Install Guide

syslink_02.00.00.56_alpha2

Contents

Articles

SysLink 02.00.00.56 alpha2 InstallGuide 1

SysLink 02.00.00.56 alpha2 InstallGuide Linux TI81XX 2

SysLink 02.00.00.56 alpha2 InstallGuide Linux OMAP3530 22

SysLink 02.00.00.56 alpha2 InstallGuide Linux OMAPL1XX 39

References

Article Sources and Contributors 56

Image Sources, Licenses and Contributors 57

SysLink 02.00.00.56 alpha2 InstallGuide

SYS/Link 02.00.00.56_alpha2 Install Guide

Install Guide

This release is intended to be used on:

- TI81XX
- OMAP3530
- OMAPL1XX

Introduction

The SysLink install guide provides information on how to install SysLink and its dependencies (where relevant). It also provides information on how to build SysLink, its dependencies (where relevant), and SysLink sample applications. Additionally, information is provided on how to run the sample applications provided with the SysLink product package.

Terms and Abbreviations

Abbreviation	Description
CCS	Code Composer Studio
IPC	Inter-Processor Communication
GPP	General Purpose Processor e.g. ARM
DSP	Digital Signal Processor e.g. C64X
CGTools	Code Gen Tools, e.g. Compiler, Linker, Archiver
SysLink	SYS/Link

This bullet indicates important information. Please read such text carefully.

Installation

Basic installation

SysLink is made available as a tar.gz file. To install the product follow the steps below:

- Unzip and untar the file syslink_<Version>_<ReleaseQualifier>.tar.gz.

Note:

- This document assumes the install path to be in the user home directory if working on a Linux PC. This path will be used in remainder of this document.
- If the installation was done at different location, make appropriate changes to the commands listed in the document.

It is advisable to archive the released sources in a configuration management system. This will help in merging:

- The updates delivered in the newer releases of SysLink.
- The changes to the product, if any, done by the users.

Dependencies

The dependencies, along with their download paths are given in the Release Notes.

IPC

Please follow the instructions given in <IPC_install>/ipc_<version>/docs/ User_install.pdf

Platform-specific Install Guides

Platform-specific install Guides for the SysLink product are available for all supported platforms:

- → TI81XX Linux
- → OMAP3530 Linux
- → OMAPL1XX Linux

SysLink 02.00.00.56 alpha2 InstallGuide Linux TI81XX

SYS/Link 02.00.00.56_alpha2 Linux TI81XX

SysLink Install Guide for TI81XX running Linux on the ARM Cortex A8.

Introduction

This document gives information about SysLink installation for using SysLink with the TI81XX platform, where the ARM Cortex A8 is running Linux OS. It also gives detailed build instructions for SysLink as well as the sample applications. In addition, instructions are also provided for running the sample applications provided within the SysLink release.

Dependencies

The dependencies, along with their versions are given in the Release Notes.

Installation

Setting up Linux Workstation

The description in this section is based on the following assumptions:

- The workstation is running on a Linux workstation
- Services telnetd, nfsd, ftpd are configured on this workstation.
- The workstation is assigned a fixed IP address.

A fixed IP address is preferred, as the IP address needs to be specified while booting the target board. This allows the boot loader configuration to be saved in flash.

Enable TFTP for downloading the kernel image to target

U-boot can be configured to download the kernel onto the target by various mechanisms:

- TFTP
- Serial Port

This section configures the Linux development host as a TFTP server. Modify the 'xinet.d/tftp' file to enable TFTP:

- Make the following changes:

```
disable     = no
```

server_args = -s /tftpboot

- Restart the network service

```
$ /etc/init.d/xinetd restart
```

The above configuration assumes that a directory 'tftpboot' has been created at the root '/' directory. The files in this directory are exposed through the TFTP protocol.

Building the Linux kernel

Kernel

Untar the kernel that comes with the Linux PSP TI81XX release. Build the kernel according to the PSP Linux User Guide document. Refer to the PSP u-boot related documentation on how to build the mkimage utility which is needed for creation of uImage.

U-Boot boot-loader

Please refer to the Linux PSP user guide to load the u.boot and x-loader to the target.

Create target file system

The target device needs a file system to boot from.

Code Composer Studio

Code composer studio can be used to connect to the DSP, run applications and debug the DSP side code

- Verified using CCS v4.2.07000
- Verified using CCS v5.0

Build

Editing files for Linux headers and tool chain paths

- Set SYSLINK_ROOT environment variable.This should be set to the base of the SysLink repository, i.e. the folder containing the `ti` directory.

```
$ export SYSLINK_ROOT=~/syslink_<VERSION>
```

- To build kernel side syslink
- Modify the KDIR variable in $SYSLINK_ROOT/ti/syslink/buildutils/hlos/knl/Makefile.inc to point to the built kernel source.

Replace the path in the Makefile.inc with the actual installation path. Alternatively, it can be overridden by passing to the make command.

```
......
ifeq ("$(SYSLINK_PLATFORM)", "TI81XX")
KDIR :=
/db/psp_git/syslink_toolchains/TI816X-LINUX-PSP-04.00.00.07/src/kernel/linux-04.00.00.07
endif # ifeq ("$(SYSLINK_PLATFORM)", "TI81XX")
......
```

Actual path in your build environment must be given for the KDIR path.

- Similarly change compile flag in the same file to point to ipc installation

```
......
COMPILE_FLAGS +=
-I/db/psp_git/syslink_toolchains/IPC/ipc_<version>/packages/
COMPILE_FLAGS += -I$(SYSLINK_ROOT)
.....
```

Replace ipc_<version> above with the actual version of the IPC.

- Alternatively, the paths can be passed to the make command as a ';' separated set of paths. For example:

```
$ make ARCH=arm
CROSS_COMPILE=/toolchains/git/arm-2009q1-203/bin/arm-none-linux-gnueabi-



SYSLINK_PKGPATH="/toolchains/IPC/ipc_<version>/packages;$HOME/syslink"
```

Type the above command in a single line

- To build user syslink
- Modify the TOOLCHAIN_PREFIX variable in $SYSLINK_ROOT/ti/syslink/buildutils/hlos/usr/Makefile.inc to point to arm toolchain. Replace the path in the Makefile.inc with the actual installation path.Alternatively, it can be overridden by passing to the make command.

```
......
ifeq ("$(SYSLINK_PLATFORM)", "TI81XX")
TOOLCHAIN_PREFIX :=
/db/psp_git/syslink_toolchains/arm-2009q1/bin/arm-none-linux-gnueabi-
endif # ifeq ("$(SYSLINK_PLATFORM)", "TI81XX")
......
```

- Similarly change compile flag in the same file to point to ipc installation

```
......
COMPILE_FLAGS +=
-I/db/psp_git/syslink_toolchains/IPC/ipc_<version>/packages/
COMPILE_FLAGS += -I$(SYSLINK_ROOT)
.....
```

Replace ipc_<version> above with the actual version of the IPC.

- Alternatively, the paths can be passed to the make command as a ';' separated set of paths. For example:

```
$ make
TOOLCHAIN_PREFIX=/db/psp_git/syslink_toolchains/arm-2009q1/bin/arm-none-linux-gnueabi-



SYSLINK_PKGPATH="/toolchains/IPC/ipc_<version>/packages;$HOME/syslink"
```

Type the above command in a single line

- Update CGTOOLS tool chain path in the following files:

```
$(SYSLINK_ROOT)/config.bld
```

For example:

```
var rootDirPre = "/toolchains/ti-tools/";
var rootDirPost = "";
C674.rootDir = rootDirPre + "c6000_7.2.0A10232" + rootDirPost;

* If build is required for only a specific configuration, comment the
other configurations in Build.targets

<pre>//list interested targets in Build.targets array
Build.targets = [
                    //C64P_COFF,
                    //C64P_ELF,
                    //C67P,
                    //C674_COFF,
                    C674_ELF,
                ];
```

Building SysLink HLOS driver/library and sample applications

The default configuration includes Ipc in the build for the drivers and the sample applications.

Switch to bash shell to build HLOS side drivers and samples. The default configuration builds for OMAP3530. To build for a different device, the SYSLINK_PLATFORM build parameter needs to be given.

Following are the steps to build the SysLink HLOS driver/library and sample applications for TI81XX:

- Set SYSLINK_ROOT environment variable

```
$ export SYSLINK_ROOT=~/syslink_<VERSION>/
```

- Build the SysLink kernel module

```
$ cd $SYSLINK_ROOT/ti/syslink/utils/hlos/knl/Linux
$ make ARCH=arm
CROSS_COMPILE=/toolchains/git/arm-2009q1-203/bin/arm-none-linux-gnueabi-
 SYSLINK_PLATFORM=TI81XX
```

- Build the SysLink user library

```
$ cd $SYSLINK_ROOT/ti/syslink/utils/hlos/usr/Linux
$ make
```

- Build SysLink kernel samples. Run make command in the respective sample directories.

```
$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/procMgr/knl/Linux
$ make ARCH=arm
CROSS_COMPILE=/toolchains/git/arm-2009q1-203/bin/arm-none-linux-gnueabi-
 SYSLINK_PLATFORM=TI81XX

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/notify/knl/Linux
$ make ARCH=arm
CROSS_COMPILE=/toolchains/git/arm-2009q1-203/bin/arm-none-linux-gnueabi-
 SYSLINK_PLATFORM=TI81XX

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/messageQ/knl/Linux
$ make ARCH=arm
CROSS_COMPILE=/toolchains/git/arm-2009q1-203/bin/arm-none-linux-gnueabi-
 SYSLINK_PLATFORM=TI81XX

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/heapBufMP/knl/Linux
$ make ARCH=arm
CROSS_COMPILE=/toolchains/git/arm-2009q1-203/bin/arm-none-linux-gnueabi-
 SYSLINK_PLATFORM=TI81XX

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/heapMemMP/knl/Linux
$ make ARCH=arm
CROSS_COMPILE=/toolchains/git/arm-2009q1-203/bin/arm-none-linux-gnueabi-
 SYSLINK_PLATFORM=TI81XX

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/frameq/knl/Linux
$ make ARCH=arm
```

```
CROSS_COMPILE=/toolchains/git/arm-2009q1-203/bin/arm-none-linux-gnueabi-
 SYSLINK_PLATFORM=TI81XX

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/listMP/knl/Linux
$ make ARCH=arm
CROSS_COMPILE=/toolchains/git/arm-2009q1-203/bin/arm-none-linux-gnueabi-
 SYSLINK_PLATFORM=TI81XX

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/ringIO/knl/Linux
$ make ARCH=arm
CROSS_COMPILE=/toolchains/git/arm-2009q1-203/bin/arm-none-linux-gnueabi-
 SYSLINK_PLATFORM=TI81XX

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/ringIO_gpp/knl/Linux
$ make ARCH=arm
CROSS_COMPILE=/toolchains/git/arm-2009q1-203/bin/arm-none-linux-gnueabi-
 SYSLINK_PLATFORM=TI81XX

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/sharedRegion/knl/Linux
$ make ARCH=arm
CROSS_COMPILE=/toolchains/git/arm-2009q1-203/bin/arm-none-linux-gnueabi-
 SYSLINK_PLATFORM=TI81XX

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/gateMP/knl/Linux
$ make ARCH=arm
CROSS_COMPILE=/toolchains/git/arm-2009q1-203/bin/arm-none-linux-gnueabi-
 SYSLINK_PLATFORM=TI81XX
```

- Build SysLink user samples. Run make command in the respective sample directories.

```
$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/procMgr/usr/Linux
$ make

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/notify/usr/Linux
$ make

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/messageQ/usr/Linux
$ make

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/listMP/usr/Linux
$ make

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/heapBufMP/usr/Linux
$ make

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/heapMemMP/usr/Linux
$ make
```

```
$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/frameq/usr/Linux
$ make SYSLINK_PLATFORM=TI81XX

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/ringIO/usr/Linux
$ make SYSLINK_PLATFORM=TI81XX

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/ringIO_gpp/usr/Linux
$ make

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/sharedRegion/usr/Linux
$ make SYSLINK_PLATFORM=TI81XX

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/gateMP/usr/Linux
$ make
```

