Intel® Infrastructure DSP Solution Version 1.1 Migration Guide: Migration from Intel® IXP400 Digital Signal Processing (DSP) Software Version 2.6.2

Application Note

August 2007

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY RELATING TO SALE AND/OR USE OF INTEL PRODUCTS, INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT, OR OTHER INTELLECTUAL PROPERTY RIGHT.

Intel Corporation may have patents or pending patent applications, trademarks, copyrights, or other intellectual property rights that relate to the presented subject matter. The furnishing of documents and other materials and information does not provide any license, express or implied, by estoppel or otherwise, to any such patents, trademarks, copyrights, or other intellectual property rights.

Intel products are not intended for use in medical, life-saving, life-sustaining, critical control or safety systems, or in nuclear-facility applications. Intel may make changes to specifications and product descriptions at any time, without notice.

Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

The Intel® Infrastructure DSP Solution software may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

MPEG is an international standard for video compression/decompression promoted by ISO. Implementations of MPEG CODECs, or MPEG enabled platforms may require licenses from various entities, including Intel Corporation.

This Application Note and the software described in it are furnished under license and may only be used or copied in accordance with the terms of the license. The information in this document is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Intel Corporation. Intel Corporation assumes no responsibility or liability for any errors or inaccuracies that may appear in this document or any software that may be provided in association with this document.

Except as permitted by such license, no part of this document may be reproduced, stored in a retrieval system, or transmitted in any form or by any means without the express written consent of Intel Corporation.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or by visiting Intel's website at http://www.intel.com.

Intel, Intel logo, and Intel XScale are trademarks of Intel Corporation in the U.S. and other countries.

*Other names and brands may be claimed as the property of others.

Copyright © Intel Corporation 2007

Contents

Figures

Tables

Revision History

Revision Number	Description	Revision Date
001	Initial release.	July 2007
002	Updated for migration to Intel® Infrastructure DSP Solution v1.1 release	August 2007

§

1 Introduction

Intel® Infrastructure DSP Solution Version 1.1 is a software module that provides basic voice and signal processing functionality for Voice-over-Internet Protocol (VoIP) on the Intel® IXP4XX Product Line of Network Processors.

This document highlights the differences between Intel® IXP400 DSP Software Version 2.6.2 and Intel® Infrastructure DSP Solution Version 1.1. New features introduced in Intel® Infrastructure DSP Solution Version 1.1 such as Multi-conference calls, programmable thread priority, extended report for VoIP statistics, EC-Tone Disabler and T.30 Preamble detection will not be covered in this document, please refer to respective Programmer's Guide or API Reference Manual document for details. The information provided in this document is intended specifically for migration purposes.

Note: Throughout this document, Intel® IXP400 DSP Software Version 2.6.2 shall be referred to as Version 2.6.2 and Intel® Infrastructure DSP Solution Version 1.1 shall be referred to as Version 1.1.

1.1 Scope

This application note is intended to highlight the major differences between Version 2.6.2 and Version 1.1 for migration purposes. This document focuses on architecture comparison, directory structure comparison, Linux kernel space and Linux user space implementation, HSS device driver and SLIC device driver implementation, and IP termination implementation. This application note is not intended to list out all the differences between both versions of software releases. For the supported environments of Version 2.6.2 and Version 1.1, please refer to the respective Release Notes

This document uses the codelets demo application (sample code) of both versions of software releases to compare the methodology of designing applications utilizing the API provided by both versions of software.

Version 1.1 does not support VxWorks* Operating System. Only Linux* Kernel 2.6.16 Operating System codelets demo application is covered in this comparison.

This application note is not intended to provide a line-by-line comparison of the codelets demo application source code of the two versions of software releases or to list out all the differences in codelets demo application. For explanation purposes, function references only include function names and do not include the argument to be passed to the function. Please refer to respective API Reference Manual document or codelets for details of the functions.

1.2 Audience

This document is intended for system architects or software engineers, who have implemented a platform solution using Version 2.6.2 codelets demo application and who wish to migrate to Version 1.1.

1.3 Terminology

Term	Description
AEC	Acoustic Echo Canceller
API	Application Programming Interface
DSP	Digital Signal Processing
EC	Echo Cancellation
FXO	Foreign Exchange Office
FXS	Foreign Exchange Subscriber
IP	Internet Protocol
MPR	Media Processing Resource
PCM	Pulse Code Modulation
RTP	Real-Time Protocol
SRTP	Secure Real-Time Protocol
SLIC	Subscriber Line Interface Circuit
TDM	Time Division Multiplexing
OSAL	Operating System Abstraction Layer
USCI	Unified Speech Component Interface

1.4 Reference Documents

Document
Intel® Infrastructure DSP Solution Version 1.1 Programmer's Guide
Intel® Infrastructure DSP Solution Version 1.1 API Reference Manual
Intel® Infrastructure DSP Solution Version 1.1 Release Notes
Intel® Infrastructure DSP Solution Version 1.1 Codelets Demo Guide

2 Architecture Comparison

Intel® IXP400 DSP Software Version 2.6.2 and Intel® Infrastructure DSP Solution Version 1.1 have major differences in architecture as listed in Table 1.

