DM814x AM387x EZ Software Developers Guide

Translate this page to Translate Show original

Contents

Welcome to the DM814x/AM387x EZ Software Developer's Guide

Thanks you for choosing the DM814x/AM387x Evaluation Module (EVM) for your application. The purpose of this guide is to get you going with developing software for the DM814x/AM387x on a Linux development host only.

Note! This Software Developer's Guide (SDG) supports version 5.02 of the DM814x/AM387x EZSDK 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 Easy Software Development Kit (EZ SDK). 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 EZSDK 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 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_dm814x-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

Please get the Code Sourcery tools that will be the compiler for the ARM Linux applications.

Code Sourcery Tools Download

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 RS-232. 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/dm814x/am387x
host $ cp helloworld ${HOME}/targetfs/home/root/dm814x/am387x
```

6. On your target this application will be accessible from `/home/root/dm814x/am387x/helloworld`. Execute it on your target:

```
target # /home/root/dm814x/am387x/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 the DaVinci and Integra demo examples

The EZSDK comes with example applications.

For DaVinci multimedia, you can use OMTB to run different OpenMAX IL chains. OMTB is the OpenMax Test Bench which is a command-line utility used for validating OpenMax.

To see the Integra device examples that are available, check out this directory on the target:

```
target # cd /usr/share/ti/ti-integra-demos
target # ls
```

Running OMTB

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
```

Execute the following commands to run OMTB.

```
target # cd /usr/share/ti/ti-omtb
target # ./omtb_<platform>_a8host.xv5T <script-name>.oms
```

For more information on OMTB and how to construct OpenMAX IL chains please refer the OMX and OMTB documentation.

Note: OMTB will require a script to run and should not be called without a valid script as an argument.

Note: You will need to turn the graphics planes back on if you wish to run any Graphics applications. You will also need to revert the change to /etc/init.d/load-hdvpss-firmware.sh in case you wish to see the video demo from the Matrix Application Launcher.

Note: The dual_display_encode_decode.oms script will pause within a couple of seconds. This script is designed to work from matrix and hence has this pause functionality built in. A script which does not pause is also present and can be used instead.

Running the Signal Analyzer Demo

Execute the following command to run the Signal Analyzer Demo application.

Note! - You should quit Matrix GUI before running the Signal Analyzer Demo manually from the command prompt.

```
target #  cd /usr/share/ti/ti-integra-demos
target # ./runSADemo -qws
```

You can view information on options/features available to the demos at How to Run Signal Analyzer Demo. This information is also located in the *docs/* folder along with other documents.

Running the SysLink examples

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

Note! The syslink samples should not 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-hdvicp2-firmware.sh stop
target # /etc/init.d/load-hdvpss-firmware.sh stop
```

Now the system is ready to run all syslink samples.

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

Execute the following script to run the example application

```
target # ./messageqapp_debug 1 DSP dsp/messageq_ti81xx_dsp.xe674
```

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

There are other syslink examples present in /usr/share/ti/ti-syslink-examples directory. Please refer to the syslink documentation in component-sources/syslink_x_xx_xx_xx/docs to experiment on these examples.

Running the Codec Engine examples

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

Note! The Codec Engine examples should not 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-hdvicp2-firmware.sh stop
target # /etc/init.d/load-hdvpss-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 cmem module is installed with memory configuration as below

```
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.

Running the C6Run examples

The C6Run package comes with a small set of applications to demonstrate its usage.

Note! The C6Run samples cannot be run without first bringing down the graphics plane. Please run the following commands first.