Building RTOS SysLink and sample applications

- Set SYSLINK_ROOT environment variable.

```
$ export SYSLINK_ROOT=~/syslink_<VERSION>
```

- Set XDC in path.

```
$ export PATH=$PATH:/toolchains/xdc/xdctools_<version>
```

Alternatively, use the full path while making calls to xdc in the commands given later in this section

- Set XDC related environment variables.

```
$
export XDCPATH="/users/ipc/ipc_<version>/packages;/toolchains/bios6/bios_<version>/packag
```

Type the above command in a single line Replace <version> above with the specific version for IPC, BIOS & XDC tools

- Following statement needs to be added to all application config(.cfg) files

```
xdc.loadPackage ('ti.syslink.ipc.rtos');
```

this ensures that all required linker options are included by default
User needs to add the above mentioned statement to application config (.cfg) files in order to include all required linker options by default

- Build RTOS SysLink. This builds RTOS FrameQ and RingIO

```
$ cd $SYSLINK_ROOT/ti/syslink/
$ xdc all XDCBUILDCFG="$SYSLINK_ROOT/config.bld" -PR .
```

- Default build profie is "whole_program_debug" to change this to "debug" run the following

```
 $ xdc all XDCARGS="profile=debug"
XDCBUILDCFG="$SYSLINK_ROOT/config.bld" -PR .
```

alternatively "XDCARGS" can be exported

```
$ export XDCARGS=profile=debug
$ xdc all XDCBUILDCFG="$SYSLINK_ROOT/config.bld" -PR .
```

- If you want to exclude Samples and build rest of the libraries, use following command

```
xdc  XDCBUILDCFG="$SYSLINK_ROOT/config.bld" -PR 'xdcpkg syslink | grep
-v /samples'
```

The step to build SysLink RTOS side also builds all sample applications. If the syslink/ipc and sample applications are to be built independently, this can be done through separate commands as given below.

- To optimize the build time by utilizing "parellel build feature of GNU make" pass command line option **--jobs** as shown below

```
$ xdc all --jobs=4 XDCBUILDCFG="$SYSLINK_ROOT/config.bld" -PR .
```

--jobs= <number of seperate jobs to be run> //usually 1.5 times no of cpu cores available on the build system

- To build only the sample applications independently, use the following commands:

```
$ cd $SYSLINK_ROOT/ti/syslink/ipc
$ xdc all -PR .

$ cd $SYSLINK_ROOT/ti/syslink/samples/rtos/platforms
$ xdc all -PR .

$ cd $SYSLINK_ROOT/ti/syslink/samples/rtos/notify
$ xdc all

$ cd $SYSLINK_ROOT/ti/syslink/samples/rtos/messageQ
$ xdc all

$ cd $SYSLINK_ROOT/ti/syslink/samples/rtos/frameq
$ xdc all

$ cd $SYSLINK_ROOT/ti/syslink/samples/rtos/heapBufMP
$ xdc all

$ cd $SYSLINK_ROOT/ti/syslink/samples/rtos/heapMemMP
$ xdc all

$ cd $SYSLINK_ROOT/ti/syslink/samples/rtos/listMP
$ xdc all

$ cd $SYSLINK_ROOT/ti/syslink/samples/rtos/ringIO
$ xdc all

$ cd $SYSLINK_ROOT/ti/syslink/samples/rtos/ringIO_gpp
$ xdc all

$ cd $SYSLINK_ROOT/ti/syslink/samples/rtos/sharedRegion
$ xdc all

$ cd $SYSLINK_ROOT/ti/syslink/samples/rtos/gateMP
```

```
$ xdc all
```

Please note that ringIO_gpp sample code does not exist for TI81XX in this release, hence executables will not be generated

Build Notes

The default build configuration for SysLink and sample applications assumes OMAP3530. If no specific SYSLINK_PLATFORM value is specified, OMAP3530 is assumed by default.

TI81XX supports the following loader configurations:

- DSP: ELF
- DSP: COFF

ELF is the default loader format for the slave executables,to change to COFF format user needs to pass **SYSLINK_LOADER=COFF** compile time option when building SysLink HLOS kernel

```
$ make ARCH=arm
CROSS_COMPILE=/toolchains/git/arm-2009q1-203/bin/arm-none-linux-gnueabi-

 SYSLINK_PLATFORM=TI81XX SYSLINK_LOADER=COFF
```

For additional information on different build parameters, please refer to the UserGuide document.

Configuring Kernel Parameters

SysLink requires a few specific arguments to be passed to the Linux kernel during boot up. For running the sample applications, 3MB of memory is used by SysLink for communication between GPP and DSP, and for DSP external memory for placing its code/data. This must be reserved by specifying 3MB less as available for the Linux kernel for its usage.

- For example,with available memory 256M, memory required for shared regions/other utils 5M and syslink 3M,

```
bootargs console=ttyS2,115200n8 root=/dev/nfs
nfsroot=HOST:nfs_root,nolock rw mem=248M ip=dhcp
```

This is just an example, bootargs may vary depending on available setup
Depending on the memory map used for the final system configuration, the memory to be reserved from Linux may be different.

Running the sample applications

Sample applications are provided with SysLink for the supported platforms. This section describes the way to execute the sample applications.

The steps for execution of the samples are given below for execution with Linux running on the GPP.

For all kernel-side applications, ProcMgrApp is expected to be run before running the specific application to load and start the slave processor.

If the following command line arguments are provided, then the user-side application also loads/starts/stops the slave processor in the application context itself. In this case, ProcMgrApp application is not required to be run before running the specific application:

```
Arguments: <procId1> <Path including name of the slave
executable>
```

Copying files to target file system

The generated binaries on the HLOS side and RTOS side must be copied to the target directory.

For executing the sample applications, follow the steps below to copy the relevant binaries:

- Copy the syslink.ko kernel module and the application kernel modules into the target file system

```
$ cp ${SYSLINK_ROOT}/ti/syslink/lib/modules/TI81XX/*.ko
${HOME}/ti81xx/target/opt/syslink/.
```

- Copy the user samples into the target file system

```
$ cp ${SYSLINK_ROOT}/ti/syslink/lib/samples/*.exe
$HOME}/ti81xx/target/opt/syslink/
```

- Copy RTOS binaries into the target file system
- To copy the DSP executables use the following steps:

```
$cd ${SYSLINK_ROOT}/ti/syslink/samples/rtos
```

Each copy command should be typed in a single line

```
$ cp
notify/ti_syslink_samples_rtos_platforms_ti816x_dsp/debug/notify_ti81xx_dsp.xe674
  ${HOME}/ti81xx/target/opt/syslink/
$ cp
messageQ/ti_syslink_samples_rtos_platforms_ti816x_dsp/debug/messageq_ti81xx_dsp.xe674
  ${HOME}/ti81xx/target/opt/syslink/
$ cp
frameq/ti_syslink_samples_rtos_platforms_ti816x_dsp/debug/frameq_ti81xx_dsp.xe674

  ${HOME}/ti81xx/target/opt/syslink/
$ cp
listMP/ti_syslink_samples_rtos_platforms_ti816x_dsp/debug/listmp_ti81xx_dsp.xe674

  ${HOME}/ti81xx/target/opt/syslink/
$ cp
heapBufMP/ti_syslink_samples_rtos_platforms_ti816x_dsp/debug/heapbufmp_ti81xx_dsp.xe674

  ${HOME}/ti81xx/target/opt/syslink/
$ cp
heapMemMP/ti_syslink_samples_rtos_platforms_ti816x_dsp/debug/heapmemmp_ti81xx_dsp.xe674

  ${HOME}/ti81xx/target/opt/syslink/
$ cp
ringIO/ti_syslink_samples_rtos_platforms_ti816x_dsp/debug/ringio_ti81xx_dsp.xe674

  ${HOME}/ti81xx/target/opt/syslink/
$ cp
ringIO_gpp/ti_syslink_samples_rtos_platforms_ti816x_dsp/debug/ringiogpp_ti81xx_dsp.xe674
  ${HOME}/ti81xx/target/opt/syslink/
$ cp
```

```
sharedRegion/ti_syslink_samples_rtos_platforms_ti816x_dsp/debug/sharedregion_ti81xx_dsp.x

  ${HOME}/ti81xx/target/opt/syslink/
$ cp
gateMP/ti_syslink_samples_rtos_platforms_ti816x_dsp/debug/gatemp_ti81xx_dsp.xe674
  ${HOME}/ti81xx/target/opt/syslink/
```

Running applications

In the below execute instructions, note that the DSP executables have extension .xe674 since they are built for ELF mode. COFF format produces extensions .x674.

Loading syslink kernel module

- The syslink kernel module must be inserted before running any sample applications.

```
$ insmod syslink.ko TRACE=1 TRACEFAILURE=1
```

Enabling TRACEFAILURE puts out prints in case any failure occurs during SysLink initialization.TRACECLASS can be enabled to see more detailed prints,refer to UerGuide for more on using TRACECLASS.

Unloading syslink kernel module

```
$ rmmod syslink
```

Running the ProcMgr sample application

ProcMgr user-side sample application

- To invoke the application enter the following commands:
- Execute the following to load all slave processors. You can replace path and executables as per your setup:

procmgrapp.exe takes two parameters remote procId and executable name.

```
$ procmgrapp.exe 0 /opt/syslink/messageq_ti81xx_dsp.xe674
```

- Press `Ctrl Z` to make it a background process
- Run the sample application by inserting its kernel module. For example, messageqapp

```
$ insmod messageqapp.ko
```

- For clean up:

```
$ rmmod messageqapp
$ fg
```

- Press 'enter' to exit and cleanup.
- The user-side procmgrapp example can also be used to act as a user-space loader while running the applications. To use it, press `Ctrl Z` after the following print is seen on the terminal:

```
Press any key to continue and perform shutdown ...
```

- Use `fg` to bring the procmgrapp.exe application back to the foreground. Press ‘enter’ to run it to completion and stop the slave.

Running the Notify sample application

Notify kernel-side sample application

- To invoke the application enter the following commands:
  - Load HOST ARM with uImage and uboot
  - Execute the following to load the slave processor(DSP). You can replace path and executable as per your setup:

```
$ procmgrapp.exe 0 /opt/syslink/notify_ti81xx_dsp.xe674
```

- Press `Ctrl Z` to make it a background process
- Run the notifyapp sample application by inserting its kernel module

```
$ insmod notifyapp.ko
```

- For clean up:

```
$ rmmod notifyapp
$ fg
```

- Press 'enter' to exit and cleanup.

Notify user-side sample application

- Load HOST ARM with uImage and uboot
- To invoke the application standalone, enter the following commands:

```
$ ./notifyapp.exe 0 /opt/syslink/notify_ti81xx_dsp.xe674
```

- Press 'enter' to exit and cleanup.

Alternatively:

- To invoke the application with ProcMgrApp, enter the following commands:
  - Execute the following to load the slave processor(DSP). You can replace path and executable as per your setup:

```
$ procmgrapp.exe 0 /opt/syslink/notify_ti81xx_dsp.xe674
```

- Press `Ctrl Z` to make it a background process
- Run the notifyapp sample application

```
$ ./notifyapp.exe
```

- For clean up:

```
$ fg
```

- Press 'enter' to exit and cleanup.

Running the MessageQ sample application

MessageQ kernel-side sample application

- To invoke the application enter the following commands:
  - Load HOST ARM with uImage and uboot
  - Execute the following to load the slave processor(DSP). You can replace path and executable as per your setup:

```
$ procmgrapp.exe 0 /opt/syslink/messageq_ti81xx_dsp.xe674
```

- Press `Ctrl Z` to make it a background process
- Run the messageqapp sample application by inserting its kernel module

```
$ insmod messageqapp.ko
```

- For clean up:

```
$ rmmod messageqapp
$ fg
```

- Press 'enter' to exit and cleanup.