Table 1. Architecture Comparison

Intel® IXP400 DSP Software Version 2.6.2	Intel® Infrastructure DSP Solution Version 1.1
Software Release only provides single binary file. Both base library (also known as framework) and media processing algorithm are provided a single binary file.	Separation of base library (also known as framework) and other plug-in modules. Each plug-in module is a media processing algorithm, which can be plugged onto base library during build process. This feature provides flexibility to plug-in media processing algorithm during build process.
Fixed memory footprint as you cannot determine which media processing algorithm is to be included in the binary file.	Flexibility to choose the required plug-in media processing algorithms during build process. This feature translates to the flexibility to minimize the memory footprint to include only the required algorithms.
Library runs in Linux Kernel Mode	Library runs in Linux User Mode
Supports T.38 Media Processing Resource	Does not support T.38 Media Processing Resource

However, both versions are similar in terms of architecture design. Both versions are implemented as independent modules having their own tasks and runtime environments. The software architecture has a two-layer hierarchy:

- Control layer, which provides the control interface and control logic
- Data processing layer, where the media data streams are processed by appropriate algorithms

Figure 1 illustrates the architecture of Version 1.1 and Figure 2 illustrates the architecture of Version 2.6.2. In Version 1.1 architecture, software developers have the flexibility to use Intel-provided media processing algorithm modules or to plug in external algorithm modules according to Unified Speech Component Interface (USCI). Figure 1 illustrates the relationship of Plug-in Handler component (which follows USCI to interface between the framework of Version 1.1) and external media processing algorithm modules. As illustrated in Figure 2, this feature is not available in Version 2.6.2.

As shown in Figure 1, there are seven Media Processing Resource (MPR) components in Version 1.1. Version 2.6.2 supports one additional MPR, which is T.38 MPR. T.38 is an ITU-T specification for fax transmission protocol over IP. T.38 MPR is not supported in Version 1.1.

For Version 1.1, Encoder MPR, Decoder MPR and Network End-Point MPR, have the flexibility to plug in media processing algorithm modules during build process.

During planning for migration, you can customize the software solution based on the following available options for pluggable media processing algorithm modules:

- Encoder algorithm (either Version 1.1 -provided encoder algorithm modules or external encoder algorithm with interface compliant with USCI)
- Decoder algorithm (either Version 1.1-provided decoder algorithm modules or external decoder algorithm with interface compliant with USCI)
- Echo Cancellation algorithm (either Version 1.1 -provided echo cancellation algorithm module (Line Echo Canceller ONLY) or external echo cancellation algorithm with interface compliant with USCI.)

Note: The third party plug-in codecs or external plug-in codecs must be the same codec type as the supported Intel-provided codec type.