```
target # /etc/init.d/pvr-init stop
target # /etc/init.d/matrix-gui-e stop
target # /etc/init.d/load-hdvicp2-firmware.sh stop
target # /etc/init.d/load-hdvpss-firmware.sh stop
```

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

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

The cmem and syslink modules need to be installed. So ensure that cmem module is re-installed with new memory configuration as below

```
target # modprobe syslink
target # modprobe cmemk phys_start=0x96C00000 phys_end=0x98000000 allowOverlap=1
```

Execute the following command to run the example application

```
target # cd examples/c6runapp/hello_world
target # ./hello_world_dsp
```

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.

Note! The C6Accel apps should not 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-hdvicp2-firmware.sh stop
target # /etc/init.d/load-hdvpss-firmware.sh stop
```

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

```
target # cd /usr/share/ti/c6accel-apps/c6accel_dsplib_testapp
```

Ensure that cmem module is installed with memory confiration as below

```
target # modprobe cmemk phys_start=0x96C00000 phys_end=0x98000000 pools=20x4096
```

Execute the following command to run the example application

```
target # ./c6accel_dsplib_testapp
```

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/c6accel_dsplib_testapp/benchmarking.txt
```

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

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

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 (Codec Engine example):

1. Download the new Codec Engine 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 *CE_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 **.**):

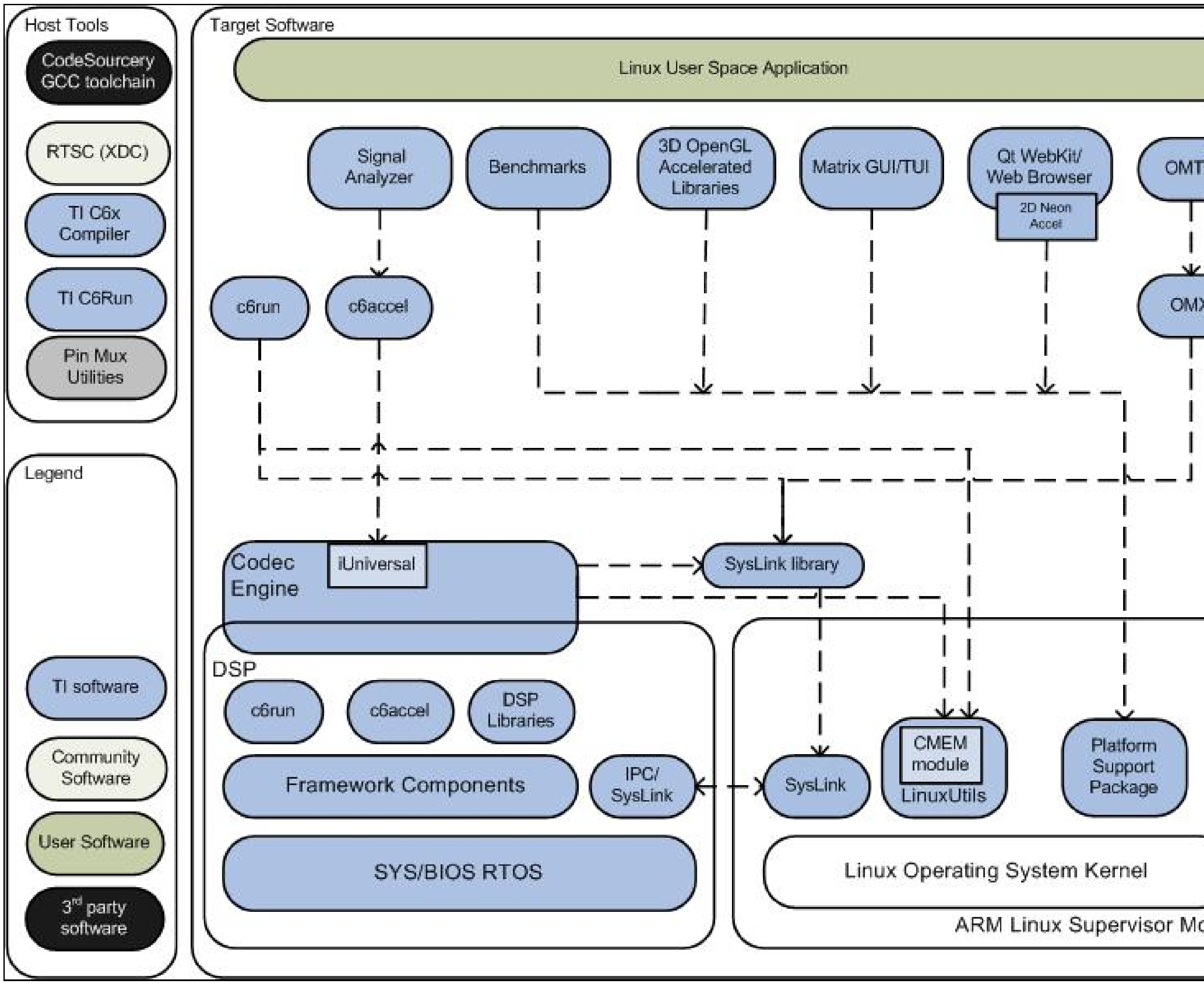
```
host $ . /path/to/dsp-devkit/environment-setup
```

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.

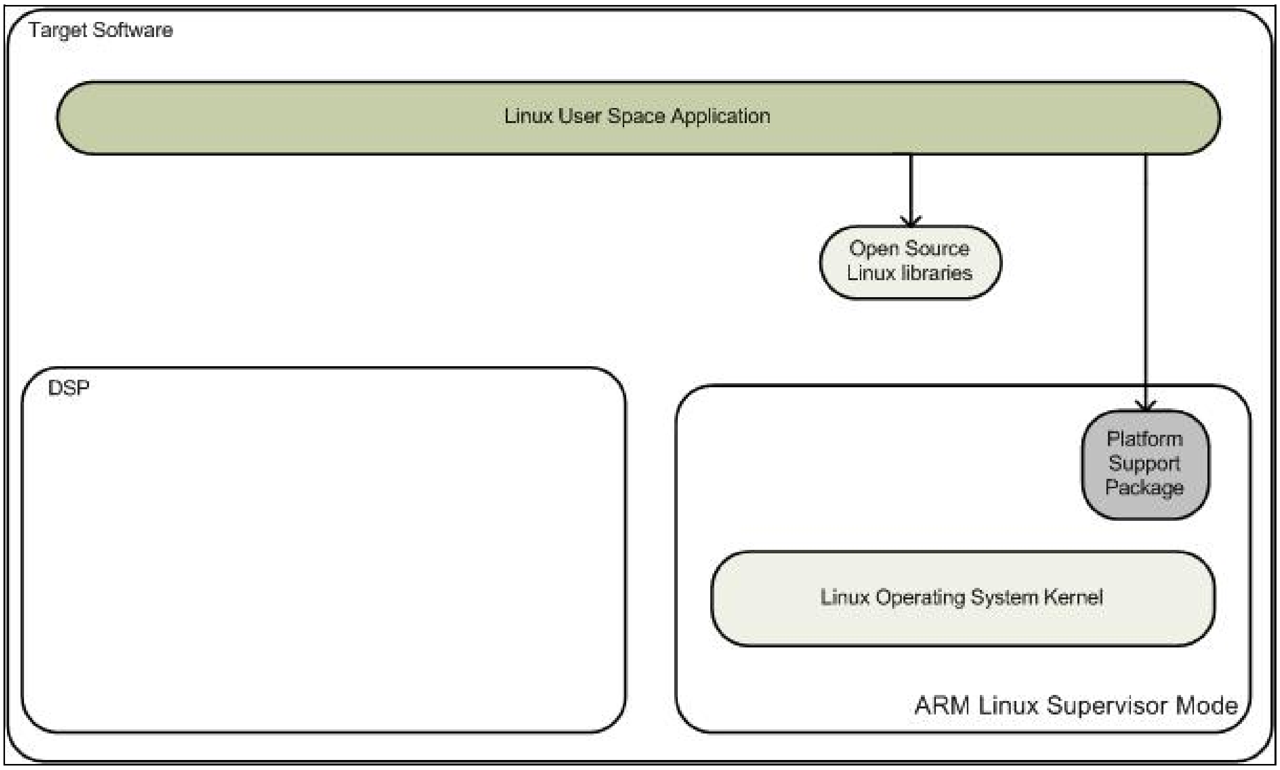