MessageQ user-side sample application

- Load HOST ARM with uImage and uboot
- To invoke the application standalone, enter the following commands:

```
$ ./messageqapp.exe 0 /opt/syslink/messageq_ti81xx_dsp.xe674
```

Alternatively:

- To invoke the application with ProcMgrApp, enter the following commands:
  - Execute the following to load the slave processor(DSP). You can replace path and executable as per your setup:

```
$ procmgrapp.exe 0 /opt/syslink/messageq_ti81xx_dsp.xe674
```

- Press `Ctrl Z` to make it a background process
- Run the messageqapp sample application

```
$ ./messageqapp.exe
```

- For clean up:

```
$ fg
```

- Press 'enter' to exit and cleanup.

Running the ListMP sample application

ListMP kernel-side sample application

- To invoke the application enter the following commands:
  - Load HOST ARM with uImage and uboot
  - Execute the following to load the slave processor(DSP). You can replace path and executable as per your setup:

```
$ procmgrapp.exe 0 /opt/syslink/listmp_ti81xx_dsp.xe674
```

- Press `Ctrl Z` to make it a background process
- Run the listmpapp sample application by inserting its kernel module

```
$ insmod listmpapp.ko
```

- For clean up:

```
$ rmmod listmpapp
$ fg
```

- Press 'enter' to exit and cleanup.

ListMP user-side sample application

- Load HOST ARM with uImage and uboot
- To invoke the application standalone, enter the following commands:

```
$ ./listmpapp.exe 0 /opt/syslink/listmp_ti81xx_dsp.xe674
```

Alternatively:

- To invoke the application with ProcMgrApp, enter the following commands:
  - Execute the following to load the slave processor(DSP). You can replace path and executable as per your setup:

```
$ procmgrapp.exe 0 /opt/syslink/listmp_ti81xx_dsp.xe674
```

- Press `Ctrl Z` to make it a background process
- Run the listmpapp sample application

```
$ ./listmpapp.exe
```

- For clean up:

```
$ fg
```

- Press 'enter' to exit and cleanup.

Running the HeapBufMP sample application

HeapBufMP kernel-side sample application

- To invoke the application enter the following commands:
  - Load HOST ARM with uImage and uboot
  - Execute the following to load the slave processor(DSP). You can replace path and executable as per your setup:

```
$ procmgrapp.exe 0 /opt/syslink/heapbufmp_ti81xx_dsp.xe674
```

- Press `Ctrl Z` to make it a background process

```
$ insmod heapbufmpapp.ko
```

- For clean up:

```
$ rmmod heapbufmpapp
$ fg
```

- Press 'enter' to exit and cleanup.

HeapBufMP user-side sample application

- Load HOST ARM with uImage and uboot
- To invoke the application standalone, enter the following commands:

```
$ ./heapbufmpapp.exe 0 /opt/syslink/heapbufmp_ti81xx_dsp.xe674
```

Alternatively:

- To invoke the application with ProcMgrApp, enter the following commands:
  - Execute the following to load the slave processor(DSP). You can replace path and executable as per your setup:

```
$ procmgrapp.exe 0 /opt/syslink/heapbufmp_ti81xx_dsp.xe674
```

- Press `Ctrl Z` to make it a background process
- Run the heapbufmpapp sample application

```
$ ./heapbufmpapp.exe
```

- For clean up:

```
$ fg
```

- Press 'enter' to exit and cleanup.

Running the HeapMemMP sample application

HeapMemMP kernel-side sample application

- To invoke the application enter the following commands:
  - Load HOST ARM with uImage and uboot
  - Execute the following to load the slave processor(DSP). You can replace path and executable as per your setup:

```
$ procmgrapp.exe 0 /opt/syslink/heapmemmp_ti81xx_dsp.xe674
```

- Press `Ctrl Z` to make it a background process
- Run the heapmemmpapp sample application by inserting its kernel module

```
$ insmod heapmemmpapp.ko
```

- For clean up:

```
$ rmmod heapmemmpapp
$ fg
```

- Press 'enter' to exit and cleanup.

HeapMemMP user-side sample application

- Load HOST ARM with uImage and uboot
- To invoke the application standalone, enter the following commands:

```
$ ./heapmemmpapp.exe 0 /opt/syslink/heapmemmp_ti81xx_dsp.xe674
```

Alternatively:

- To invoke the application with ProcMgrApp, enter the following commands:
  - Execute the following to load the slave processor(DSP). You can replace path and executable as per your setup:

```
$ procmgrapp.exe 0 /opt/syslink/heapmemmp_ti81xx_dsp.xe674
```

- Press `Ctrl Z` to make it a background process
- Run the heapmemmpapp sample application

```
$ ./heapmemmpapp.exe
```

- For clean up:

```
$ fg
```

- Press 'enter' to exit and cleanup.

Running the FrameQ sample application

FrameQ kernel-side sample application

- To invoke the application enter the following commands:
  - Load HOST ARM with uImage and uboot
  - Execute the following to load the slave processor(DSP). You can replace path and executable as per your setup:

```
$ procmgrapp.exe 0 /opt/syslink/frameq_ti81xx_dsp.xe674
```

- Press `Ctrl Z` to make it a background process
- Run the frameqapp sample application by inserting its kernel module

```
$ insmod frameqapp.ko
```

- For clean up:

```
$ rmmod frameqapp
$ fg
```

- Press 'enter' to exit and cleanup.

FrameQ user-side sample application

- Load HOST ARM with uImage and uboot
- To invoke the application standalone, enter the following commands:

```
$ ./frameqapp.exe 0 /opt/syslink/frameq_ti81xx_dsp.xe674
```

Alternatively:

- To invoke the application with ProcMgrApp, enter the following commands:
  - Execute the following to load the slave processor(DSP). You can replace path and executable as per your setup:

```
$ procmgrapp.exe 0 /opt/syslink/frameq_ti81xx_dsp.xe674
```

- Press `Ctrl Z` to make it a background process
- Run the frameqapp sample application

```
$ ./frameqapp.exe
```

- For clean up:

```
$ fg
```

- Press 'enter' to exit and cleanup.

Running the RingIO sample application

RingIO kernel-side sample application

- To invoke the application enter the following commands:
  - Load HOST ARM with uImage and uboot
  - Execute the following to load the slave processor(DSP). You can replace path and executable as per your setup:

```
$ procmgrapp.exe 0 /opt/syslink/ringio_ti81xx_dsp.xe674
```

- Press `Ctrl Z` to make it a background process
- Run the ringioapp sample application by inserting its kernel module

```
$ insmod ringioapp.ko
```

- For clean up:

```
$ rmmod ringioapp
$ fg
```

- Press 'enter' to exit and cleanup.

RingIO user-side sample application

- Load HOST ARM with uImage and uboot
- To invoke the application standalone, enter the following commands:

```
$ ./ringioapp.exe 0 /opt/syslink/ringio_ti81xx_dsp.xe674
```

Alternatively:

- To invoke the application with ProcMgrApp, enter the following commands:
  - Execute the following to load the slave processor(DSP). You can replace path and executable as per your setup:

```
$ procmgrapp.exe 0 /opt/syslink/ringio_ti81xx_dsp.xe674
```

- Press `Ctrl Z` to make it a background process
- Run the ringioapp sample application

```
$ ./ringioapp.exe
```

- For clean up:

```
$ fg
```

- Press 'enter' to exit and cleanup.

Running the SharedRegion sample application

SharedRegion kernel-side sample application

- To invoke the application enter the following commands:
  - Load HOST ARM with uImage and uboot
  - Execute the following to load the slave processor(DSP). You can replace path and executable as per your setup:

```
$ procmgrapp.exe 0 /opt/syslink/sharedregion_ti81xx_dsp.xe674
```

- Press `Ctrl Z` to make it a background process
- Run the sharedregionapp sample application by inserting its kernel module

```
$ insmod sharedregionapp.ko
```

- For clean up:

```
$ rmmod sharedregionapp
$ fg
```

- Press 'enter' to exit and cleanup.

SharedRegion user-side sample application

- Load HOST ARM with uImage and uboot
- To invoke the application standalone, enter the following commands:

```
$ ./sharedregionapp.exe 0 /opt/syslink/sharedregion_ti81xx_dsp.xe674
```

Alternatively:

- To invoke the application with ProcMgrApp, enter the following commands:
  - Execute the following to load the slave processor(DSP). You can replace path and executable as per your setup:

```
$ procmgrapp.exe 0 /opt/syslink/sharedregion_ti81xx_dsp.xe674
```

- Press `Ctrl Z` to make it a background process
- Run the sharedregionapp sample application

```
$ ./sharedregionapp.exe
```

- For clean up:

```
$ fg
```

- Press 'enter' to exit and cleanup.

Running the GateMP sample application

GateMP kernel-side sample application

- To invoke the application enter the following commands:
  - Load HOST ARM with uImage and uboot
  - Execute the following to load the slave processor(DSP). You can replace path and executable as per your setup:

```
$ procmgrapp.exe 0 /opt/syslink/gatemp_ti81xx_dsp.xe674
```

- Press `Ctrl Z` to make it a background process
- Run the gatempapp sample application by inserting its kernel module

```
$ insmod gatempapp.ko
```

- For clean up:

```
$ rmmod gatempapp
$ fg
```

- Press 'enter' to exit and cleanup.

GateMP user-side sample application

- Load HOST ARM with uImage and uboot
- To invoke the application standalone, enter the following commands:

```
$ ./gatempapp.exe 0 /opt/syslink/gatemp_ti81xx_dsp.xe674
```

Alternatively:

- To invoke the application with ProcMgrApp, enter the following commands:
  - Execute the following to load the slave processor(DSP). You can replace path and executable as per your setup:

```
$ procmgrapp.exe 0 /opt/syslink/gatemp_ti81xx_dsp.xe674
```

- Press `Ctrl Z` to make it a background process
- Run the gatempapp sample application

```
$ ./gatempapp.exe
```

- For clean up:

```
$ fg
```

- Press 'enter' to exit and cleanup.

SysLink 02.00.00.56 alpha2 InstallGuide Linux OMAP3530

SYS/Link 02.00.00.56_alpha2 Linux OMAP3530

SysLink Install Guide for OMAP3530 running Linux on the ARM Cortex A8

Introduction

This document gives information about SysLink installation for using SysLink with the OMAP3530 platform, where the ARM Cortex A8 is running Linux OS. It also gives detailed build instructions for SysLink as well as the sample applications. In addition, instructions are also provided for running the sample applications provided within the SysLink release.

Dependencies

The dependencies, along with their versions are given in the Release Notes.

Installation

Setting up Linux Workstation

The description in this section is based on the following assumptions:

- The workstation is running on a Linux workstation
- Services telnetd, nfsd, ftpd are configured on this workstation.
- The workstation is assigned a fixed IP address.

A fixed IP address is preferred, as the IP address needs to be specified while booting the target board. This allows the boot loader configuration to be saved in flash.

Enable TFTP for downloading the kernel image to target

U-boot can be configured to download the kernel onto the target by various mechanisms:

- TFTP
- Serial Port

This section configures the Linux development host as a TFTP server. Modify the ‘xinet.d/tftp’ file to enable TFTP:

- Make the following changes:

```
disable     = no
```

server_args = -s /tftpboot

- Restart the network service

```
$ /etc/init.d/xinetd restart
```

The above configuration assumes that a directory 'tftpboot' has been created at the root '/' directory. The files in this directory are exposed through the TFTP protocol.

Building the Linux kernel

This section assumes that Linux release is installed on the development workstation at `/toolchains/git`.

