IPC Install Guide Android

Contents

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 often distributed and installed within a larger Software Developer's Kit (SDK) or part of the Android Filesystem Sources (AFS) for your device. In those cases, no installation is required.

Outside of an SDK or AFS, IPC can be downloaded here (http://software-dl.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/ipc/index.html), and is released as a zip file. To install, simply extract the file.

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

On Android, the IPC product is designed to be built as part of the AFS distribution thus should be unzip within the AFS source structure (recommended **$(AFS_ROOT)/hardware/ti** directory). If your AFS already contains IPC, the sources are located in the **$(AFS_ROOT)/hardware/ti/ipc** directory.

> **NOTE**
>
> - This document assumes the IPC install path to be the within the AFS directory on a Linux host machine (**$(AFS_ROOT)/hardware/ti/ipc**). 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 slave-side libraries, so rebuilding them isn't necessary. The Android host-side user libraries may also be provided prebuilt on certain TI device's filsystems or as part of an SDK, but customers may want to rebuild if not available.

Host side

IPC provides Android specific makefile(s) to rebuild all its libraries and a few test applications from within an AFS distribution. Instructions on how to obtain an AFS for your device are available on the OMAPpedia page (http://omappedia.org/wiki/Main_Page).

Libraries

To build IPC libraries within a given AFS, you can issue the following commands from the top of your AFS distribution:

```
buildhost$ . build/envsetup.sh
buildhost$ lunch full_jacinto6evm-userdebug
```

```
buildhost$ make libmmrpc libtiipc libtiipcutils libtiipcutils_lad libtitransportrpmsg
```

IPC Daemon

To build the IPC daemon within a given AFS, you can issue the following commands from the top of your AFS distribution:

```
buildhost$ . build/envsetup.sh
buildhost$ lunch full_jacinto6evm-userdebug
buildhost$ make lad_dra7xx
```

Information on the IPC Daemon can be found here (http://processors.wiki.ti.com/index.php/IPC_Daemon)

Test applications

IPC development source repository comes with a few test applications to validate proper IPC functionality. To build IPC the test apps within a given AFS, you can issue the following commands from the top of your AFS distribution:

```
buildhost$ . build/envsetup.sh
buildhost$ lunch full_jacinto6evm-userdebug
buildhost$ make -j messageQApp messageQBench messageQMulti nameServerApp ping_rpmsg
```

Slave side

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:

- Variables used by **BIOS-side** build scripts
  - **PLATFORM** - Device to build for
  - **XDC_INSTALL_DIR** - Path to TI's XDCTools installation (e.g. **/opt/ti/xdctools_<version>**)
  - **BIOS_INSTALL_DIR** - Path to TI's SYS/BIOS installation (e.g. **/opt/ti/bios_<version>**)
  - **ti.targets.<device target and file format>** - Path to TI toolchain for the device. (e.g. **/opt/ti/ccsv5/tools/compiler/c6000_<version>**)
    - Set only the variables to the targets your device supports to minimize build time.

> **NOTE**
>
> The specific versions of dependent components can be found in the IPC Release Notes, provided in the product.

Libraries and Test applications

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.

> **NOTE**
>
> The BIOS-side libraries often come pre-built, so in many cases, rebuilding the BIOS-side is not necessary. Some reasons you may want to rebuild:
>
> - Your distribution of IPC **didn't** come with the necessary pre-built libraries
> - You intend to run the 'test' executables (which often don't come pre-built)
> - You want to use a specific toolchain or dependency version
> - You want to tune some of the compile options

Run

This section outlines some basic information on kernel support needed for IPC and how to run some of the test applications built above.

Configuring Kernel

IPC functionality requires the host kernel to support and enable the rpmsg and remoteproc drivers. Based on the device and kernel version, the drivers may may already be enabled by default.