EZSDK software overview



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

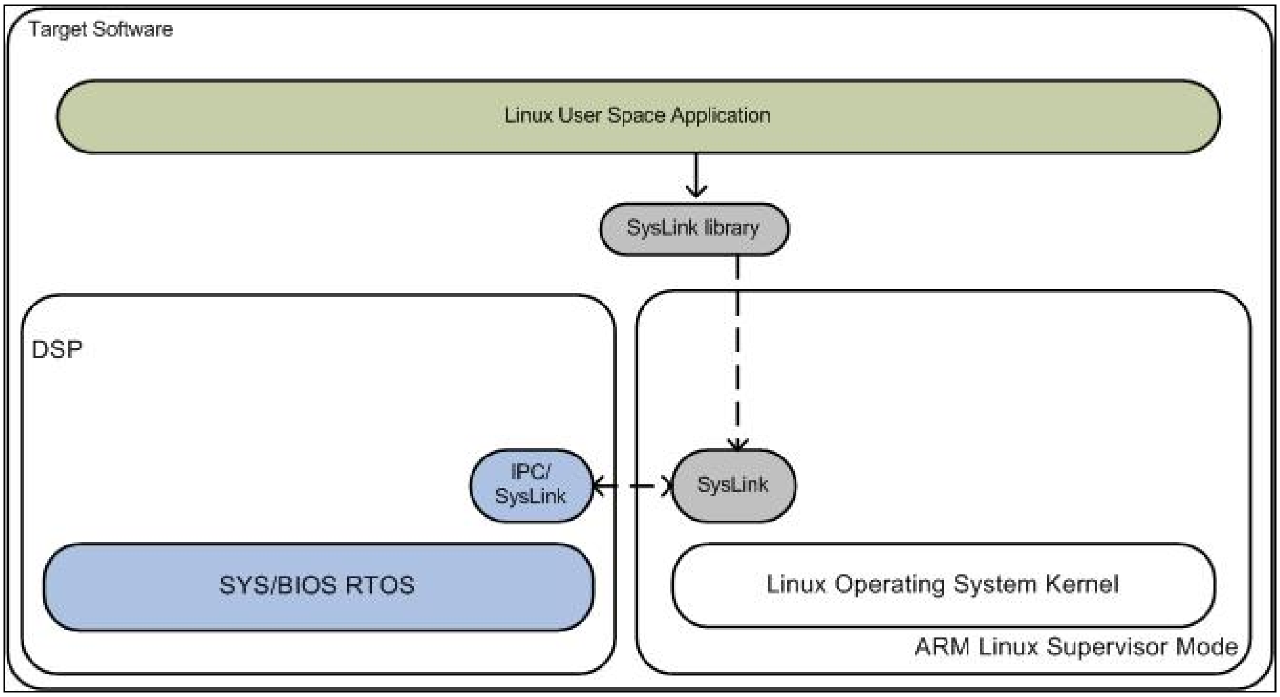
Overview of a basic Linux application component usage

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
CodeSourcery GCC toolchain	Cross compiler for generating ARM Linux binaries.	User specified location outside 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):

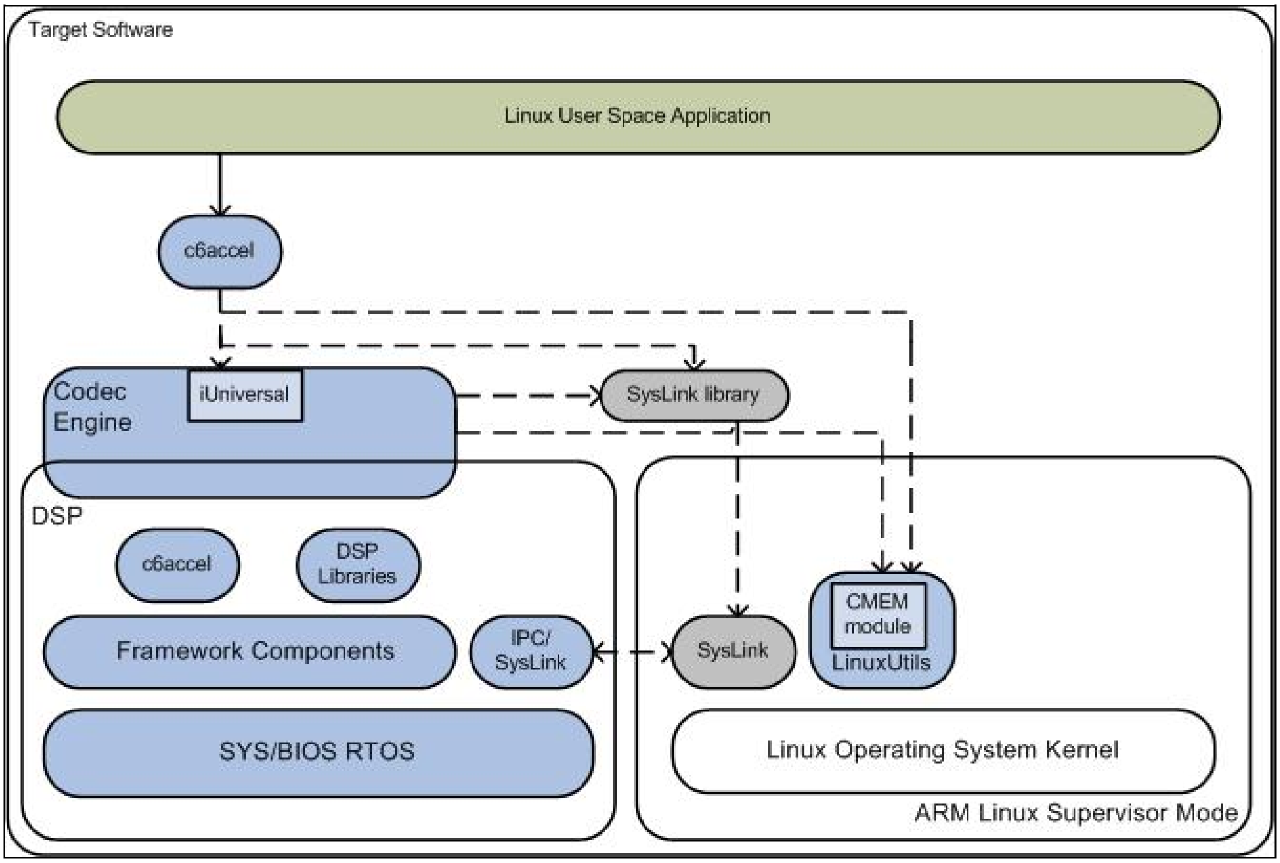
Component	Purpose in this application	Location in the EZSDK
SYS/BIOS	Real-Time Operation System for TI DSPs	component-sources/sysbios_x_xx_xx_xx
SysLink	HLOS to RTOS communication link for passing messages and data in	component-sources/syslink_x_xx_xx_xx

	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 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 EZSDK
Codec Engine	Cross platform framework for the applications invoking multimedia codecs and other algorithms.	component-sources/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).	component-sources/linuxutils_xx_xx_xx_xx