Tool chain

Untar the arm-2009q1-203 tool chain and add the tool chain directory in your path.

```
$ tar -xjvf
arm-2009q1-203-arm-none-linux-gnueabi-i686-pc-linux-gnu.tar.bz2.tar
$ export PATH=$PATH:/toolchains/git/arm-2009q1-203/bin
```

Kernel

Untar the kernel that comes with the Linux release. Build the kernel according to the PSP Linux User Guide document. Refer to the PSP u-boot related documentation on how to build the mkimage utility which is needed for creation of uImage. Along with the tool chain, location of mkimage utility should also be there in the path.

```
$ tar -zxvf AM35x-OMAP35x-PSP-SDK-03.00.00.03.tgz
$ cd AM35x-OMAP35x-PSP-SDK-03.00.00.03/src/kernel
$ tar -zxvf linux-03.00.00.03.tar.gz
```

Patching the kernel

Download a kernel patch required (see Dependencies) to enable iommu for iva2 subsystem and apply it on the kernel as shown below:

```
#Copy the patch to kernel source directory
$ cp enable-iommu-for-iva2.patch
AM35x-OMAP35x-PSP-SDK-03.00.00.03/src/kernel/linux-03.00.00.03
$ cd AM35x-OMAP35x-PSP-SDK-03.00.00.03/src/kernel/linux-03.00.00.03
$ patch -p0 < enable-iommu-for-iva2.patch
```

Ensure that this patch is applied before creating uImage. Since syslink uses iommu internally, it is mandatory to have this patch applied.

U-Boot boot-loader

Please refer the Linux PSP user guide to load the u.boot and x-loader to the target.

Create target file system

The target device needs a file system to boot from. The file system can be exported to the target through NFS.

Exporting target file system through NFS

- A directory on the development host can be setup and exported for this purpose.
- AM35x-OMAP35x-PSP-SDK-03.00.00.03/images/fs/omap3530 contains the NFS file system nfs.tar.gz.

```
$ tar -xzvf nfs.tar.gz
```

You need to be 'root' to successfully execute this command.

The file system can be copied to a different location. In such a case ~/omap3530/target can be a soft link to the actual location.

- libpthread libraries required for SysLink may not be available by default within the target file system. Copy this from the tool-chain.

```
$ cp
/toolchains/git/arm-2009q1-203/arm-none-linux-gnueabi/libc/lib/libpthread*
 ~/omap3530/target/lib
```

This step is not required if libpthread libraries are available by default in the target file system.

- The directory `~/omap3530/target` will be mounted as root directory on the target through NFS.

To do so, add the following line to the file /etc/exports.

```
/home/<user>/omap3530/target  *(rw,no_root_squash)
```

Replace "/home/<user>" in the path above with the actual path of your home directory on the development workstation.

Code Composer Studio

Code composer studio can be used to connect DSP, run applications and debug the DSP side code

- Verified using CCS v4.2.07000
- Verified using CCS v5.0, refer to CCS v5.0 documentation on how to debug linux side code using CCS v5.0

CCS can attach to only ARM in the beginning. It can attach to the DSP only after the ARM-side application releases DSP from reset.

Build

Editing files for Linux headers and tool chain paths

- Set SYSLINK_ROOT environment variable.This should be set to the base of the SysLink repository, i.e. the folder containing the `ti` directory.

```
$ export SYSLINK_ROOT=~/syslink_<VERSION>
```

- Update KDIR variable in the base Makefile with the Linux source path in the following files. Alternatively, it can be overridden by passing to the make command.

```
$(SYSLINK_ROOT)/ti/syslink/buildutils/hlos/knl/Makefile.inc
KDIR :=
/toolchains/omap3530/AM35x-OMAP35x-PSP-SDK-03.00.00.03/src/kernel/linux-03.00.00.03
```

Actual path in your build environment must be given for the KDIR path.

- Update TOOLCHAIN_PREFIX variable in the base Makefile with the Linux tool chain in the following files. Alternatively, it can be overridden by passing to the make command.

```
$(SYSLINK_ROOT)/ti/syslink/buildutils/hlos/usr/Makefile.inc
TOOLCHAIN_PREFIX :=
/toolchains/git/arm-2009q1-203/bin/arm-none-linux-gnueabi-
```

- The path to the SysLink and IPC product installation packages need to be given as include paths for the SysLink and application builds. For this, the paths can be set in the Makefile.inc.

```
COMPILE_FLAGS += -I/toolchains/IPC/ipc_<version>/packages/
COMPILE_FLAGS += -I$(SYSLINK_ROOT)
```

Replace ipc_<version> above with the actual version of the IPC.

- Alternatively, the paths can be passed to the make command as a ';' separated set of paths. For example:

```
$ make ARCH=arm
CROSS_COMPILE=/toolchains/git/arm-2009q1-203/bin/arm-none-linux-gnueabi-

SYSLINK_PKGPATH="/toolchains/IPC/ipc_<version>/packages;$HOME/syslink"
```

Type the above command in a single line Replace ipc_<version> above with the actual version of the IPC.

- Update CGTOOLS tool chain path in the following files:

```
$(SYSLINK_ROOT)/config.bld
```

For example:

```
var rootDirPre = "/toolchains/ti-tools/";
var rootDirPost = "";
C64P.rootDir = rootDirPre + "c6000_7.2.0A10232" + rootDirPost;
```

- If build is required for only a specific configuration, comment the other configurations in Build.targets

```
//list interested targets in Build.targets array
Build.targets = [
                    C64P_COFF,
                    //C64P_ELF,
                    //C67P,
                    //C674,
                ];
```

Building SysLink HLOS driver/library and sample applications

The default configuration includes Ipc in the build for the drivers and the sample applications.

Switch to bash shell to build HLOS side drivers and samples. The default configuration builds for OMAP3530. To build for a different device, the SYSLINK_PLATFORM build parameter needs to be given.

Following are the steps to build the SysLink HLOS driver/library and sample applications in the default configuration:

- Set SYSLINK_ROOT environment variable.This should be set to the base of the SysLink repository, i.e. the folder containing the `ti` directory.

```
$ export SYSLINK_ROOT=~/syslink_<VERSION>
```

- Build the SysLink kernel module

```
$ cd $SYSLINK_ROOT/ti/syslink/utils/hlos/knl/Linux
$ make ARCH=arm
CROSS_COMPILE=/toolchains/git/arm-2009q1-203/bin/arm-none-linux-gnueabi-
```

- Build the SysLink user library

```
$ cd $SYSLINK_ROOT/ti/syslink/utils/hlos/usr/Linux
$ make
```

- Build SysLink kernel samples. Run make command in the respective sample directories.

```
$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/procMgr/knl/Linux
$ make ARCH=arm
CROSS_COMPILE=/toolchains/git/arm-2009q1-203/bin/arm-none-linux-gnueabi-

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/notify/knl/Linux
$ make ARCH=arm
CROSS_COMPILE=/toolchains/git/arm-2009q1-203/bin/arm-none-linux-gnueabi-

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/messageQ/knl/Linux
$ make ARCH=arm
CROSS_COMPILE=/toolchains/git/arm-2009q1-203/bin/arm-none-linux-gnueabi-

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/heapBufMP/knl/Linux
$ make ARCH=arm
CROSS_COMPILE=/toolchains/git/arm-2009q1-203/bin/arm-none-linux-gnueabi-

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/heapMemMP/knl/Linux
$ make ARCH=arm
CROSS_COMPILE=/toolchains/git/arm-2009q1-203/bin/arm-none-linux-gnueabi-

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/frameq/knl/Linux
$ make ARCH=arm
CROSS_COMPILE=/toolchains/git/arm-2009q1-203/bin/arm-none-linux-gnueabi-

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/listMP/knl/Linux
$ make ARCH=arm
CROSS_COMPILE=/toolchains/git/arm-2009q1-203/bin/arm-none-linux-gnueabi-

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/ringIO/knl/Linux
$ make ARCH=arm
CROSS_COMPILE=/toolchains/git/arm-2009q1-203/bin/arm-none-linux-gnueabi-

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/ringIO_gpp/knl/Linux
$ make ARCH=arm
CROSS_COMPILE=/toolchains/git/arm-2009q1-203/bin/arm-none-linux-gnueabi-

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/sharedRegion/knl/Linux
$ make ARCH=arm
CROSS_COMPILE=/toolchains/git/arm-2009q1-203/bin/arm-none-linux-gnueabi-

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/gateMP/knl/Linux
$ make ARCH=arm
CROSS_COMPILE=/toolchains/git/arm-2009q1-203/bin/arm-none-linux-gnueabi-
```

- Build SysLink user samples. Run make command in the respective sample directories.

```
$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/procMgr/usr/Linux
$ make
```

```
$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/notify/usr/Linux
$ make

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/messageQ/usr/Linux
$ make

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/listMP/usr/Linux
$ make

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/heapBufMP/usr/Linux
$ make

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/heapMemMP/usr/Linux
$ make

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/frameq/usr/Linux
$ make SYSLINK_PLATFORM=OMAP3530

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/ringIO/usr/Linux
$ make SYSLINK_PLATFORM=OMAP3530

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/ringIO_gpp/usr/Linux
$ make

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/sharedRegion/usr/Linux
$ make SYSLINK_PLATFORM=OMAP3530

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/gateMP/usr/Linux
$ make
```

Building RTOS SysLink and sample applications

- Set SYSLINK_ROOT environment variable.This should be set to the base of the SysLink repository, i.e. the folder containing the `ti` directory.

```
$ export SYSLINK_ROOT=~/syslink_<VERSION>
```

- Set XDC in path.

```
$ export PATH=$PATH:/toolchains/xdc/xdctools_<version>
```

Alternatively, use the full path while making calls to xdc in the commands given later in this section

- Set XDC related environment variables.

```
$ export
XDCPATH="/users/ipc/ipc_<version>/packages;/toolchains/bios6/bios_<version>/packages"
```

Replace <version> above with the specific version for IPC, BIOS & XDC tools

- Following statement needs to be added to all application config(.cfg) files

```
xdc.loadPackage ('ti.syslink.ipc.rtos');
```

this ensures that all required linker options are included by default
User needs to add the above mentioned statement to application config (.cfg) files in order to include all required linker options by default

- Build RTOS SysLink. This builds RTOS FrameQ and RingIO

```
$ cd $SYSLINK_ROOT/ti/syslink/
$ xdc all XDCBUILDCFG="$SYSLINK_ROOT/config.bld" -PR .
```

- Default build profie is "whole_program_debug" to change this to "debug" run the following

```
 $ xdc all XDCARGS="profile=debug"
XDCBUILDCFG="$SYSLINK_ROOT/config.bld" -PR .
```

alternatively "XDCARGS" can be exported

```
export XDCARGS=profile=debug
```

```
$ xdc all XDCBUILDCFG="$SYSLINK_ROOT/config.bld" -PR .
```

- If you want to exclude Samples and build rest of the libraries, use following command

```
 xdc  XDCBUILDCFG="$SYSLINK_ROOT/config.bld" -PR 'xdcpkg syslink | grep
 -v /samples'
```

The step to build SysLink RTOS side also builds all sample applications. If the syslink/ipc and sample applications are to be built independently, this can be done through separate commands as given below.

- To build only the sample applications independently, use the following commands:

```
$ cd $SYSLINK_ROOT/ti/syslink/ipc
$ xdc all -PR .

$ cd $SYSLINK_ROOT/ti/syslink/samples/rtos/platforms
$ xdc all -PR .

$ cd $SYSLINK_ROOT/ti/syslink/samples/rtos/notify
$ xdc all

$ cd $SYSLINK_ROOT/ti/syslink/samples/rtos/messageQ
$ xdc all

$ cd $SYSLINK_ROOT/ti/syslink/samples/rtos/frameq
$ xdc all

$ cd $SYSLINK_ROOT/ti/syslink/samples/rtos/heapBufMP
$ xdc all

$ cd $SYSLINK_ROOT/ti/syslink/samples/rtos/heapMemMP
$ xdc all

$ cd $SYSLINK_ROOT/ti/syslink/samples/rtos/listMP
```

```
$ xdc all

$ cd $SYSLINK_ROOT/ti/syslink/samples/rtos/ringIO
$ xdc all

$ cd $SYSLINK_ROOT/ti/syslink/samples/rtos/ringIO_gpp
$ xdc all

$ cd $SYSLINK_ROOT/ti/syslink/samples/rtos/sharedRegion
$ xdc all

$ cd $SYSLINK_ROOT/ti/syslink/samples/rtos/gateMP
$ xdc all
```

Build Notes

The default build configuration for SysLink and sample applications assumes OMAP3530. If no specific SYSLINK_PLATFORM value is specified, OMAP3530 is assumed by default.

