IPC Install Guide Linux

Introduction

Inter/Intra Processor Communication (IPC) is a product designed to enable communication between processors in a multi-processor environment. Features of IPC include message passing, multi-processor gates, shared memory primitives, and more.

IPC is designed for use with processors running SYS/BIOS applications. This is typically an ARM or DSP. IPC includes support for High Level Operating Systems (HLOS) like Linux, as well as the SYS/BIOS RTOS. The breadth of IPC features supported in an HLOS environment is reduced in an effort to simplify the product.

Install

IPC is released as a zip file. To install, simply extract the file.

```
buildhost$ unzip ipc_<version>.zip
```

This will extract the IPC product in a directory with its product name and version information (e.g. **c:\ipc_3_xx_xx_xx** or **/home/<user>/ipc_3_xx_xx_xx**)

NOTE

- This document assumes the IPC install path to be the user's home directory on a Linux host machine (**/home/<user>**) or the user's main drive on a Windows host machine (**C:**). The variable **IPC_INSTALL_DIR** will be used throughout the document. If IPC was installed at a different location, make appropriate changes to commands.
- Some customers find value in archiving the released sources in a configuration management system. This can help in identifying any changes made to the original sources - often useful when updating to newer releases.

Build

The IPC product often comes with prebuilt SYS/BIOS-side libraries, so rebuilding them isn't necessary. The Linux-side user libraries may also be provided prebuilt, but customers often want to change the configuration (e.g. static, dynamic).

IPC provides GNU makefile(s) to rebuild all its libraries at the base of the product, details are below.

NOTE

GNU make version 3.81 or greater is required. The XDC tools (provided with most SDKs and CCS distributions) includes a pre-compiled version of GNU make 3.81 in $(XDC_INSTALL_DIR)/gmake.

products.mak

IPC contains a **products.mak** file at the root of the product that specifies the necessary paths and options to build IPC for the various OS support.

Edit **products.mak** and set the following variables:

- Linux
  - **TOOLCHAIN_INSTALL_DIR** - Path to the devices ARM Linux cross-compiler toolchain
  - **TOOLCHAIN_LONGNAME** - Long name of the devices toolchain (e.g. arm-none-linux-gnueabi)
  - **PLATFORM** - (Optional) Device to configure for Linux-side builds.
    - To find the supported platform list run: **./configure --help**

    - If not set, all supported platforms will be built.
  - **CMEM_INSTALL_DIR** - (Optional) Path to TI Linux Utils package to locate the pre-built CMEM [1] libraries used by some tests.
    - This option will build additional test applications for select platforms.
- SYS/BIOS
  - **XDC_INSTALL_DIR** - Path to TI's XDCTools installation
  - **BIOS_INSTALL_DIR** - Path to TI's SYS/BIOS installation
  - **ti.targets.<device target and file format>** - Path to TI toolchain for the device.
  - **gnu.targets.arm.<device target and file format>** - Path to GNU toolchain for the device.
    - Set only the variables to the targets your device supports to minimize build time.

NOTE

The dependencies applicable for each supported device can be found in the IPC Release Notes provided in the product.

ipc-linux.mak

The Linux-side build is provided as a GNU Autotools (Autoconf, Automake, Libtool) project. If you are familiar with Autoconf GNU projects, you can proceed with using the **./configure** script directly to cross-compile the Linux user libraries and tests.

For those that require some assistance, the IPC package provides a GNU makefile (**ipc-linux.mak**) to configure the Linux-side build, using the options and component paths set in the **products.mak** file. To configure the build using these files, issue the following command:

```
<buildhost> make -f ipc-linux.mak config
```

There are few additional target goals provided in the ipc-linux.mak file for commonly used configurations. These goals include:

- config - (Default) Configure both static and shared (dynamic) Linux IPC user libraries. Test applications link against the shared libraries.
- config-static - Configure static only libraries and tests.
- config-shared - Configure shared (dynamic) only libraries and tests.