Figure 1. Architecture of Intel® Infrastructure DSP Solution Version 1.1

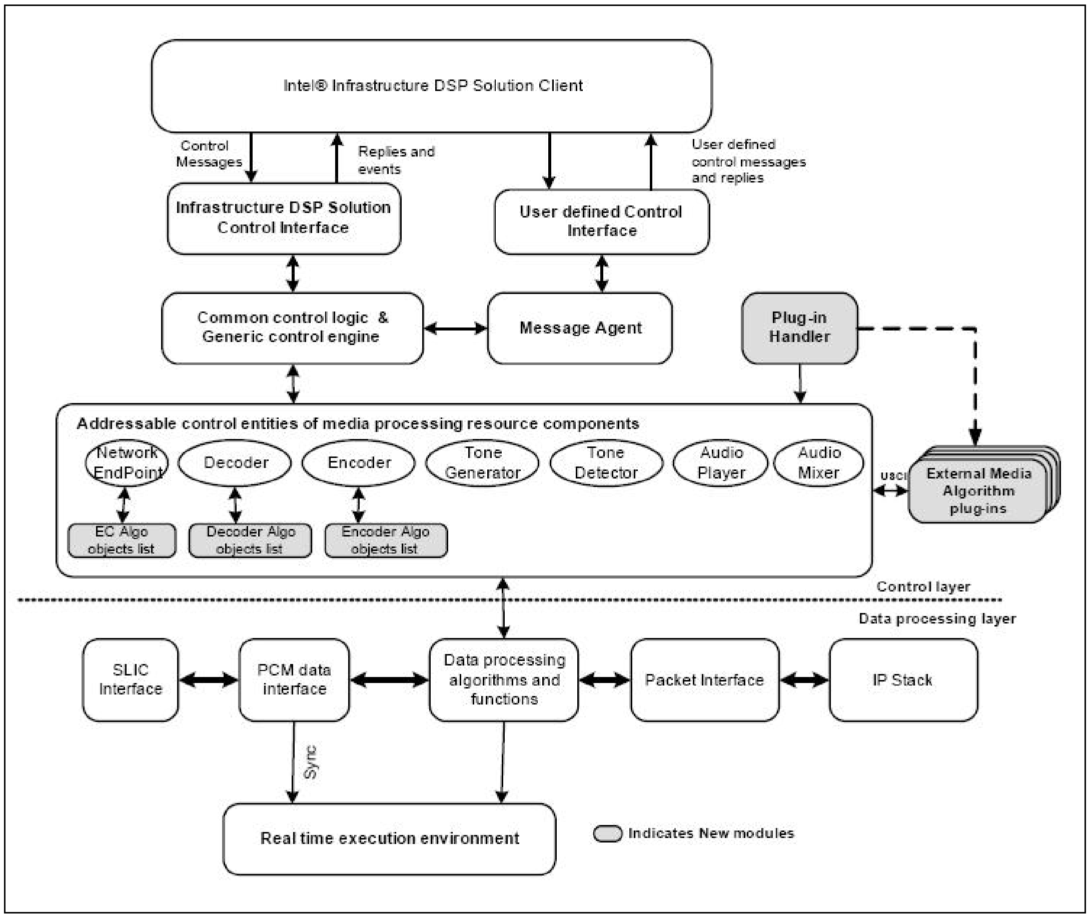
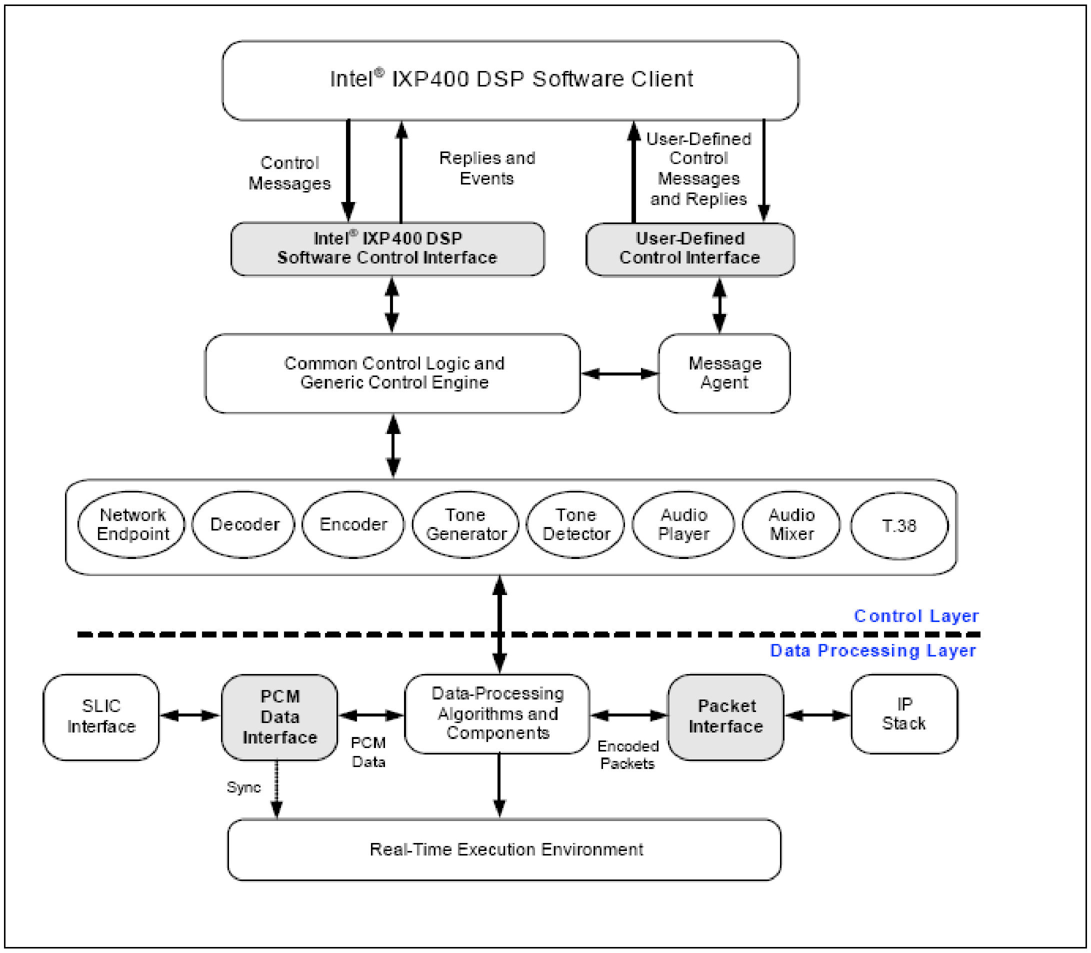