DRA7XX

There are a few key config parameters to enable the rpmsg and remoteproc drivers for IPC:

```
CONFIG_VIRTIO=y

CONFIG_REMOTEPROC=y
CONFIG_OMAP_REMOTEPROC=y

CONFIG_RPMSG=y
CONFIG_RPMSG_RPC=y
```

Each slave core on the device can be enabled as needed to support remoteproc.

```
CONFIG_OMAP_REMOTEPROC_IPU=y
CONFIG_OMAP_REMOTEPROC_DSP=y
CONFIG_OMAP_REMOTEPROC_IPU1=y
CONFIG_OMAP_REMOTEPROC_DSP2=y
```

Installing Binaries

This section describes how to assemble the IPC test executables and libraries into a directory structure suitable for running on the devices's file-system.

Host

The host-side test binaries and libraries, once built, are found in your $(AFS_ROOT)/out/target/product/<device>/system/lib or bin directory.

The list of libraries for IPC are the following and should be copied to your device's /system/lib directory:

```
libmmrpc.so
libtiipc.so
libtiipcutils.so
libtiipcutils_lad.so
libtitransportrpmsg.so
```

The list of executables are the following and should be copied to your device's /system/bin directory:

```
lad_dra7xx
messageQApp
messageQBench
messageQMulti
nameServerApp
ping_rpmsg
```

The example below illustrates the use of ADB to copy the files from the host build machine to the device's filesystem for execution.

```
buildhost$ adb push $(AFS_ROOT)/out/target/product/jacinto6evm/vendor/bin/lad_dra7xx /vendor/bin/.
```

Or, for IPC releases prior to IPC 3.47.00:

```
buildhost$ adb push $(AFS_ROOT)/out/target/product/jacinto6evm/system/bin/lad_dra7xx /system/bin/.
```

Slave

The slave-side test binaries, once built, are found in your IPC_INSTALL_DIR/packages/ti/ipc/tests/bin/<platform>_<core> directory.

Copy the appropriate slave-side executable onto the device's filesystem **/vendor/firmware** directory. For example, DRA7XX developers would perform the following using ADB:

```
buildhost$ adb push IPC_INSTALL_DIR/packages/ti/ipc/tests/bin/ti_platforms_evmDRA7XX_ipu2/messageq_single.xem4 /vendor/firmware/ipu2/messageq_single.xem4
```

Once binaries are copied to the device's filesystem, the remoteproc driver expects specific binary names for the different cores enabled in the kernel. The files should be located at the root of the **/system/firmware** directory.

DRA7XX

For the dra7xxx the following slave binary names are expected.

```
For IPU1 - dra7-ipu1-fw.xem4
For IPU2 - dra7-ipu2-fw.xem4
For DSP1 - dra7-dsp1-fw.xe66
For DSP2 - dra7-dsp2-fw.xe66
```

Assume the following slave binaries on the device's filesystem /vendor/firmware/ipu2/messageq_single.xem4. We'll create a link to the file as follows:

```
target# ln -s /vendor/firmware/ipu2/messageq_single.xem4 /vendor/firmware/dra7-ipu2-fw.xem4
```

If the remoteproc driver for the slave core is enabled in the kernel, the binary will be loaded to the appropriate slave core based on it's name during bootup.

IPC Daemon

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). 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 /systen/bin/) after following the #Installing Binaries section. To run LAD, execute:

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

This forks the LAD daemon and leaves it running in the background.

LAD takes an optional argument to indicate a filename into which log statements should be emitted. This file will be created in the **/data/lad/LAD/** directory. How to specify the filename varies based on your IPC release. For example, to instruct LAD to emit log statements into a 'lad.txt' file, start LAD like this:

- Releases before IPC 3.21:

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

- IPC 3.21 and after:

```
target# /system/bin/lad_<device> -l lad.txt
```

Additionally, LAD takes an optional argument to enable GateMP support. To enable GateMP support in the LAD, launch the LAD with the "-g" option, like this:

```
target# /system/bin/lad_<device> -g
```

Running Test Applications

The test applications are already on the target's filesystem in /system/bin assuming the #Installing Binaries section has been followed. Make you the IPC daemon (LAD) has been started and the slave core has loaded the messageq_single.xe<device extension> binary.