Then build the Linux side of IPC by issuing the following:

```
<buildhost> make
```

You can also specify a PLATFORM to (re)configure for on the command line which overrides any options set in the products.mak file as follows:

```
  <buildhost> make -f ipc-linux.mak config PLATFORM=omapl138
  <buildhost> make
```

When reconfiguring for a new Linux toolchain or platform, the Linux build should be clean(ed) before reconfiguring:

```
  <buildhost> make clean
```

ipc-bios.mak

The SYS/BIOS-side IPC is built with a GNU makefile. After editing **products.mak**, issue the following command:

```
<buildhost> make -f ipc-bios.mak all
```

Based on the number of targets you're building for, this may take some time.

Run

The IPC product provides a way to install (copy) the necessary IPC executables and libraries onto the device's target file-system to simplify the execution of the applications.

Configuring Kernel

The IPC product provides a set of Linux kernel patches that need to be applied to the different device supported kernels to add necessary kernel support. The patches are located in the linux/patches directory of the IPC installation.

OMAP-L138

The kernel for the OMAP-L138, can be obtained from Gitorious linux-davinci project [2].

The patches apply to the following commit id: **595ab716fc6e648b7dc79a58a01917ebb67b9508**

The specific patches needed for this kernel can be found in the **linux/patches/3.8.0** of your IPC installation.

Once the patches are applied, there are a few key config parameters needed for rpmsg and socket driver to build/work.

CONFIG_REMOTEPROC=m

CONFIG_DA8XX_REMOTEPROC=m

CONFIG_RPMSG=m

CONFIG_VIRTIO=m

It is also recommended to compile a Linux kernel with the debugfs facility

CONFIG_DEBUG_FS=y

Re-build the kernel. For example:

```
buildhost$ make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi-  uImage
```

You will also need to re-build the kernel modules and install them on your target's file system. For example:

```
buildhost$ make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi-  modules
buildhost$ make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi-
INSTALL_MOD_PATH=<target filesystem> modules_install
```

Kernel Boot-up Parameters

IPC requires an argument to be passed to the Linux kernel during boot up to properly run the tests. The remote processor(s) (rproc) memory location needs to be set.

- For example,

```
bootargs console=ttyS2,115200n8 root=/dev/nfs
nfsroot=HOST:nfs_root,nolock rw ip=dhcp rproc_mem=16M@0xC3000000
```

This is just an example, bootargs may vary depending on available setup

Depending on the memory map used in the final system configuration, the memory to be reserved for rproc usage

may differ.

Installing Tests

To assemble the IPC test executables and libraries into a directory structure suitable for running on the target's file-system, issue the following command in the IPC install directory:

```
buildhost$ make install prefix=<target filesystem>/usr
```

*Depending on you target's filesystem directory privileges, you may be required to run **sudo make install** to properly install the files*

NOTE

The test executables and libraries will be installed in the location path set by the **prefix** variable. If you are installing directly on a host mounted Network Filesystem(NFS), make sure to specify **usr** at the end of the **prefix** variable path. As with other variables, you can override this on the command line:

```
buildhost$ sudo make install prefix=<target filesystem>/usr
```

The remote processor's applications will be loaded via the remote_proc kernel module but they need to reside on the devices target filesystem in **/lib/firmware** directory. The location of the remote core application within the IPC product various based on device.

OMAP-L138

The OMAP-L138 remote core applications can be found in IPC_INSTALL_DIR/packages/ti/ipc/test/ti_platforms_evmOMAPL138_DSP directory.

Copy the messageq_single.xe674 onto the devices target filesystem into the **/lib/firmware** directory.

```
buildhost$ cp
ti/ipc/test/ti_platforms_evmOMAPL138_DSP/ti/ipc/test/ti_platforms_evmOMAPL138_DSP
 <target filesystem>/lib/firmware/.
```

IPC Daemons and Drivers

IPC provides system-wide services across multiple applications, and utilizes low-level system hardware (e.g. interrupts and shared memory). To facilitate these services, IPC uses a user-space daemon (LAD) and several kernel device drivers.