Intel® Infrastructure DSP Solution Client
Control Messages
Replies and events
User defined control messages and replies
Infrastructure DSP Solution Control Interface
User defined Control Interface
Common control logic & Generic control engine
Message Agent
Plug-in Handler
Addressable control entities of media processing resource components
Network EndPoint
Decoder
Encoder
Tone Generator
Tone Detector
Audio Player
Audio Mixer
USCI
External Media Algorithm plug-ins
EC Algo objects list
Decoder Algo objects list
Encoder Algo objects list
Control layer
Data processing layer
SLIC Interface
PCM data interface
Data processing algorithms and functions
Packet Interface
IP Stack
Sync
Real time execution environment
Indicates New modules

Figure 2. Architecture of Intel® IXP400 DSP Software Version 2.6.2


Intel® IXP400 DSP Software Client
Control Messages
Replies and Events
User-Defined Control Messages and Replies
Intel® IXP400 DSP Software Control Interface
User-Defined Control Interface
Common Control Logic and Generic Control Engine
Message Agent
Network Endpoint
Decoder
Encoder
Tone Generator
Tone Detector
Audio Player
Audio Mixer
T.38
Control Layer
Data Processing Layer
SLIC Interface
PCM Data Interface
Data-Processing Algorithms and Components
Packet Interface
IP Stack
PCM Data
Encoded Packets
Sync
Real-Time Execution Environment

2.1 Directory Structure Comparison

Version 2.6.2 software patches some of the Intel® IXP400 Software Release files located in `ixp400_xscale_sw` directory (for example, files located in `ixp400_xscale_sw/src/hssAcc` directory). Version 1.1 does not patch any IXP400 Software Release files and all the Version 1.1 software files are located in a new directory called `IDS` directory. Figure 3 and Figure 4 show the directory structure of both the software releases. Besides the directory structure, another major difference is that Version 2.6.2 is packaged in a single zip file, and Version 1.1 is packaged in five packages (Foundation Library, Codec Library, Codelets, HSS Device Driver, and SLIC Device Driver). Figure 3 and Figure 4 show the directory structure after the package is unzipped.

Figure 3. Intel® Infrastructure DSP Solution Version 1.1 directory structure

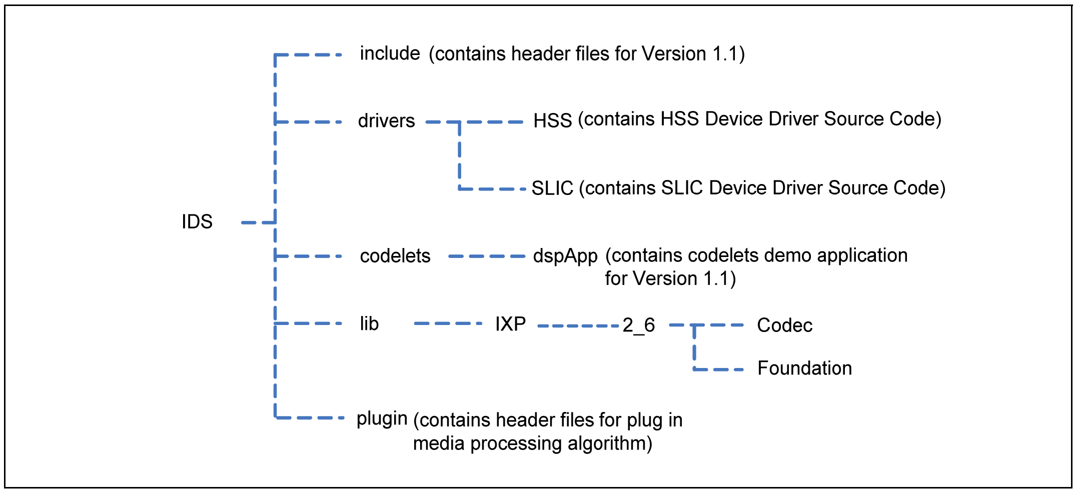
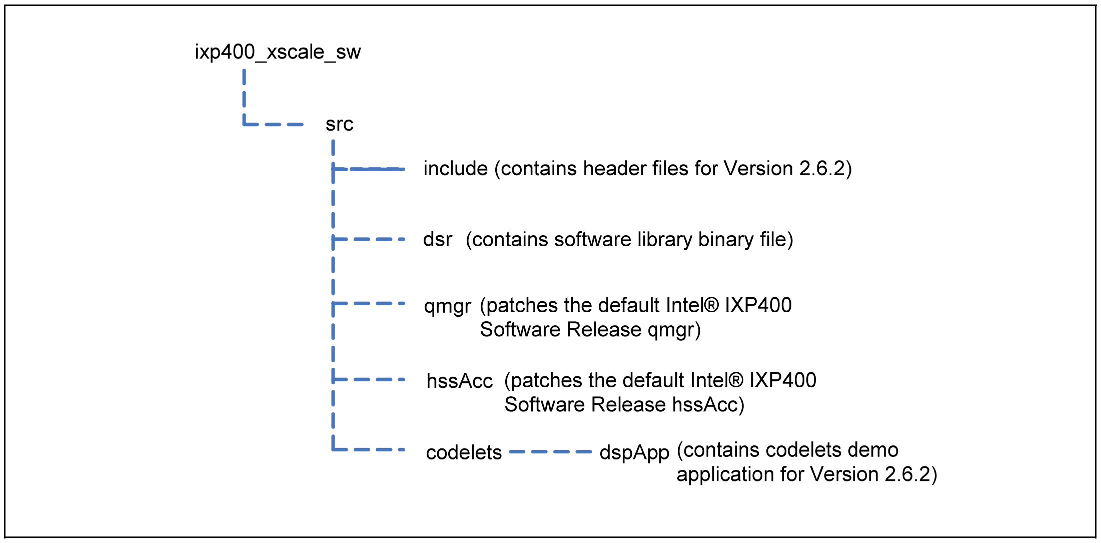