OMAP3530 supports both COFF & ELF build for the slave executables. The choice of which loader is to be used can be done through a build flag SYSLINK_LOADER, which can be passed to the make command for the syslink kernel module. If no flag is specified, the default is COFF. If SYSLINK_LOADER=ELF is used, the ELF loader is used.

For additional information on different build parameters, please refer to the UserGuide document.

Configuring Kernel Parameters

SysLink requires a few specific arguments to be passed to the Linux kernel during boot up. For running the sample applications, 3MB of memory is used by SysLink for communication between GPP and DSP, and for DSP external memory for placing its code/data. This must be reserved by specifying 3MB less as available for the Linux kernel for its usage.

Depending on the memory map used for the final system configuration, the memory to be reserved from Linux may be different.

Running the sample applications

Sample applications are provided with SysLink for the supported platforms. This section describes the way to execute the sample applications.

The steps for execution of the samples are given below for execution with Linux running on the GPP.

For all kernel-side applications, ProcMgrApp is expected to be run before running the specific application to load and start the slave processor.

For all user-side applications, if no command line arguments are provided while running the sample application, the ProcMgrApp application is expected to be run before running the specific application to load and start the slave processor.

If the following command line arguments are provided, then the user-side application also loads/starts/stops the slave processor in the application context itself. In this case, ProcMgrApp application is not required to be run before running the specific application:

```
Arguments: <procId1> <Path including name of the slave
executable>
```

Copying files to target file system

The generated binaries on the HLOS side and RTOS side must be copied to the target directory.

When built for COFF mode, the generated executables have an extension `.x64P`. ELF mode executables have an extension `.xe64P`. In the below instructions, copy instructions are given for COFF mode RTOS executables. To copy ELF mode executables, replace the `.x64P` extensions with `.xe64P` below.

For executing the sample applications, follow the steps below to copy the relevant binaries:

- Copy the syslink.ko kernel module and the application kernel modules into the target file system

```
$ cp ${SYSLINK_ROOT}/ti/syslink/lib/modules/OMAP3530/*.ko
${HOME}/omap3530/target/opt/syslink/.
```

- Copy the user samples into the target file system

```
$ cp ${SYSLINK_ROOT}/ti/syslink/lib/samples/*.exe
$HOME}/omap3530/target/opt/syslink/
```

To copy RTOS binaries, use the following steps:

- Copy the rtos executables into the target file system

```
$ cp
${SYSLINK_ROOT}/ti/syslink/samples/rtos/notify/evm3530_dsp/debug/notify_omap3530_dsp.x64P
 ${HOME}/omap3530/target/opt/syslink/
$ cp
${SYSLINK_ROOT}/ti/syslink/samples/rtos/messageQ/evm3530_dsp/debug/messageq_omap3530_dsp.
 ${HOME}/omap3530/target/opt/syslink/
$ cp
${SYSLINK_ROOT}/ti/syslink/samples/rtos/frameq/evm3530_dsp/debug/frameq_omap3530_dsp.x64P
 ${HOME}/omap3530/target/opt/syslink/
$ cp
${SYSLINK_ROOT}/ti/syslink/samples/rtos/listMP/evm3530_dsp/debug/listmp_omap3530_dsp.x64P
 ${HOME}/omap3530/target/opt/syslink/
$ cp
${SYSLINK_ROOT}/ti/syslink/samples/rtos/heapBufMP/evm3530_dsp/debug/heapbufmp_omap3530_ds
 ${HOME}/omap3530/target/opt/syslink/
$ cp
${SYSLINK_ROOT}/ti/syslink/samples/rtos/heapMemMP/evm3530_dsp/debug/heapmemmp_omap3530_ds
 ${HOME}/omap3530/target/opt/syslink/
$ cp
${SYSLINK_ROOT}/ti/syslink/samples/rtos/ringIO/evm3530_dsp/debug/ringio_omap3530_dsp.x64P
 ${HOME}/omap3530/target/opt/syslink/
$ cp
${SYSLINK_ROOT}/ti/syslink/samples/rtos/ringIO_gpp/evm3530_dsp/debug/ringiogpp_omap3530_d
 ${HOME}/omap3530/target/opt/syslink/
$ cp
${SYSLINK_ROOT}/ti/syslink/samples/rtos/sharedRegion/evm3530_dsp/debug/sharedregion_omap3
 ${HOME}/omap3530/target/opt/syslink/
```

```
$ cp
${SYSLINK_ROOT}/ti/syslink/samples/rtos/gateMP/evm3530_dsp/debug/gatemp_omap3530_dsp.x64P
 ${HOME}/omap3530/target/opt/syslink/
```

Running applications

When built for COFF mode, the generated executables have an extension `.x64P`. ELF mode executables have an extension `.xe64P`. In the below instructions, copy instructions are given for COFF mode RTOS executables. To copy ELF mode executables, replace the `.x64P` extensions with `.xe64P` below.

Loading syslink kernel module

- The syslink kernel module must be inserted before running any sample applications.

```
$ insmod syslink.ko TRACE=1 TRACEFAILURE=1
```

Enabling TRACEFAILURE puts out prints in case any failure occurs during SysLink initialization.

Unloading syslink kernel module

```
$ rmmod syslink
```

Running the ProcMgr sample application

ProcMgr user-side sample application

- To invoke the application enter the following commands:

```
$ procmgrapp.exe 0 /opt/syslink/notify_omap3530_dsp.x64P
```

procmgrapp.exe takes two parameters remote procId and executable name.

- Run procMgr user sample to load and start DSP:

```
$ procmgrapp.exe 0 /opt/syslink/notify_omap3530_dsp.x64P
```

- The user-side procmgrapp example can also be used to act as a user-space loader while running the applications. To use it, press `Ctrl Z` after the following print is seen on the terminal:

```
Press any key to continue and perform shutdown ...
```

- Now run the specific application. For example:

```
$ ./notifyapp.exe
```

- Use `fg` to bring the procmgrapp.exe application back to the foreground. Press ‘enter’ to run it to completion and stop the slave.

Running the Notify sample application

Notify kernel-side sample application

- To invoke the application enter the following commands:

```
$ procmgrapp.exe 0 /opt/syslink/notify_omap3530_dsp.x64P
```

- Press `Ctrl Z` to make it a background process
- Run the notifyapp sample application by inserting its kernel module

```
$ insmod notifyapp.ko
$ rmmod notifyapp
$ fg
```

- Press 'enter' to exit and cleanup.

Notify user-side sample application

- To invoke the application standalone, enter the following commands:

```
$ ./notifyapp.exe 0 /opt/syslink/notify_omap3530_dsp.x64P
```

- Press 'enter' to exit and cleanup.

Alternatively:

- To invoke the application with ProcMgrApp, enter the following commands:

```
$ procmgrapp.exe 0 /opt/syslink/notify_omap3530_dsp.x64P
```

- Press `Ctrl Z` to make it a background process
- Run the notifyapp sample application

```
$ ./notifyapp.exe
$ fg
```

- Press 'enter' to exit and cleanup.

Running the MessageQ sample application

MessageQ kernel-side sample application

- To invoke the application enter the following commands:

```
$ procmgrapp.exe 0 /opt/syslink/messageq_omap3530_dsp.x64P
```

- Press `Ctrl Z` to make it a background process
- Run the messageqapp sample application by inserting its kernel module

```
$ insmod messageqapp.ko
$ rmmod messageqapp
$ fg
```

- Press 'enter' to exit and cleanup.

MessageQ user-side sample application

- To invoke the application standalone, enter the following commands:

```
$ ./messageqapp.exe 0 /opt/syslink/messageq_omap3530_dsp.x64P
```

Alternatively:

- To invoke the application with ProcMgrApp, enter the following commands:

```
$ procmgrapp.exe 0 /opt/syslink/messageq_omap3530_dsp.x64P
```

- Press `Ctrl Z` to make it a background process
- Run the messageqapp sample application

```
$ ./messageqapp.exe
$ fg
```

- Press 'enter' to exit and cleanup.

Running the ListMP sample application

ListMP kernel-side sample application

- To invoke the application enter the following commands:

```
$ procmgrapp.exe 0 /opt/syslink/listmp_omap3530_dsp.x64P
```

- Press `Ctrl Z` to make it a background process
- Run the listmpapp sample application by inserting its kernel module

```
$ insmod listmpapp.ko
$ rmmod listmpapp
$ fg
```

- Press 'enter' to exit and cleanup.

ListMP user-side sample application

- To invoke the application standalone, enter the following commands:

```
$ ./listmpapp.exe 0 /opt/syslink/listmp_omap3530_dsp.x64P
```

Alternatively:

- To invoke the application with ProcMgrApp, enter the following commands:

```
$ procmgrapp.exe 0 /opt/syslink/listmp_omap3530_dsp.x64P
```

- Press `Ctrl Z` to make it a background process
- Run the listmpapp sample application

```
$ ./listmpapp.exe
$ fg
```

- Press 'enter' to exit and cleanup.

Running the HeapBufMP sample application

HeapBufMP kernel-side sample application

- To invoke the application enter the following commands:

```
$ procmgrapp.exe 0 /opt/syslink/heapbufmp_omap3530_dsp.x64P
```

- Press `Ctrl Z` to make it a background process
- Run the heapbufmpapp sample application by inserting its kernel module

```
$ insmod heapbufmpapp.ko
$ rmmod heapbufmpapp
$ fg
```

- Press 'enter' to exit and cleanup.

HeapBufMP user-side sample application

- To invoke the application standalone, enter the following commands:

```
$ ./heapbufmpapp.exe 0 /opt/syslink/heapbufmp_omap3530_dsp.x64P
```

Alternatively:

- To invoke the application with ProcMgrApp, enter the following commands:

```
$ procmgrapp.exe 0 /opt/syslink/heapbufmp_omap3530_dsp.x64P
```

- Press `Ctrl Z` to make it a background process
- Run the heapbufmpapp sample application

```
$ ./heapbufmpapp.exe
$ fg
```

- Press 'enter' to exit and cleanup.

Running the HeapMemMP sample application

HeapMemMP kernel-side sample application

- To invoke the application enter the following commands:

```
$ procmgrapp.exe 0 /opt/syslink/heapmemmp_omap3530_dsp.x64P
```

- Press `Ctrl Z` to make it a background process
- Run the heapmemmpapp sample application by inserting its kernel module

```
$ insmod heapmemmpapp.ko
$ rmmod heapmemmpapp
$ fg
```

- Press 'enter' to exit and cleanup.

HeapMemMP user-side sample application

- To invoke the application standalone, enter the following commands:

```
$ ./heapmemmpapp.exe 0 /opt/syslink/heapmemmp_omap3530_dsp.x64P
```

Alternatively:

- To invoke the application with ProcMgrApp, enter the following commands:

```
$ procmgrapp.exe 0 /opt/syslink/heapmemmp_omap3530_dsp.x64P
```

- Press `Ctrl Z` to make it a background process
- Run the heapmemmpapp sample application

```
$ ./heapmemmpapp.exe
$ fg
```

- Press 'enter' to exit and cleanup.

Running the FrameQ sample application

FrameQ kernel-side sample application

- To invoke the application enter the following commands:

```
$ procmgrapp.exe 0 /opt/syslink/frameq_omap3530_dsp.x64P
```

- Press `Ctrl Z` to make it a background process
- Run the frameqapp sample application by inserting its kernel module

```
$ insmod frameqapp.ko
$ rmmod frameqapp
$ fg
```

- Press 'enter' to exit and cleanup.