RTSC (XDC)	Tool used to configure Codec Engine, Framework Components and multimedia codecs for your application.	component-sources/xdctools_xx_xx_xx_xx
XDAIS	TI Algorithm Interface Standard used for algorithm standardization which is used by various other components including Codec Engine	component-sources/xdais_x_xx_xx_xx
SysLink		component-sources/syslink_x_xx_xx_xx

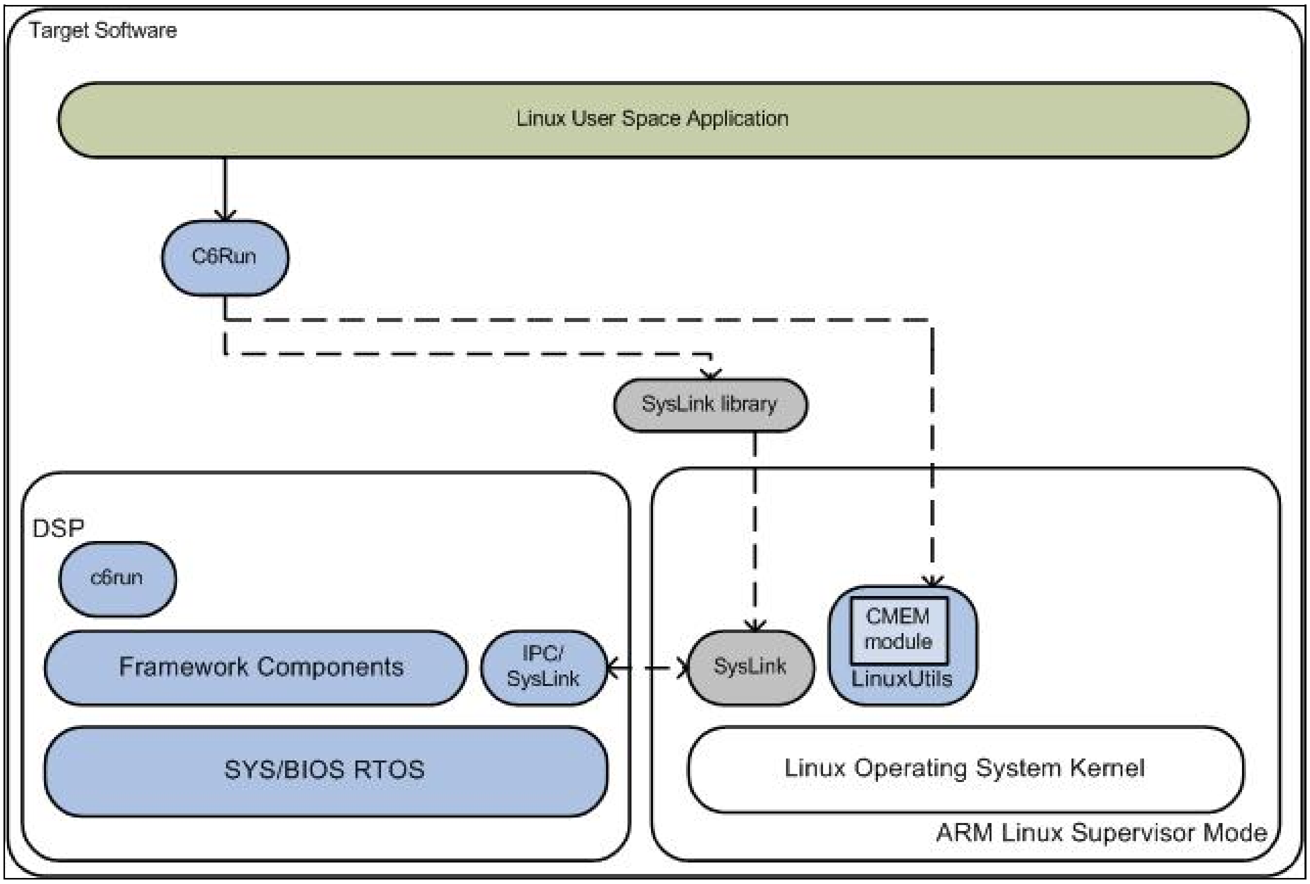
	HLOS to RTOS communication link for passing messages and data in multiprocessor systems	
IPC	RTOS commmunication link for passing messages and data	component-sources/ipc_x_xx_xx_xx

Good application examples to start from include:

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

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

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 EZSDK
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.	component-sources/linuxutils_xx_xx_xx_xx