IDS
include (contains header files for Version 1.1)
drivers
HSS (contains HSS Device Driver Source Code)
SLIC (contains SLIC Device Driver Source Code)
codelets
dspApp (contains codelets demo application for Version 1.1)
lib
IXP
2_6
Codec
Foundation
plugin (contains header files for plug in media processing algorithm)

Figure 4. Intel® IXP400 DSP Software Version 2.6.2 directory structure


ixp400_xscale_sw
src
include (contains header files for Version 2.6.2)
dsr (contains software library binary file)
qmgr (patches the default Intel® IXP400 Software Release qmgr)
hssAcc (patches the default Intel® IXP400 Software Release hssAcc)
codelets
dspApp (contains codelets demo application for Version 2.6.2)

2.2 Build Process and Makefile

Due to the differences in architecture, the build process of both versions is different. Version 2.6.2 build process includes building codelets (*ixp400_codelets_dspEng.o*) and demo application (*IxDspCodeletApp*). In Version 1.1, the build process includes building HSS driver, SLIC driver, and demo applications (also known as codelets) by linking the base library and other plug-ins. For detailed instructions on building Version 1.1, please refer to *Intel® Infrastructure DSP Solution Version 1.1 Release Notes.*

You are required to modify the Makefile of Version 1.1 for adding and removing plug-ins during the build process. To add or to remove the default, Intel-provided plug-ins, please refer to *Intel® Infrastructure DSP Solution Version 1.1 Release Notes* section to edit *PlugInConfig.c* and *PlugInConfig.h* file. For example, to add external (third party) plug-ins, please refer to Appendix B of *Intel® Infrastructure DSP Solution Version 1.1 Programmer's Guide.*

Version 1.1 uses OSAL located in `IDS/lib/IXP/2_6/Foundation` directory. You need to edit the Codelet Makefile to include this library prior to building a DSP application.

3 Linux User Space and Kernel Space

One of the major differences between the two versions of software is the base library (also known as foundation library) running in either Linux Kernel Space or Linux User Space. In Version 2.6.2, the software library runs in Linux Kernel Space. In Version 1.1, the software library runs in Linux User Space.

3.1 Intel® IXP400 DSP Software Version 2.6.2 Running in Kernel Space

Version 2.6.2 demonstrates the implementation of a character device driver, called `ixDspCodeletModule` (with major number 253 and minor number 0). This device driver is for the codelets demo application from the user space to interface with kernel space software library. The operations supported by this device driver are defined in Version 2.6.2 `ixp400_xscale_sw/src/codelets/dspEng` directory `IxDspCodeletModuleSymbols.c` file.

For example, during initialization, the codelets demo application (in `ixp400_xscale_sw/src/codelets/dspEng/dspApp` directory) `IxDspCodeletApp.c` file main function calls `xInitOssl` to initialize the DSP software. Part of the initialization process includes initializing the inbound and outbound message queues to pass/receive control messages to/from Version 2.6.2. The outbound message queue is registered through the `ixDspCodeletModule` device driver (`ixp400_xscale_sw/src/codelets/dspEng/dspApp/IxDspCodeletAppOsslLib.c`).

```
outMsgQue = open("/dev/ixDspCodeletModule",O_RDWR);

... ...

rc = ioctl(outMsgQue,IX_DSP_CODELET_SETOUTMSGQUEUE, &dspConfig);
```

The “`ioctl`” function calls `ixDspCodeletModule_ioctl` function (defined in `ixp400_xscale_sw/src/codelets/dspEng/IxDspCodeletModuleSymbols.c`) to execute the case of `IX_DSP_CODELET_SETOUTMSGQUEUE` to register the outbound message queue.

With the registration of inbound and outbound queue and the definition of functions such as `xMsgWrite` and `xMsgRead` functions in `IxDspCodeletAppOsslLib.c` file, Version 2.6.2 codelets demo application is able to send/receive messages to/from kernel space.

Version 2.6.2 is able to send/receive data to/from HSS and SLIC driver without any Linux device driver implementation because in Version 2.6.2 software, HSS driver and SLIC driver run in kernel space.

