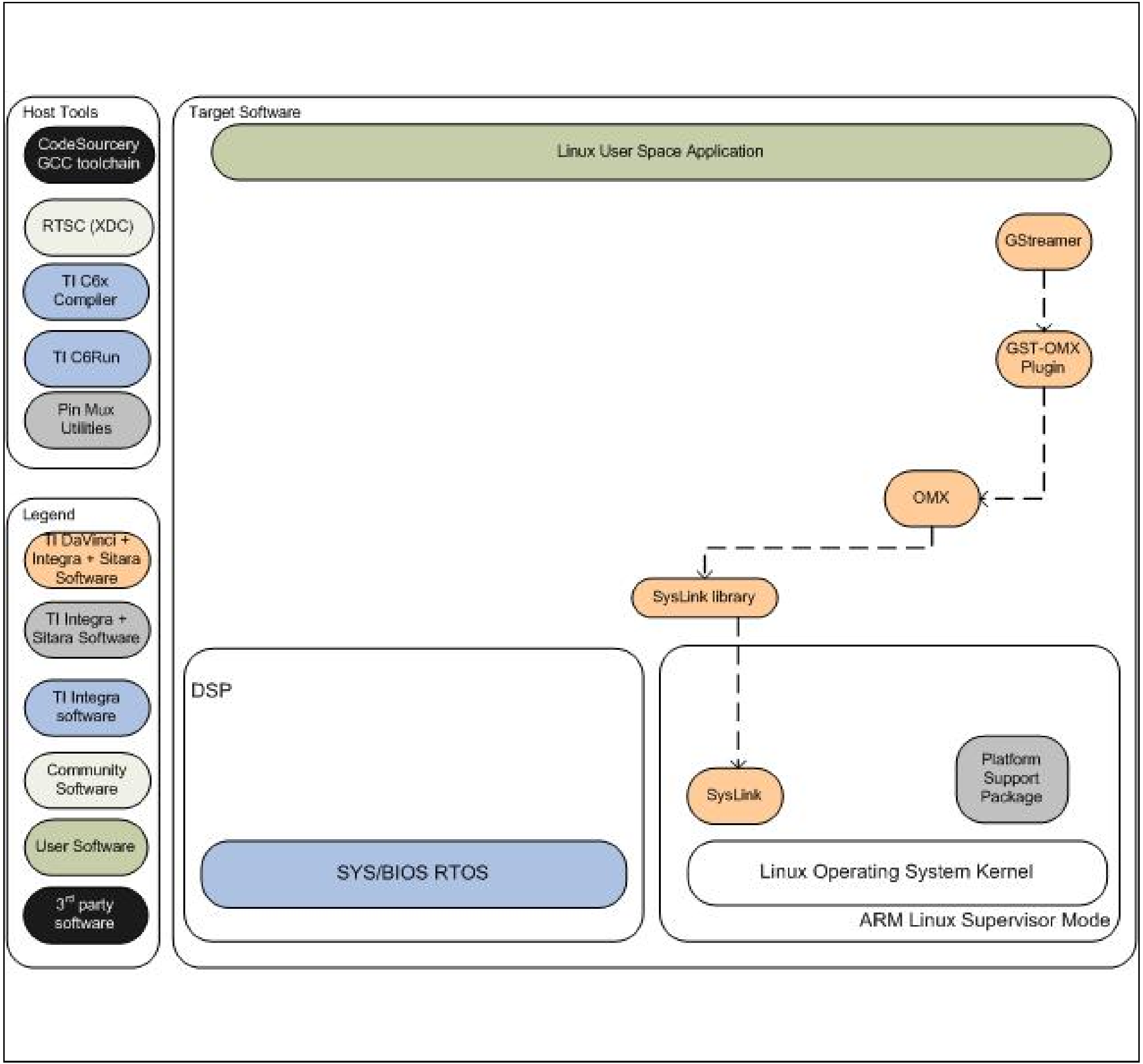
2. Next, follow the typical Qt/e recommended method for cross compiling your application on host.

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

Matrix User's Guide

Please refer to the The Matrix User's Guide for more information.

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. The gst-openmax plugin accelerates multimedia using OpenMax.

Compared to creating an application directly on top of OMX 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 OMX app, these are used:

Component	Purpose in this application	Location in the EZSDK
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 at gstreamer.ti.com.