FrameQ user-side sample application

- To invoke the application standalone, enter the following commands:

```
$ ./frameqapp.exe 0 /opt/syslink/frameq_omap3530_dsp.x64P
```

Alternatively:

- To invoke the application with ProcMgrApp, enter the following commands:

```
$ procmgrapp.exe 0 /opt/syslink/frameq_omap3530_dsp.x64P
```

- Press `Ctrl Z` to make it a background process
- Run the frameqapp sample application

```
$ ./frameqapp.exe
$ fg
```

- Press 'enter' to exit and cleanup.

Running the RingIO sample application

RingIO kernel-side sample application

- To invoke the application enter the following commands:

```
$ procmgrapp.exe 0 /opt/syslink/ringio_omap3530_dsp.x64P
```

- Press `Ctrl Z` to make it a background process
- Run the ringioapp sample application by inserting its kernel module

```
$ insmod ringioapp.ko
$ rmmod ringioapp
$ fg
```

- Press 'enter' to exit and cleanup.

RingIO user-side sample application

- To invoke the application standalone, enter the following commands:

```
$ ./ringioapp.exe 0 /opt/syslink/ringio_omap3530_dsp.x64P
```

Alternatively:

- To invoke the application with ProcMgrApp, enter the following commands:

```
$ procmgrapp.exe 0 /opt/syslink/ringio_omap3530_dsp.x64P
```

- Press `Ctrl Z` to make it a background process
- Run the ringioapp sample application

```
$ ./ringioapp.exe
$ fg
```

- Press 'enter' to exit and cleanup.

Running the RingIO GPP sample application

RingIO GPP kernel-side sample application

- To invoke the application enter the following commands:

```
$ procmgrapp.exe 0 /opt/syslink/ringiogpp_omap3530_dsp.x64P
```

- Press `Ctrl Z` to make it a background process
- Run the ringiogppapp sample application by inserting its kernel module

```
$ insmod ringiogppapp.ko
$ rmmod ringiogppapp
$ fg
```

- Press 'enter' to exit and cleanup.

RingIO GPP user-side sample application

- To invoke the application standalone, enter the following commands:

```
$ ./ringiogppapp.exe 0 /opt/syslink/ringiogpp_omap3530_dsp.x64P
```

Alternatively:

- To invoke the application with ProcMgrApp, enter the following commands:

```
$ procmgrapp.exe 0 /opt/syslink/ringiogpp_omap3530_dsp.x64P
```

- Press `Ctrl Z` to make it a background process
- Run the ringiogppapp sample application

```
$ ./ringiogppapp.exe
$ fg
```

- Press 'enter' to exit and cleanup.

Running the SharedRegion sample application

SharedRegion kernel-side sample application

- To invoke the application enter the following commands:

```
$ procmgrapp.exe 0 /opt/syslink/sharedregion_omap3530_dsp.x64P
```

- Press `Ctrl Z` to make it a background process
- Run the sharedregionapp sample application by inserting its kernel module

```
$ insmod sharedregionapp.ko
$ rmmod sharedregionapp
$ fg
```

- Press 'enter' to exit and cleanup.

SharedRegion user-side sample application

- To invoke the application standalone, enter the following commands:

```
$ ./sharedregionapp.exe 0 /opt/syslink/sharedregion_omap3530_dsp.x64P
```

Alternatively:

- To invoke the application with ProcMgrApp, enter the following commands:

```
$ procmgrapp.exe 0 /opt/syslink/sharedregion_omap3530_dsp.x64P
```

- Press `Ctrl Z` to make it a background process
- Run the sharedregionapp sample application

```
$ ./sharedregionapp.exe
$ fg
```

- Press 'enter' to exit and cleanup.

Running the GateMP sample application

GateMP kernel-side sample application

- To invoke the application enter the following commands:

```
$ procmgrapp.exe 0 /opt/syslink/gatemp_omap3530_dsp.x64P
```

- Press `Ctrl Z` to make it a background process
- Run the gatempapp sample application by inserting its kernel module

```
$ insmod gatempapp.ko
$ rmmod gatempapp
$ fg
```

- Press 'enter' to exit and cleanup.

GateMP user-side sample application

- To invoke the application standalone, enter the following commands:

```
$ ./gatempapp.exe 0 /opt/syslink/gatemp_omap3530_dsp.x64P
```

Alternatively:

- To invoke the application with ProcMgrApp, enter the following commands:

```
$ procmgrapp.exe 0 /opt/syslink/gatemp_omap3530_dsp.x64P
```

- Press `Ctrl Z` to make it a background process
- Run the gatempapp sample application

```
$ ./gatempapp.exe
$ fg
```

- Press 'enter' to exit and cleanup.

SysLink 02.00.00.56 alpha2 InstallGuide Linux OMAPL1XX

SYS/Link 02.00.00.56_alpha2 Linux OMAPL1XX

SysLink Install Guide for OMAPL1XX running Linux on the ARM.

Introduction

This document gives information about SysLink installation for using SysLink with the OMAPL1XX platform, where the ARM is running Linux OS. It also gives detailed build instructions for SysLink as well as the sample applications. In addition, instructions are also provided for running the sample applications provided within the SysLink release.

Dependencies

The dependencies, along with their versions are given in the Release Notes.

Installation

Setting up Linux Workstation

The description in this section is based on the following assumptions:

- The workstation is running on a Linux workstation
- Services telnetd, nfsd, ftpd are configured on this workstation.
- The workstation is assigned a fixed IP address.

A fixed IP address is preferred, as the IP address needs to be specified while booting the target board. This allows the boot loader configuration to be saved in flash.

Enable TFTP for downloading the kernel image to target

U-boot can be configured to download the kernel onto the target by various mechanisms:

- TFTP
- Serial Port

This section configures the Linux development host as a TFTP server. Modify the 'xinet.d/tftp' file to enable TFTP:

- Make the following changes:

```
disable     = no
```

server_args = -s /tftpboot

- Restart the network service

```
$ /etc/init.d/xinetd restart
```

The above configuration assumes that a directory 'tftpboot' has been created at the root '/' directory. The files in this directory are exposed through the TFTP protocol.

Building the Linux kernel

This section assumes that Linux release is installed on the development workstation at `/toolchains/git`.

Tool chain

Untar the arm-2009q1-203 tool chain and add the tool chain directory in your path.

```
$ tar -xjvf
arm-2009q1-203-arm-none-linux-gnueabi-i686-pc-linux-gnu.tar.bz2.tar
$ export PATH=$PATH:/toolchains/git/arm-2009q1-203/bin
```

Kernel

Untar the kernel that comes with the Linux release. Build the kernel according to the PSP Linux User Guide document. Refer to the PSP u-boot related documentation on how to build the mkimage utility which is needed for creation of uImage. Along with the tool chain, location of mkimage utility should also be there in the path.

U-Boot boot-loader

Please refer to the Linux PSP user guide to load the u.boot and x-loader to the target.

Create target file system

The target device needs a file system to boot from. The file system can be exported to the target through NFS.

Exporting target file system through NFS

- A directory on the development host can be setup and exported for this purpose.
- AM35x-OMAP35x-PSP-SDK-03.00.00.03/images/fs/omapl1xx contains the NFS file system nfs.tar.gz.

```
$ tar -xzvf nfs.tar.gz
```

You need to be 'root' to successfully execute this command.

The file system can be copied to a different location. In such a case ~/omapl1xx/target can be a soft link to the actual location.

- libpthread libraries required for SysLink may not be available by default within the target file system. Copy this from the tool-chain.

```
$ cp
/toolchains/git/arm-2009q1-203/arm-none-linux-gnueabi/libc/lib/libpthread*
 ~/omapl1xx/target/lib
```

This step is not required if libpthread libraries are available by default in the target file system.

- The directory `~/omapl1xx/target` will be mounted as root directory on the target through NFS.

To do so, add the following line to the file /etc/exports.

```
/home/<user>/omapl1xx/target  *(rw,no_root_squash)
```

Replace "/home/<user>" in the path above with the actual path of your home directory on the development workstation.

Code Composer Studio

Code composer studio can be used to connect DSP, run applications and debug the DSP side code

- Verified using CCS v4.2.07000
- Verified using CCS v5.0, refer to CCS v5.0 documentation on how to debug linux side code using CCS v5.0

CCS can attach to only ARM in the beginning. It can attach to the DSP only after the ARM-side application releases DSP from reset.

Build

Editing files for Linux headers and tool chain paths

- Set SYSLINK_ROOT environment variable.This should be set to the base of the SysLink repository, i.e. the folder containing the `ti` directory.

```
$ export SYSLINK_ROOT=~/syslink_<VERSION>
```

- Update KDIR variable in the base Makefile with the Linux source path in the following files. Alternatively, it can be overridden by passing to the make command.

```
$(SYSLINK_ROOT)/ti/syslink/buildutils/hlos/knl/Makefile.inc
KDIR :=
/toolchains/omapl1xx/DaVinci-PSP-SDK-03.20.00.11/src/kernel/linux-03.20.00.11
```

Actual path in your build environment must be given for the KDIR path.

- Update TOOLCHAIN_PREFIX variable in the base Makefile with the Linux tool chain in the following files. Alternatively, it can be overridden by passing to the make command.

```
$(SYSLINK_ROOT)/ti/syslink/buildutils/hlos/usr/Makefile.inc
TOOLCHAIN_PREFIX :=
/toolchains/git/arm-2009q1-203/bin/arm-none-linux-gnueabi-
```

- The path to the SysLink and IPC product installation packages need to be given as include paths for the SysLink and application builds. For this, the paths can be set in the Makefile.inc.

```
COMPILE_FLAGS += -I/toolchains/IPC/ipc_<version>/packages/
COMPILE_FLAGS += -I$(SYSLINK_ROOT)
```

Replace ipc_<version> above with the actual version of the IPC.

- Alternatively, the paths can be passed to the make command as a ';' separated set of paths. For example:

```
$ make ARCH=arm
CROSS_COMPILE=/toolchains/git/arm-2009q1-203/bin/arm-none-linux-gnueabi-

SYSLINK_PKGPATH="/toolchains/IPC/ipc_<version>/packages;$HOME/syslink"
```

Type the above command in a single line

- Update CGTOOLS tool chain path in the following files:

```
$(SYSLINK_ROOT)/config.bld
```

For example:

```
var rootDirPre = "/toolchains/ti-tools/";
var rootDirPost = "";
C674.rootDir = rootDirPre + "c6000_7.2.0A10232" + rootDirPost;
```

- If build is required for only a specific configuration, comment the other configurations in Build.targets

```
//list interested targets in Build.targets array
Build.targets = [
                    //C64P_COFF,
                    //C64P_ELF,
                    //C67P,
                    //C674_COFF,
                    C674_ELF,
                ];
```

Building SysLink HLOS driver/library and sample applications

The default configuration includes Ipc in the build for the drivers and the sample applications.

Switch to bash shell to build HLOS side drivers and samples. The default configuration builds for OMAP3530. To build for a different device, the SYSLINK_PLATFORM build parameter needs to be given.