3.2 Intel® Infrastructure DSP Solution v1.1 Running in User Space

Version 1.1 runs in user space. Figure 5 illustrates that codelets demo application and Version 1.1 software library is running in Linux User Mode. Hence, there is no need to specifically register the inbound or outbound queue to pass messages between kernel and user space. Figure 6 illustrates that the base library of Version 2.6.2 is running in Linux Kernel Mode. Compared to Version 2.6.2 software, the Version 1.1 codelets demo application is able to directly call software library APIs provided, such as `xMsgRead` function or `xMsgWrite` function, without registering the inbound or outbound queue using "`ioctl`" function.

Due to Version 1.1 software running in User Space, the codelets demo application does not require `ixDspCodeletModule` character device driver to pass data/messages between codelets demo application and Version 1.1 software library. However, two additional device drivers are required. The HSS driver and SLIC driver run in kernel space, as shown in Figure 5. Hence, Linux Device Drivers associated with both devices are required to be defined in order to pass data or messages between codelets demo application and the devices.

Figure 5. Intel® Infrastructure DSP Solution Version 1.1 Application in Linux

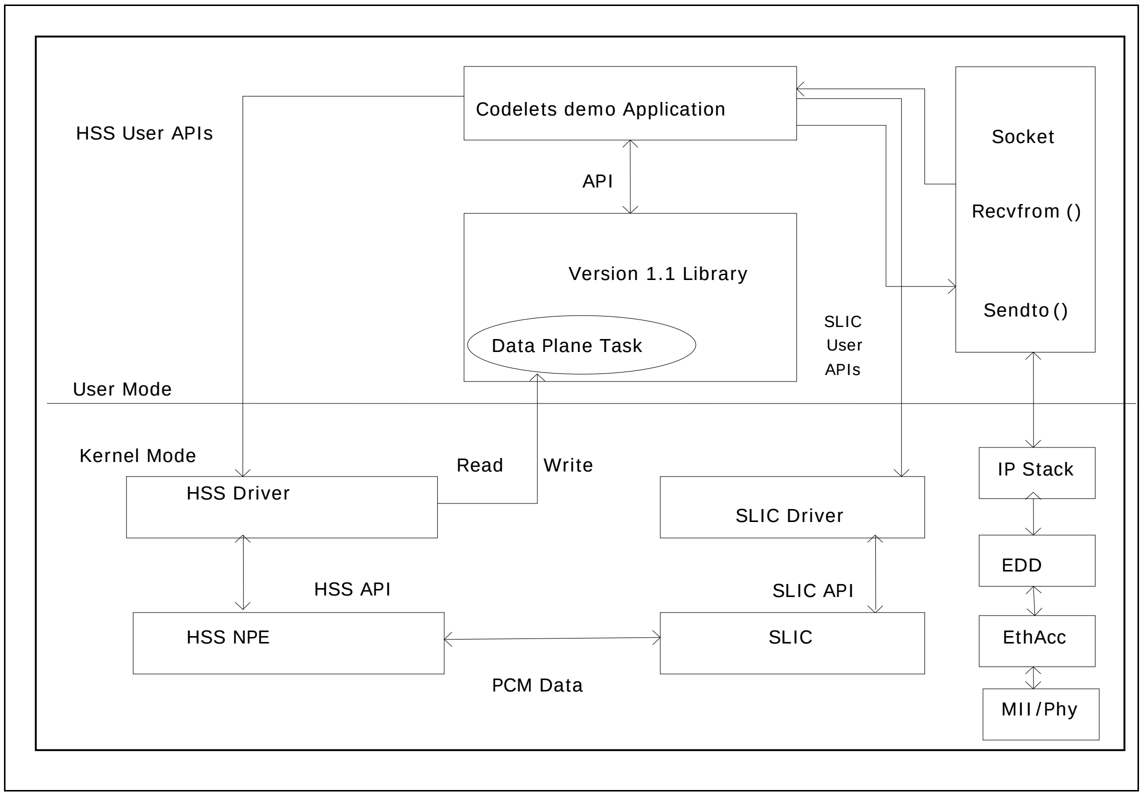
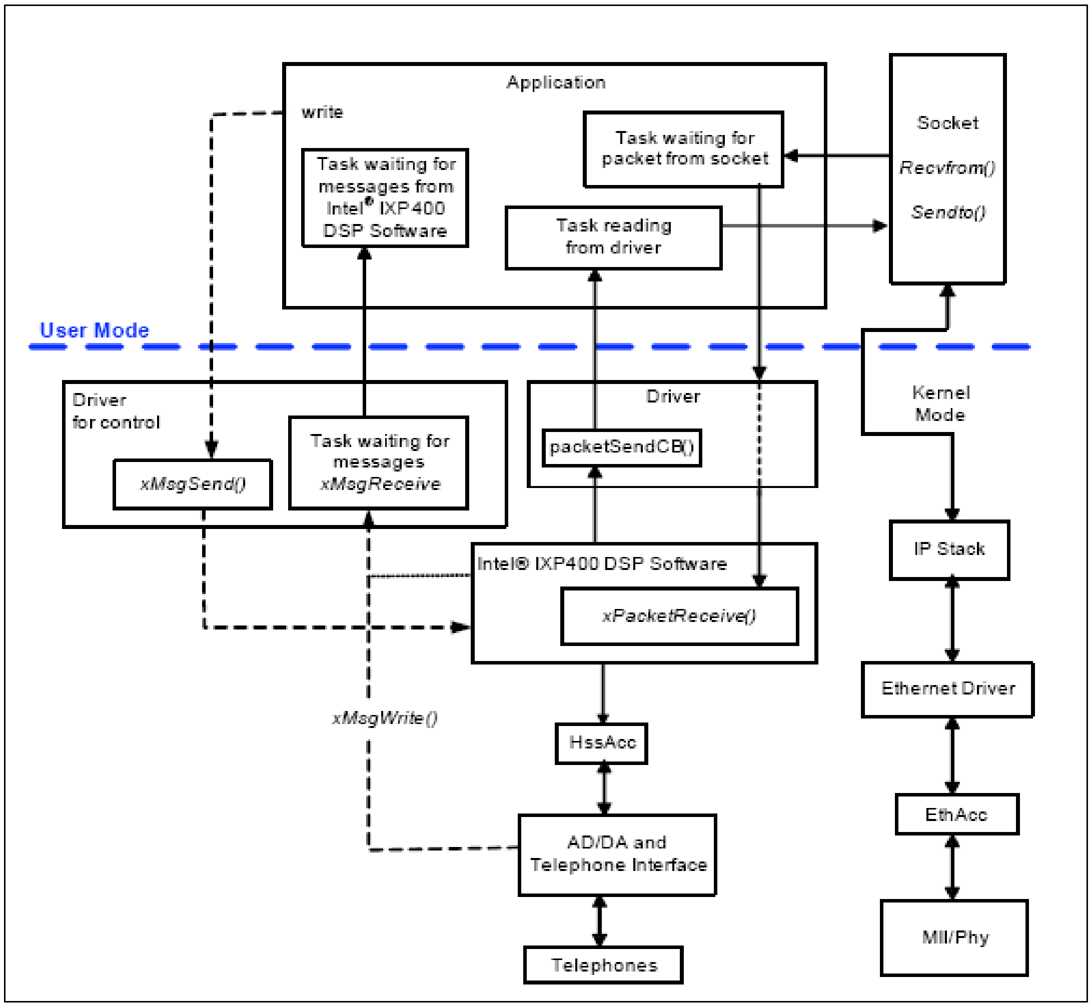