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

Compiling a GStreamer application

EZSDK 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 $ arm-none-linux-gnueabi-gcc -o decode decode.c `pkg-config --libs --cflags gstreamer-0.10`
```

Additional Procedures

Setting up cross compilation environment

To enable your application development, EZ SDK 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 ${EZSDK}/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.

Modifying the EZSDK Memory Map

By Default, the EZSDK Ships with a memory map that is configured for 1GB of DDR. More details on how to configure the memory map to different memory sizes or to even change the partitioning is available on TI's Processor Wiki at http://processors.wiki.ti.com/index.php/EZSDK_Memory_Map.

How to setup the EZ SDK for DDR2 EVMs

By default, the EZ SDK software is setup for DDR3 EVMs and configured for 800MHz DDR3. If you have an older DDR2 EVM, then you will need to rebuild uboot after editing uboot sources to enable support DDR2.

Edit arch/arm/include/asm/arch-ti81xx/clocks_ti816x.h in the U-Boot Source.

```
host $ cd $EZSDK/board-support/u-boot-YYYY.MM-pspxx.yy.zz.pp
host $ vi arch/arm/include/asm/arch-ti81xx/clocks_ti816x.h
```

Comment the line defining DDR_PLL_796 and add following line to the clocks_ti816x.h

```
//#define DDR_PLL_796
#define DDR_PLL_400
```

Next, edit include/configs/ti8168_evm.h in the U-Boot Source.

```
host $ vi include/configs/ti8168_evm.h
```

Uncomment the DDR2 config and comment the DDR3 config macros

```
//#define CONFIG_TI816X_EVM_DDR3
#define CONFIG_TI816X_EVM_DDR2
```

Now, build the u-boot.

```
host $ cd ${EZSDK}
host $ make u-boot
```

This will build the MLO, u-boot.bin and u-boot.noxip.bin files. These u-boot files need to be used instead of the default DDR3 image. They can be copied over to the board-support/prebuilt-images directory. Please follow procedures on how to build an SD Card if you would to create an SD Card image for your DDR2 EVM.

For more information please refer the PSP Documentation.

Rebuilding the EZ SDK components

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

Note: The EZ SDK 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 EZSDK components by first entering the EZ SDK directory using:

```
host $ cd ${EZSDK}
```

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

```
host $ make help
```

Some of the components delivered in the EZ SDK are not pre-built. The provided 'make clean' & 'make components' build targets are designed to clean and build all components (e.g. Linux Kernel, 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 $ make <target>_install
```

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

You can remove all components generated files at any time using:

```
host $ make clean
```

And you can rebuild all components using:

```
host $ make all
```

You can then install all the resulting target files using:

```
host $ make install
```

Creating your own Linux kernel image

The pre-built Linux kernel image (uImage) provided with the EZSDK 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 EZSDK, 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_EZ_SDK to setup your build environment.

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

```
host $ cd ${EZSDK}
host $ make linux_clean
host $ make linux
host $ 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_EZ_SDK section.

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

```
host $ cp ${HOME}/targetfs/home/root/dm816x-evm/boot/uImage /tftpboot
```

5. Copy the exported Linux kernel modules from the EXEC_DIR to the /lib/modules directory:

```
host $ sudo cp -r ${HOME}/targetfs/lib/modules ${HOME}/targetfs/lib/modules_original
host $ sudo cp -r ${HOME}/targetfs/home/root/dm816x-evm/lib/modules ${HOME}/targetfs/lib
```

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

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

Note! In this release of the EZ SDK, U-Boot does not read the MAC Address from eFuses. As a result the ethernet MAC Address needs to be set manually by choosing a valid random MAC Address. More details are available in the PSP U-Boot documentation. Please run the following command to set the ethernet MAC Address

```
u-boot :> set ethaddr <value of the MAC address chosen>
```

7. Note that when you change your kernel, it is important to rebuild the kernel modules supplied by the EZSDK 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 ${EZSDK}
host $ make cmem_clean
host $ make cmem
host $ make cmem_install
host $ sudo mv ${HOME}/targetfs/lib/modules/2.6.32-rc2-davinci1/kernel/drivers/dsp/cmemk.ko \
${HOME}/targetfs/lib/modules/2.6.32-rc2-davinci1/kernel/drivers/dsp/cmemk.ko.orig
host $ sudo cp ${HOME}/targetfs/home/root/dm816x-evm/cmem/cmemk.ko \
${HOME}/targetfs/lib/modules/2.6.32-rc2-davinci1/kernel/drivers/dsp
```

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.