Following are the steps to build the SysLink HLOS driver/library and sample applications for OMAPl1XX:

- Set SYSLINK_ROOT environment variable.This should be set to the base of the SysLink repository, i.e. the folder containing the `ti` directory.

```
$ export SYSLINK_ROOT=~/syslink_<VERSION>
```

- Build the SysLink kernel module

```
$ cd $SYSLINK_ROOT/ti/syslink/utils/hlos/knl/Linux
$ make ARCH=arm
CROSS_COMPILE=/toolchains/git/arm-2009q1-203/bin/arm-none-linux-gnueabi-
 SYSLINK_PLATFORM=OMAPL1XX
```

- Build the SysLink user library

```
$ cd $SYSLINK_ROOT/ti/syslink/utils/hlos/usr/Linux
$ make
```

- Build SysLink kernel samples. Run make command in the respective sample directories.

```
$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/procMgr/knl/Linux
$ make ARCH=arm
CROSS_COMPILE=/toolchains/git/arm-2009q1-203/bin/arm-none-linux-gnueabi-
 SYSLINK_PLATFORM=OMAPL1XX

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/notify/knl/Linux
$ make ARCH=arm
CROSS_COMPILE=/toolchains/git/arm-2009q1-203/bin/arm-none-linux-gnueabi-
 SYSLINK_PLATFORM=OMAPL1XX

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/messageQ/knl/Linux
$ make ARCH=arm
```

```
CROSS_COMPILE=/toolchains/git/arm-2009q1-203/bin/arm-none-linux-gnueabi-
 SYSLINK_PLATFORM=OMAPL1XX

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/heapBufMP/knl/Linux
$ make ARCH=arm
CROSS_COMPILE=/toolchains/git/arm-2009q1-203/bin/arm-none-linux-gnueabi-
 SYSLINK_PLATFORM=OMAPL1XX

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/heapMemMP/knl/Linux
$ make ARCH=arm
CROSS_COMPILE=/toolchains/git/arm-2009q1-203/bin/arm-none-linux-gnueabi-
 SYSLINK_PLATFORM=OMAPL1XX

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/frameq/knl/Linux
$ make ARCH=arm
CROSS_COMPILE=/toolchains/git/arm-2009q1-203/bin/arm-none-linux-gnueabi-
 SYSLINK_PLATFORM=OMAPL1XX

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/listMP/knl/Linux
$ make ARCH=arm
CROSS_COMPILE=/toolchains/git/arm-2009q1-203/bin/arm-none-linux-gnueabi-
 SYSLINK_PLATFORM=OMAPL1XX

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/ringIO/knl/Linux
$ make ARCH=arm
CROSS_COMPILE=/toolchains/git/arm-2009q1-203/bin/arm-none-linux-gnueabi-
 SYSLINK_PLATFORM=OMAPL1XX

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/ringIO_gpp/knl/Linux
$ make ARCH=arm
CROSS_COMPILE=/toolchains/git/arm-2009q1-203/bin/arm-none-linux-gnueabi-
 SYSLINK_PLATFORM=OMAPL1XX

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/sharedRegion/knl/Linux
$ make ARCH=arm
CROSS_COMPILE=/toolchains/git/arm-2009q1-203/bin/arm-none-linux-gnueabi-
 SYSLINK_PLATFORM=OMAPL1XX

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/gateMP/knl/Linux
$ make ARCH=arm
CROSS_COMPILE=/toolchains/git/arm-2009q1-203/bin/arm-none-linux-gnueabi-
 SYSLINK_PLATFORM=OMAPL1XX
```

- Build SysLink user samples. Run make command in the respective sample directories.

```
$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/procMgr/usr/Linux
$ make
```

```
$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/notify/usr/Linux
$ make

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/messageQ/usr/Linux
$ make

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/listMP/usr/Linux
$ make

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/heapBufMP/usr/Linux
$ make

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/heapMemMP/usr/Linux
$ make

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/frameq/usr/Linux
$ make SYSLINK_PLATFORM=OMAPL1XX

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/ringIO/usr/Linux
$ make SYSLINK_PLATFORM=OMAPL1XX

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/ringIO_gpp/usr/Linux
$ make

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/sharedRegion/usr/Linux
$ make SYSLINK_PLATFORM=OMAPL1XX

$ cd $SYSLINK_ROOT/ti/syslink/samples/hlos/gateMP/usr/Linux
$ make
```

Building RTOS SysLink and sample applications

- Set SYSLINK_ROOT environment variable.This should be set to the base of the SysLink repository, i.e. the folder containing the `ti` directory.

```
$ export SYSLINK_ROOT=~/syslink_<VERSION>
```

- Set XDC in path.

```
$ export PATH=$PATH:/toolchains/xdc/xdctools_<version>
```

Alternatively, use the full path while making calls to xdc in the commands given later in this section

- Set XDC related environment variables.

```
$ export
XDCPATH="/users/ipc/ipc_<version>/packages;/toolchains/bios6/bios_<version>/packages"
```

Replace <version> above with the specific version for IPC, BIOS & XDC tools

- Following statement needs to be added to all application config(.cfg) files

```
xdc.loadPackage ('ti.syslink.ipc.rtos');
```

this ensures that all required linker options are included by default
User needs to add the above mentioned statement to application config (.cfg) files in order to include all required linker options by default

- Build RTOS SysLink. This builds RTOS FrameQ and RingIO

```
$ cd $SYSLINK_ROOT/ti/syslink/
$ xdc all XDCBUILDCFG="$SYSLINK_ROOT/config.bld" -PR .
```

- Default build profie is "whole_program_debug" to change this to "debug" run the following

```
 $ xdc all XDCARGS="profile=debug"
XDCBUILDCFG="$SYSLINK_ROOT/config.bld" -PR .
```

alternatively "XDCARGS" can be exported

```
export XDCARGS=profile=debug
```

```
$ xdc all XDCBUILDCFG="$SYSLINK_ROOT/config.bld" -PR .
```

- If you want to exclude Samples and build rest of the libraries, use following command

```
 xdc  XDCBUILDCFG="$SYSLINK_ROOT/config.bld" -PR 'xdcpkg syslink | grep
 -v /samples'
```

The step to build SysLink RTOS side also builds all sample applications. If the syslink/ipc and sample applications are to be built independently, this can be done through separate commands as given below.

- To build only the sample applications independently, use the following commands:

```
$ cd $SYSLINK_ROOT/ti/syslink/ipc
$ xdc all -PR .

$ cd $SYSLINK_ROOT/ti/syslink/samples/rtos/platforms
$ xdc all -PR .

$ cd $SYSLINK_ROOT/ti/syslink/samples/rtos/notify
$ xdc all

$ cd $SYSLINK_ROOT/ti/syslink/samples/rtos/messageQ
$ xdc all

$ cd $SYSLINK_ROOT/ti/syslink/samples/rtos/frameq
$ xdc all

$ cd $SYSLINK_ROOT/ti/syslink/samples/rtos/heapBufMP
$ xdc all

$ cd $SYSLINK_ROOT/ti/syslink/samples/rtos/heapMemMP
$ xdc all

$ cd $SYSLINK_ROOT/ti/syslink/samples/rtos/listMP
$ xdc all
```

```
$ cd $SYSLINK_ROOT/ti/syslink/samples/rtos/ringIO
$ xdc all

$ cd $SYSLINK_ROOT/ti/syslink/samples/rtos/ringIO_gpp
$ xdc all

$ cd $SYSLINK_ROOT/ti/syslink/samples/rtos/sharedRegion
$ xdc all

$ cd $SYSLINK_ROOT/ti/syslink/samples/rtos/gateMP
$ xdc all
```

Build Notes

The default build configuration for SysLink and sample applications assumes OMAP3530. If no specific SYSLINK_PLATFORM value is specified, OMAP3530 is assumed by default.

Only COFF format is supported for the DSP.

For additional information on different build parameters, please refer to the UserGuide document.

Configuring Kernel Parameters

SysLink requires a few specific arguments to be passed to the Linux kernel during boot up. For running the sample applications, 3MB of memory is used by SysLink for communication between GPP and DSP, and for DSP external memory for placing its code/data. This must be reserved by specifying 3MB less as available for the Linux kernel for its usage.

Depending on the memory map used for the final system configuration, the memory to be reserved from Linux may be different.

Running the sample applications

Sample applications are provided with SysLink for the supported platforms. This section describes the way to execute the sample applications.

The steps for execution of the samples are given below for execution with Linux running on the GPP.

For all kernel-side applications, ProcMgrApp is expected to be run before running the specific application to load and start the slave processor.

For all user-side applications, if no command line arguments are provided while running the sample application, the ProcMgrApp application is expected to be run before running the specific application to load and start the slave processor.

If the following command line arguments are provided, then the user-side application also loads/starts/stops the slave processor in the application context itself. In this case, ProcMgrApp application is not required to be run before running the specific application:

```
Arguments: <procId1> <Path including name of the slave
executable>
```

Copying files to target file system

The generated binaries on the HLOS side and RTOS side must be copied to the target directory.

For executing the sample applications, follow the steps below to copy the relevant binaries:

- Copy the syslink.ko kernel module and the application kernel modules into the target file system

```
$ cp ${SYSLINK_ROOT}/ti/syslink/lib/modules/OMAPL1XX/*.ko
${HOME}/omapl1xx/target/opt/syslink/.
```

- Copy the user samples into the target file system

```
$ cp ${SYSLINK_ROOT}/ti/syslink/lib/samples/*.exe
$HOME}/omapl1xx/target/opt/syslink/
```

To copy RTOS binaries, use the following steps:

- Copy the rtos executables into the target file system

```
$ cp
${SYSLINK_ROOT}/ti/syslink/samples/rtos/notify/evmDA830_dsp/debug/notify_omapl1xx_dsp.x67
 ${HOME}/omapl1xx/target/opt/syslink/
$ cp
${SYSLINK_ROOT}/ti/syslink/samples/rtos/messageQ/evmDA830_dsp/debug/messageq_omapl1xx_dsp
 ${HOME}/omapl1xx/target/opt/syslink/
$ cp
${SYSLINK_ROOT}/ti/syslink/samples/rtos/frameq/evmDA830_dsp/debug/frameq_omapl1xx_dsp.x67
 ${HOME}/omapl1xx/target/opt/syslink/
$ cp
${SYSLINK_ROOT}/ti/syslink/samples/rtos/listMP/evmDA830_dsp/debug/listmp_omapl1xx_dsp.x67
 ${HOME}/omapl1xx/target/opt/syslink/
$ cp
${SYSLINK_ROOT}/ti/syslink/samples/rtos/heapBufMP/evmDA830_dsp/debug/heapbufmp_omapl1xx_d
 ${HOME}/omapl1xx/target/opt/syslink/
$ cp
${SYSLINK_ROOT}/ti/syslink/samples/rtos/heapMemMP/evmDA830_dsp/debug/heapmemmp_omapl1xx_d
 ${HOME}/omapl1xx/target/opt/syslink/
$ cp
${SYSLINK_ROOT}/ti/syslink/samples/rtos/ringIO/evmDA830_dsp/debug/ringio_omapl1xx_dsp.x67
 ${HOME}/omapl1xx/target/opt/syslink/
$ cp
${SYSLINK_ROOT}/ti/syslink/samples/rtos/ringIO_gpp/evmDA830_dsp/debug/ringiogpp_omapl1xx_
 ${HOME}/omapl1xx/target/opt/syslink/
$ cp
${SYSLINK_ROOT}/ti/syslink/samples/rtos/sharedRegion/evmDA830_dsp/debug/sharedregion_omap
 ${HOME}/omapl1xx/target/opt/syslink/
$ cp
${SYSLINK_ROOT}/ti/syslink/samples/rtos/gateMP/evmDA830_dsp/debug/gatemp_omapl1xx_dsp.x67
 ${HOME}/omapl1xx/target/opt/syslink/
```

Running applications

Loading syslink kernel module

- The syslink kernel module must be inserted before running any sample applications.

```
$ insmod syslink.ko TRACE=1 TRACEFAILURE=1
```

Enabling TRACEFAILURE puts out prints in case any failure occurs during SysLink initialization.

Unloading syslink kernel module

```
$ rmmod syslink
```

Running the ProcMgr sample application

ProcMgr user-side sample application

- To invoke the application enter the following commands:

```
$ procmgrapp.exe 0 /opt/syslink/notify_omapl1xx_dsp.x674
```

procmgrapp.exe takes two parameters remote procId and executable name.