Figure 6. Intel® IXP400 DSP Software Version 2.6.2 Application in Linux



In Version 1.1, three new parameters (`taskPriReal`, `taskPriCtrl` and `taskPriPCMRead`) were added to the `pSysConfig` structure. You can configure the real-time task priority, control task priority and PCM read task priority through these new members of the `pSysConfig` structure as shown in Figure 7. The parameters `taskPriBase` and `taskPriOrder` are not used in Version 1.1; these parameters are reserved for future use.

In Version 2.6.2, you can configure the base priority for the real-time task, PCM read task and control task through `taskPriBase` and `taskPriOrder` parameters; however the priority for individual task is not configurable. Refer to Figure 8 for Version 2.6.2 `pSysConfig` Structure.

Figure 7. Intel® Infrastructure DSP Solution Version 1.1 pSysConfig Structure

```
typedef struct{
    int     numChTDM;          /* number of channels of TDM termination */
    int     numChIP;           /* number of channels of IP termination */
    int     numPlayers;        /* number of player instances */
    int     numMixers;         /* number of Audio Mixers */
    int     numPortsPerMixer;  /* number of ports per mixer */
    int     countryCode;       /* country code */
    int     taskPriBase;       /* the base priority of DSP module */
    int     taskPriOrder;      /* the priority ordering of the OS */
    int     taskPriReal;       /* realtime task priority  of DSP module*/
    int     taskPriCtrl;       /* control task priority of DSP module */
    #ifndef OS_VXWORK
    int     taskPriPCMRead;    /* PCM read task priority of DSP module */
    #endif
    IxHssAccHssPort      port;          /* HSS port */
    IxHssAccConfigParams *pHssCfgParms; /* HSS configuration parameters*/
    IxHssAccTdmSlotUsage *pHssTDMSlots; /* HSS TDM time slot mapping */
    XDSPChanTdmSlots_t   *pChanTsMap;   /* channel vs. slot mapping for WB
mode */
    XPktRcvFxn_t         pktRcvFxn;   /* packet receiver function */
    XMsgAgentDec_t       msgDecoder;  /* message decoder function of MA*/
    XMsgAgentEnc_t       msgEncoder;  /* message encoder function of MA*/
} XDSPSysConfig_t;
```