RTSC (XDC)	Tool required to configure and build DSP/BIOS real-time kernel for the DSP.	component-sources/xdctools_xx_xx_xx_xx
Sys/BIOS	Real-Time Operation System for TI DSPs	component-sources/sysbios_x_xx_xx_xx
SysLink	HLOS to RTOS processor communication link for passing messages and data in multiprocessor systems	component-sources/syslink_x_xx_xx_xx
IPC	RTOS communication link for passing messages and data communication	component-sources/ipc_x_xx_xx_xx
C6000 Code Generation Tools	TI DSP code generation tools	dsp-devkit/cgt6x_x_x_xx

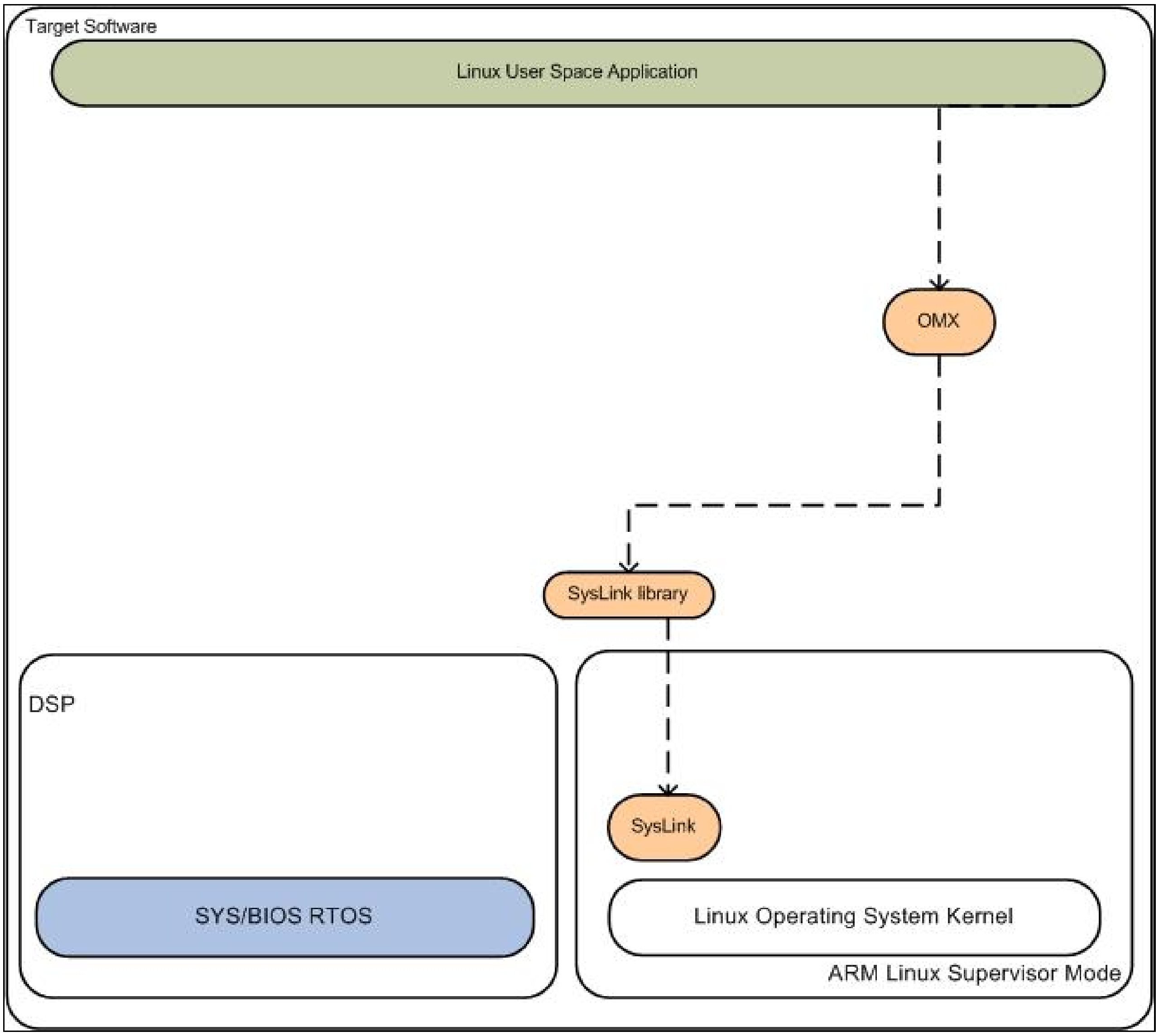
Good application examples to start from:

- The C6Run package contains sample applications to test/validate the functionality. The applications are located in the component-sources/c6run_xx_xx_xx_xx/examples and the component-sources/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 an OpenMax IL application


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

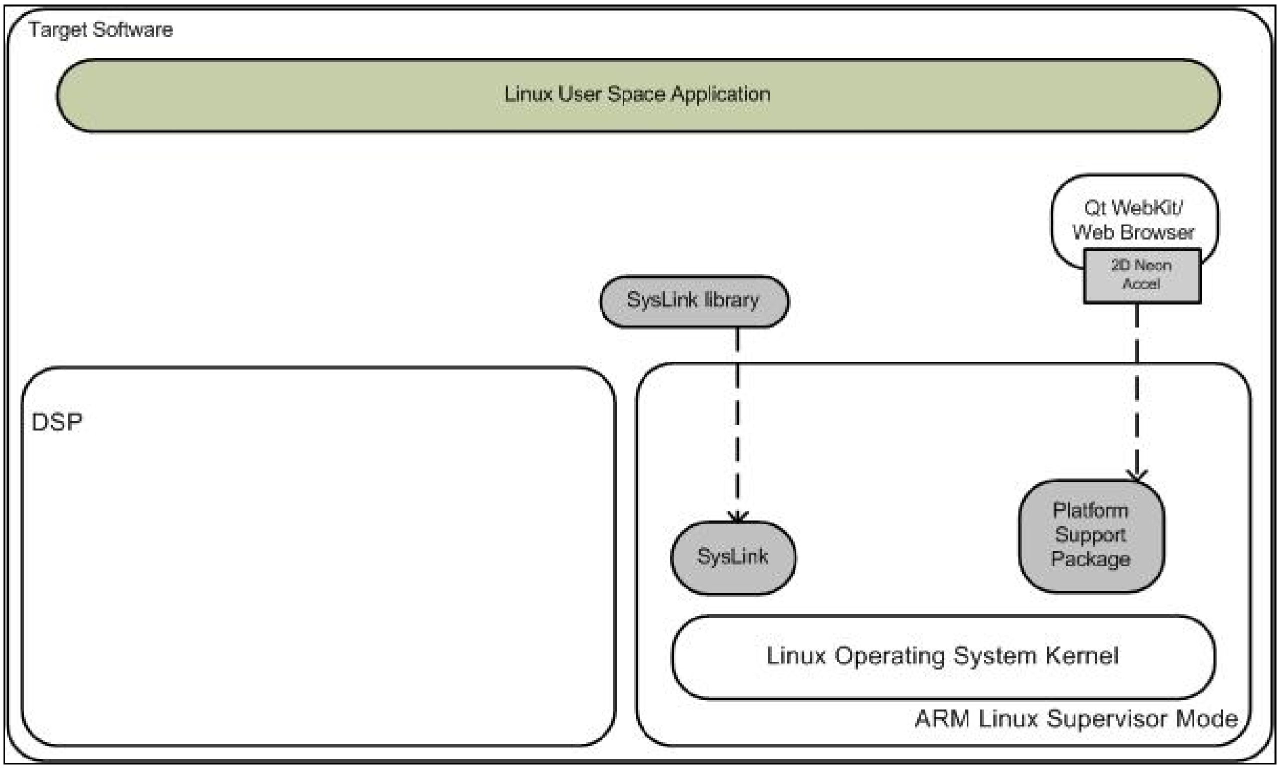