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
```

NOTE: ***Kernel Bootargs can be generated by running the setup script. See the section***

***#Setting_up_the_EZ_SDK** for details on running the setup script.*

How to create an SD card

This section explained the procedure required for creating SD card image for DM816x and the steps has been verified on 2GB, 4GB and 8GB SD cards.

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: [sdb] 3862528 512-byte logical blocks: (1.97 GB/1.84 GiB)
   [14365.310602] sd 6:0:0:1: [sdb] Assuming drive cache: write through
   [14365.325542] sd 6:0:0:1: [sdb] Assuming drive cache: write through
   [14365.325571]  sdb: sdb1 sdb2
```

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

3. Run mksdboot script installed in EZ SDK as show below

```
host $ sudo ${EZSDK}/bin/mksdboot.sh --device /dev/sdb --sdk ${EZSDK}
```

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

4. Power OFF the DM816x EVM.

5. Set the SW3 switch to boot from SD.

- SW3 = 0000010111 (high to low, i.e. SW3.1 = 1)
- 1 = "On" position on the switch

6. Insert the SD card into the DM816x EVM.

7. Power ON the EVM.

Note! If your flash already has a u-boot environment stored, this will get picked up even while booting from SD-card. If this is the case, halt the u-boot auto boot process and enter the following command to erase the NAND environment variables:

```
u-boot :> nand erase 0x260000 0
```

Note! If you want to recreate the full SD card with a separate partition for the EZSDK installer and the CCSv5 installer execute the following:

```
host $ sudo ${EZSDK}/bin/mksdboot.sh --device /dev/sdb --sdk ${EZSDK} \
/path/to/ezsdk_dm816x-evm_5_xx_xx_xx_xx_setuplinux /path/to/setup_CCS_5.x.x.xxxxx.tar.gz
```

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

How to display a logo at boot

This section explains the procedure to display an image at boot time. Booting with the SD card created by following the steps described in the previous section displays the Texas Instruments logo.

The linux logo appears by default when the u-boot stages are being loaded. After the linux logo, one can give the commands to load an image using the tftp and display the image at the u-boot command prompt.

To display a bmp image(24 bit) as logo, follow the following steps:

1. Copy the image to the tftp directory in the host system.

```
host $ sudo cp $EZSDK/bin/ti_logo.bmp /tftpboot
```

2. Add the commands to display the bmp image at the u-boot to environment variable "bootcmd" and save the environment to NAND (**Note:** NAND should be enabled).

```
u-boot :> setenv bootcmd 'tftp 0x81000000 ti_logo.bmp; bmp display 0x81000000; $bootcmd'
```

The logo would appear for around 15 seconds by default. To change the time for which the logo is displayed, change the value of environment variable *logotime* at the u-boot but it should not be greater than 15. Save the environment to NAND to see the change in time for which the logo is displayed during the next boot.

```
u-boot :> setenv logotime value
```

Note: By default, if the environment variable "logotime" is not defined in u-boot environment, it uses the default value of 15 to display the logo for that much time.

How to copy boot loaders to NAND flash

The setup-nand-images.sh in $EZSDK/bin can be used to flash uboot to NAND. By running this script images from SD Card will be copied over to the NAND.

Note! You need ensure that the EVM is connected to the Host via RS-232. You also need to ensure that the NAND is available and turned on.

Please refer the U-boot documentation under the board-support folder in your EZ SDK installation for the procedure required for copying boot loaders (MLO and u-boot) on NAND flash.

How to change the display (Graphics) resolution

The EZ SDK supports multiple displays resolutions but by default boots with 1080p60 resolution. To change the resolution on you HDMI display, you can execute the following command. The command below demonstrates resolution change to 720p60. In a similar manner, the resolution can be set to 720p60, 1080i60, 1080p30 and 1080p60.

```
target #  cd /usr/share/ti/ti-media-controller-utils
target # ./change_resolution.sh 720p60
```

Note! You will need to reboot your board after executing the above command.

Note! If you want to change the video resolution, then you must edit the OMTB script or call the necessary OMX API. For more information please refer the OMTB and OMX documentation. You must make sure that both the Graphics and Video resolution is configured to the same value.

How to change from OMX to V4L2 firmware for capture/display

The OMX Capture/Display software is orthogonal to the V4L2 drivers. Hence a different set of firmware is provided if you wish to use V4L2 instead of OMX VFDC, VFPC and VFCC components. To support V4L2 and a different initscript is provided. Copy over the V4L2 specific initscript to the /etc/init.d directory.

```
target # cd /usr/share/ti/ti-media-controller-utils
target # cp load-hd-v4l2-firmware.sh /etc/init.d/load-hd-firmware.sh
target # sync
```

Now power cycle the board and it will be setup to load the alternate firmware which supports V4L2.