LAD

System-wide IPC state is managed by a user-space daemon (LAD). This daemon is specific to a given device, and is named lad_<device>. It will reside on the target's filesystem (typically in /usr/bin/) after following the #Installing Tests section. To run LAD, execute:

```
target# /usr/bin/lad_<device>
```

LAD takes an optional argument which is a filename where log statements will be emitted. This file will be created in the /tmp/LAD/ directory. For example, to instruct LAD to emit log statements into a 'lad.txt' file, start LAD like this:

```
target# /usr/bin/lad_<device> lad.txt
```

Drivers

The kernel drivers/modules added by the Linux patches must be inserted into the kernel for IPC applications to run correctly. Refer to the #Configuring Kernel section. The required modules must be configured, built and loaded onto the target's filesystem.

Prior to loading the modules, a directory (/debug) must be created at the root of your devices filesystem. This directory will be mounted as a debugfs (debug filesystem) which the kernel modules will use to provide details about the slaves (e.g. running state, trace output, etc). If the /debug directory doesn't exist, simply create it as follows:

```
target# mkdir /debug
```

OMAP-L138

The kernel modules can be loaded by issuing the following command on the target's file-system:

```
target# depmod -a
target# mount -t debugfs none /debug
target# modprobe remoteproc
target# modprobe da8xx_remoteproc fw_name=messageq_single.xe674
target# modprobe virtio_rpmsg_bus
target# modprobe rpmsg_proto
```

The kernel modules can be unloaded by issuing the following command on the target's file-system:

```
target# umount /debug
target# rmmod rpmsg_proto
target# rmmod virtio_rpmsg_bus
target# rmmod da8xx_remoteproc
target# rmmod remoteproc
```

Running Test Applications

The test applications are already on the target's filesystem in /usr/bin assuming the #Installing Tests section has been followed.

To run the test application's, execute the following on the target's filesystem:

```
target# /usr/bin/MessageQApp_<device>
```

OMAP-L138

The expected output on the Linux-side should be:

```
Using numLoops: 100; procId : 1
Entered MessageQApp_execute
Local MessageQId: 0x1
Remote queueId  [0x10000]
Exchanging 100 messages with remote processor DSP...
MessageQ_get #0 Msg = 0x15328
Exchanged 1 messages with remote processor DSP
MessageQ_get #1 Msg = 0x15328
...
...
Exchanged 99 messages with remote processor DSP
MessageQ_get #99 Msg = 0x15328
```

```
Exchanged 100 messages with remote processor DSP
Sample application successfully completed!
Leaving MessageQApp_execute
```

The output on the remote processor, can be obtained by running the following on the target filesystem:

```
target# cat /debug/remoteproc/remoteproc0/trace0
```

The expected output on the remote processor should be:

```
3 Resource entries at 0xc3100000
messageq_single.c:main: MultiProc id = 1
registering rpmsg-proto service on 61 with HOST
tsk1Fxn: created MessageQ: SLAVE_DSP; QueueID: 0x10000
Awaiting sync message from host...
[t=0x00000001:67984156] ti.ipc.rpmsg.MessageQCopy: MessageQCopy_send:
no object for endpoint: 53
[t=0x00000001:67f626ed] ti.ipc.rpmsg.MessageQCopy: MessageQCopy_send:
no object for endpoint: 53
Received msg from (procId:remoteQueueId): 0x0:0x1
       payload: 8 bytes; loops: 100 with printing.
Got msg #0 (40 bytes) from procId 0
Sending msg Id #0 to procId 0
Got msg #1 (40 bytes) from procId 0
Sending msg Id #1 to procId 0
...
...
Got msg #98 (40 bytes) from procId 0
Sending msg Id #98 to procId 0
Got msg #99 (40 bytes) from procId 0
Sending msg Id #99 to procId 0
Awaiting sync message from host...
[t=0x00000015:7b46c4c2] ti.ipc.rpmsg.MessageQCopy: MessageQCopy_send:
no object for endpoint: 53
[t=0x00000015:7b6315fb] ti.ipc.rpmsg.MessageQCopy: MessageQCopy_send:
no object for endpoint: 53
```

References

[1] http://processors.wiki.ti.com/index.php/Linux_Utils_Overview
[2] http://gitorious.org/linux-davinci

Article Sources and Contributors

IPC Install Guide Linux *Source*: http://ap-fpdsp-swapps.dal.design.ti.com/index.php?oldid=159725 *Contributors*: ChrisRing