Overview of a basic OMX application component usage

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.

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.

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/dm814x/am387x/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/dm814x/am387x/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/dm814x/am387x/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 DM814x/AM387x 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 DM814x/AM387x EVM.

5. Set the SW1 switch to boot from SD.

- SW1 = 000001010111 (high to low, i.e. SW1.1 = 1)
- 1 = "On" position on the switch

6. Insert the SD card into the DM814x/AM387x 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_dm814x-evm_5_xx_xx_xx_xx_setuplinux setup_CCS_5.x.x.xxxxx.tar.gz
```

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

How to copy boot loaders to NAND flash

Please refer the U-boot documentation under the psp 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 resolution

The EZ SDK supports multiple displays resolutions but by default boots with 720p60 resolution. To change the resolution on your display, you can execute the following command. The command below demonstrates resolution change to 1080p60. 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 1080p60
```

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

How to change the display from LCD to HDMI

The EZ SDK supports multiple displays but by default displays on the LCD. To change the display to HDMI , you can execute the following command. In a similar manner, the display can be changed back to LCD.

```
target #  cd /usr/share/ti/ti-media-controller-utils
target # ./change_display.sh hdmi
```

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

FAQ

Frequently Asked Questions on The EZ SDK are available at EZ SDK FAQ. This information is also located in the *docs/* folder along with other documents.