- Run procMgr user sample to load and start DSP:

```
$ procmgrapp.exe 0 /opt/syslink/notify_omapl1xx_dsp.x674
```

- The user-side procmgrapp example can also be used to act as a user-space loader while running the applications. To use it, press `Ctrl Z` after the following print is seen on the terminal:

```
Press any key to continue and perform shutdown ...
```

- Now run the specific application. For example:

```
$ ./notifyapp.exe
```

- Use `fg` to bring the procmgrapp.exe application back to the foreground. Press ‘enter’ to run it to completion and stop the slave.

Running the Notify sample application

Notify kernel-side sample application

- To invoke the application enter the following commands:

```
$ procmgrapp.exe 0 /opt/syslink/notify_omapl1xx_dsp.x674
```

- Press `Ctrl Z` to make it a background process
- Run the notifyapp sample application by inserting its kernel module

```
$ insmod notifyapp.ko
$ rmmod notifyapp
$ fg
```

- Press 'enter' to exit and cleanup.

Notify user-side sample application

- To invoke the application standalone, enter the following commands:

```
$ ./notifyapp.exe 0 /opt/syslink/notify_omapl1xx_dsp.x674
```

- Press 'enter' to exit and cleanup.

Alternatively:

- To invoke the application with ProcMgrApp, enter the following commands:

```
$ procmgrapp.exe 0 /opt/syslink/notify_omapl1xx_dsp.x674
```

- Press `Ctrl Z` to make it a background process
- Run the notifyapp sample application

```
$ ./notifyapp.exe
$ fg
```

- Press 'enter' to exit and cleanup.

Running the MessageQ sample application

MessageQ kernel-side sample application

- To invoke the application enter the following commands:

```
$ procmgrapp.exe 0 /opt/syslink/messageq_omapl1xx_dsp.x674
```

- Press `Ctrl Z` to make it a background process
- Run the messageqapp sample application by inserting its kernel module

```
$ insmod messageqapp.ko
$ rmmod messageqapp
$ fg
```

- Press 'enter' to exit and cleanup.

MessageQ user-side sample application

- To invoke the application standalone, enter the following commands:

```
$ ./messageqapp.exe 0 /opt/syslink/messageq_omapl1xx_dsp.x674
```

Alternatively:

- To invoke the application with ProcMgrApp, enter the following commands:

```
$ procmgrapp.exe 0 /opt/syslink/messageq_omapl1xx_dsp.x674
```

- Press `Ctrl Z` to make it a background process
- Run the messageqapp sample application

```
$ ./messageqapp.exe
$ fg
```

- Press 'enter' to exit and cleanup.

Running the ListMP sample application

ListMP kernel-side sample application

- To invoke the application enter the following commands:

```
$ procmgrapp.exe 0 /opt/syslink/listmp_omapl1xx_dsp.x674
```

- Press `Ctrl Z` to make it a background process
- Run the listmpapp sample application by inserting its kernel module

```
$ insmod listmpapp.ko
$ rmmod listmpapp
$ fg
```

- Press 'enter' to exit and cleanup.

ListMP user-side sample application

- To invoke the application standalone, enter the following commands:

```
$ ./listmpapp.exe 0 /opt/syslink/listmp_omapl1xx_dsp.x674
```

Alternatively:

- To invoke the application with ProcMgrApp, enter the following commands:

```
$ procmgrapp.exe 0 /opt/syslink/listmp_omapl1xx_dsp.x674
```

- Press `Ctrl Z` to make it a background process
- Run the listmpapp sample application

```
$ ./listmpapp.exe
$ fg
```

- Press 'enter' to exit and cleanup.

Running the HeapBufMP sample application

HeapBufMP kernel-side sample application

- To invoke the application enter the following commands:

```
$ procmgrapp.exe 0 /opt/syslink/heapbufmp_omapl1xx_dsp.x674
```

- Press `Ctrl Z` to make it a background process
- Run the heapbufmpapp sample application by inserting its kernel module

```
$ insmod heapbufmpapp.ko
$ rmmod heapbufmpapp
$ fg
```

- Press 'enter' to exit and cleanup.

HeapBufMP user-side sample application

- To invoke the application standalone, enter the following commands:

```
$ ./heapbufmpapp.exe 0 /opt/syslink/heapbufmp_omapl1xx_dsp.x674
```

Alternatively:

- To invoke the application with ProcMgrApp, enter the following commands:

```
$ procmgrapp.exe 0 /opt/syslink/heapbufmp_omapl1xx_dsp.x674
```

- Press `Ctrl Z` to make it a background process
- Run the heapbufmpapp sample application

```
$ ./heapbufmpapp.exe
$ fg
```

- Press 'enter' to exit and cleanup.

Running the HeapMemMP sample application

HeapMemMP kernel-side sample application

- To invoke the application enter the following commands:

```
$ procmgrapp.exe 0 /opt/syslink/heapmemmp_omapl1xx_dsp.x674
```

- Press `Ctrl Z` to make it a background process
- Run the heapmemmpapp sample application by inserting its kernel module

```
$ insmod heapmemmpapp.ko
$ rmmod heapmemmpapp
$ fg
```

- Press 'enter' to exit and cleanup.

HeapMemMP user-side sample application

- To invoke the application standalone, enter the following commands:

```
$ ./heapmemmpapp.exe 0 /opt/syslink/heapmemmp_omapl1xx_dsp.x674
```

Alternatively:

- To invoke the application with ProcMgrApp, enter the following commands:

```
$ procmgrapp.exe 0 /opt/syslink/heapmemmp_omapl1xx_dsp.x674
```

- Press `Ctrl Z` to make it a background process
- Run the heapmemmpapp sample application

```
$ ./heapmemmpapp.exe
$ fg
```

- Press 'enter' to exit and cleanup.

Running the FrameQ sample application

FrameQ kernel-side sample application

- To invoke the application enter the following commands:

```
$ procmgrapp.exe 0 /opt/syslink/frameq_omapl1xx_dsp.x674
```

- Press `Ctrl Z` to make it a background process
- Run the frameqapp sample application by inserting its kernel module

```
$ insmod frameqapp.ko
$ rmmod frameqapp
$ fg
```

- Press 'enter' to exit and cleanup.

FrameQ user-side sample application

- To invoke the application standalone, enter the following commands:

```
$ ./frameqapp.exe 0 /opt/syslink/frameq_omapl1xx_dsp.x674
```

Alternatively:

- To invoke the application with ProcMgrApp, enter the following commands:

```
$ procmgrapp.exe 0 /opt/syslink/frameq_omapl1xx_dsp.x674
```

- Press `Ctrl Z` to make it a background process
- Run the frameqapp sample application

```
$ ./frameqapp.exe
$ fg
```

- Press 'enter' to exit and cleanup.

Running the RingIO sample application

RingIO kernel-side sample application

- To invoke the application enter the following commands:

```
$ procmgrapp.exe 0 /opt/syslink/ringio_omapl1xx_dsp.x674
```

- Press `Ctrl Z` to make it a background process
- Run the ringioapp sample application by inserting its kernel module

```
$ insmod ringioapp.ko
$ rmmod ringioapp
$ fg
```

- Press 'enter' to exit and cleanup.

RingIO user-side sample application

- To invoke the application standalone, enter the following commands:

```
$ ./ringioapp.exe 0 /opt/syslink/ringio_omapl1xx_dsp.x674
```

Alternatively:

- To invoke the application with ProcMgrApp, enter the following commands:

```
$ procmgrapp.exe 0 /opt/syslink/ringio_omapl1xx_dsp.x674
```

- Press `Ctrl Z` to make it a background process
- Run the ringioapp sample application

```
$ ./ringioapp.exe
$ fg
```

- Press 'enter' to exit and cleanup.

Running the RingIO GPP sample application

RingIO GPP kernel-side sample application

- To invoke the application enter the following commands:

```
$ procmgrapp.exe 0 /opt/syslink/ringiogpp_omapl1xx_dsp.x674
```

- Press `Ctrl Z` to make it a background process
- Run the ringiogppapp sample application by inserting its kernel module

```
$ insmod ringiogppapp.ko
$ rmmod ringiogppapp
$ fg
```

- Press 'enter' to exit and cleanup.

RingIO GPP user-side sample application

- To invoke the application standalone, enter the following commands:

```
$ ./ringiogppapp.exe 0 /opt/syslink/ringiogpp_omapl1xx_dsp.x674
```

Alternatively:

- To invoke the application with ProcMgrApp, enter the following commands:

```
$ procmgrapp.exe 0 /opt/syslink/ringiogpp_omapl1xx_dsp.x674
```

- Press `Ctrl Z` to make it a background process
- Run the ringiogppapp sample application

```
$ ./ringiogppapp.exe
$ fg
```

- Press 'enter' to exit and cleanup.

Running the SharedRegion sample application

SharedRegion kernel-side sample application

- To invoke the application enter the following commands:

```
$ procmgrapp.exe 0 /opt/syslink/sharedregion_omapl1xx_dsp.x674
```

- Press `Ctrl Z` to make it a background process
- Run the sharedregionapp sample application by inserting its kernel module

```
$ insmod sharedregionapp.ko
$ rmmod sharedregionapp
$ fg
```

- Press 'enter' to exit and cleanup.

SharedRegion user-side sample application

- To invoke the application standalone, enter the following commands:

```
$ ./sharedregionapp.exe 0 /opt/syslink/sharedregion_omapl1xx_dsp.x674
```

Alternatively:

- To invoke the application with ProcMgrApp, enter the following commands:

```
$ procmgrapp.exe 0 /opt/syslink/sharedregion_omapl1xx_dsp.x674
```

- Press `Ctrl Z` to make it a background process
- Run the sharedregionapp sample application

```
$ ./sharedregionapp.exe
$ fg
```

- Press 'enter' to exit and cleanup.

Running the GateMP sample application

GateMP kernel-side sample application

- To invoke the application enter the following commands:

```
$ procmgrapp.exe 0 /opt/syslink/gatemp_omapl1xx_dsp.x674
```

- Press `Ctrl Z` to make it a background process
- Run the gatempapp sample application by inserting its kernel module

```
$ insmod gatempapp.ko
$ rmmod gatempapp
$ fg
```

- Press 'enter' to exit and cleanup.

GateMP user-side sample application

- To invoke the application standalone, enter the following commands:

```
$ ./gatempapp.exe 0 /opt/syslink/gatemp_omapl1xx_dsp.x674
```

Alternatively:

- To invoke the application with ProcMgrApp, enter the following commands:

```
$ procmgrapp.exe 0 /opt/syslink/gatemp_omapl1xx_dsp.x674
```

- Press `Ctrl Z` to make it a background process
- Run the gatempapp sample application

```
$ ./gatempapp.exe
$ fg
```

- Press 'enter' to exit and cleanup.

Article Sources and Contributors

SysLink 02.00.00.56 alpha2 InstallGuide *Source*: http://ap-fpdsp-swapps.dal.design.ti.com/index.php?oldid=72956 *Contributors*: MugdhaKamoolkar, NagabhushanReddy, Ravindranath Andela, Rk

SysLink 02.00.00.56 alpha2 InstallGuide Linux TI81XX *Source*: http://ap-fpdsp-swapps.dal.design.ti.com/index.php?oldid=75054 *Contributors*: MugdhaKamoolkar, NagabhushanReddy, Ravindranath Andela, Rk

SysLink 02.00.00.56 alpha2 InstallGuide Linux OMAP3530 *Source*: http://ap-fpdsp-swapps.dal.design.ti.com/index.php?oldid=72967 *Contributors*: MugdhaKamoolkar, NagabhushanReddy, Ravindranath Andela

SysLink 02.00.00.56 alpha2 InstallGuide Linux OMAPL1XX *Source*: http://ap-fpdsp-swapps.dal.design.ti.com/index.php?oldid=72971 *Contributors*: MugdhaKamoolkar, NagabhushanReddy, Ravindranath Andela, Rk

Image Sources, Licenses and Contributors

Image:TIBanner.PNG *Source*: http://ap-fpdsp-swapps.dal.design.ti.com/index.php?title=File:TIBanner.PNG *License*: unknown *Contributors*: SekharNori