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

```
target# /system/bin/messageQApp <# of messages> <processor id>
```

DRA7XX

The expected output on the Android-side should be:

```
Using numLoops: 100; procId : 1
Entered MessageQApp_execute
Local MessageQId: 0x1
Remote queueId  [0x10000]
Exchanging 100 messages with remote processor IPU2...
MessageQ_get #1 Msg = 0xb7b05478
MessageQ_get #2 Msg = 0xb7b05478
...
...
MessageQ_get #99 Msg = 0xb7b05478
MessageQ_get #100 Msg = 0xb7b05478
Exchanged 100 messages with remote processor IPU2
Sample application successfully completed!
Leaving MessageQApp_execute
```

The output on the remote processor, can be obtained by running the following on the device's filesystem:

```
target# cat /d/remoteproc/remoteproc1/trace0
```

The expected output on the remote processor should be:

```
[0][      0.000] 18 Resource entries at 0x3000
[0][      0.000] messageq_single.c:main: MultiProc id = 1
[0][      0.000] registering rpmsg-proto:rpmsg-proto service on 61 with HOST
[0][      0.000] tsk1Fxn: created MessageQ: SLAVE_IPU2; QueueID: 0x10000
[0][      0.000] Awaiting sync message from host...
[0][     54.206] Received msg from (procId:remoteQueueId): 0x0:0x1
[0][     54.206]        payload: 8 bytes; loops: 100 without printing.
[0][     54.306] 100 iterations took 100 ticks or 1000 usecs/msg
[0][     54.306] Awaiting sync message from host...
```

See Also

- IPC 3.x
- IPC Users Guide
- IPC 3.x FAQ
- IPC Install Guide Linux
- IPC Install Guide QNX
- IPC Install Guide BIOS

{{ 1. switchcategory:MultiCore= ▪ For technical support on MultiCore devices, please post your questions in the C6000 MultiCore Forum ▪ For questions related to the BIOS MultiCore SDK (MCSDK), please use the BIOS Forum Please post only comments related to the article **IPC Install Guide Android** here.	Keystone= ▪ For technical support on MultiCore devices, please post your questions in the C6000 MultiCore Forum ▪ For questions related to the BIOS MultiCore SDK (MCSDK), please use the BIOS Forum	C2000=*For technical support on the C2000 please post your questions on The C2000 Forum. Please post only comments about the article **IPC Install Guide***	DaVinci=*For technical support on DaVincoplease post your questions on The DaVinci Forum. Please post only comments about the article **IPC Install Guide Android** here.*	MSP430=*For technical support on MSP430 please post your questions on The MSP430 Forum. Please post only comments about the article **IPC Install Guide***	OMAP35x=*For technical support on OMAP please post your questions on The OMAP Forum. Please post only comments about the article **IPC Install Guide Android** here.*	OMAPL1=*For technical support on OMAP please post your questions on The OMAP Forum. Please post only comments about the article **IPC Install Guide Android** here.*	MAVRK=*For technical support on MAVRK please post your questions on The MAVRK Toolbox Forum. Please post only comments about the article **IPC Install***	*For technical s please post yo questions at http://e2e.ti.cor Please post on comments abo article **IPC Inst Guide Androi*** }}

Please post only comments related to the article **IPC Install Guide Android** here. ***Android*** *here.* ***Android*** *here.* ***Guide*** ***Android*** *here.*

Links

Amplifiers & Linear
Audio
Broadband RF/IF & Digital Radio
Clocks & Timers
Data Converters
DLP & MEMS
High-Reliability
Interface
Logic
Power Management
Processors

- ARM Processors
- Digital Signal Processors (DSP)
- Microcontrollers (MCU)
- OMAP Applications Processors

Switches & Multiplexers
Temperature Sensors & Control ICs
Wireless Connectivity