Figure 8. Intel® IXP400 DSP Software Version 2.6.2 pSysConfig Structure

```
typedef struct{
    int     numChTDM;          /* number of channels of TDM termination */
    int     numChIP;           /* number of channels of IP termination */
    int     numPlayers;        /* number of player instances */
    int     numMixers;         /* number of Audio Mixers */
    int     numPortsPerMixer;  /* number of ports per mixer */
    int     countryCode;       /* country code */
    int     taskPriBase;       /* the base priority of DSP module */
    int     taskPriOrder;      /* the priority ordering of the OS */
    IxHssAccHssPort      port;           /* HSS port */
    IxHssAccConfigParams *pHssCfgParms; /* HSS configuration parameters*/
    IxHssAccTdmSlotUsage *pHssTDMSlots; /* HSS TDM time slot mapping */
    XDSPChanTdmSlots_t   *pChanTsMap;    /* channel vs. slot mapping for
WB mode */
    XPktRcvFxn_t         pktRcvFxn;   /* packet receiver function */
    XMsgAgentDec_t       msgDecoder;  /* message decoder function of MA*/
    XMsgAgentEnc_t       msgEncoder;  /* message encoder function of MA*/
} XDSPSysConfig_t
```

4 Device Drivers

Intel® Infrastructure DSP Solution Version 1.1 does not implement `ixDspCodeletModule` character device driver (as implemented in Version2.6.2) because the software library runs in user space. HSS device driver and SLIC device driver are required in Version 1.1. In Version 2.6.2, device drivers are not required to access HSS device and SLIC device because the software library runs in kernel space.

4.1 PCM Data Interface and HSS Device Driver

PCM data represents the audio data stream between software releases and the telephone interface. Both versions of software use the TDM bus of Intel® IXP4XX Product Line HSS interface as PCM data interface. To be able to utilize HSS device, user application is required to setup the HSS interface. The codelets demo application of both versions of software show the method to configure the HSS device. However, Version 1.1 implements HSS device driver (which separates from codelets demo application) and Version 2.6.2 integrates the HSS device enabling code in base binary library file.

Table 2. Comparison of PCM Data Interface and HSS driver

Intel® IXP400 DSP Software Version 2.6.2	Intel® Infrastructure DSP Solution Version 1.1
HSS enabling source code is not located in driver directory	HSS enabling source code located in `IDS/drivers/HSS`
No Linux device driver associated with HSS driver to pass/receive data to/from codelets demo application	Linux device driver, `hssdriver` defined to pass data/messages to/from codelets demo application
Does not implement any standalone module for HSS	HSS driver is a standalone module, `hssdriver.o`

4.1.1 HSS Device Driver in Intel® Infrastructure DSP Solution Version 1.1

For Version 1.1, the HSS device driver source code is located in directory `IDS/drivers/HSS`. The HSS device driver is defined as Linux character device driver. Referring to this directory, it includes HSS driver source code for initialization function, clean-up function, open and close functions, read and write functions, input/output control functions and callback functions. All of these functions are required to build the HSS driver to be run in kernel mode.

Refer to *Intel® Infrastructure DSP Solution Version 1.1 Release Notes* section *Building HSS Driver*. Running the command '`make hssdriver`' in IDS directory will make the HSS device driver. It is required to define the HSS device driver as character device with major number 251 and minor number 0 (`mknod /dev/hssdriver c 251 0`).

With the definition of HSS device driver and inserting the device driver module into the target Linux* Operating System, the codelets demo application (IxDspCodeletApp) is able to access to the HSS device driver. The HSS device driver provides three APIs to user applications:

- IxHssDriverHssPortInit(Hss_config *port_config)
- IxHssDriverNpeBCInit(UINT32 npeB_image_id, UINT32 npeC_image_id)
- IxHssDriverNpeCInit(UINT32 npeC_image_id)

From Version 1.1 codelets demo application, the three APIs are called from `IxDspCodeletMain.c` file `demostart(void)` function where the `demostart(void)` is called from the main function entry point `main(void)` function in `IxDspCodeletApp.c`. `IxHssDriverHssPortInit` is called to download NPE A microcode image and initialize NPE A. This API also initializes and configures HSS device and connects the device with the HSS device driver. `IxHssDriverNpeBCInit` downloads NPE Images to NPE B and NPE C and starts both NPEs, whereas `IxHssDriverNpeCInit` downloads NPE Image to NPE C and starts NPE C. By default, codelets demo application does not call `IxHssDriverNpeBCInit` or `IxHssDriverNpeCInit` API because Ethernet Device Driver performs the same operation as these APIs. However, if user application does not use Ethernet Device Driver, either one of the APIs is required to be called to enable NPE B and NPE C, depending on the platform used.

Another major functionality of HSS device driver is to define the call-back function to pass PCM data between the software library running in user space and the HSS driver in kernel space. The HSS driver defines `xHssCallBack` function in `IDS/drivers/HSS` directory `HssDriverFuncs.c` file. This callback function is attached to HSS port (using `ixHssAccChanConnect` function) during HSS device initialization process. With this implementation, Version 1.1 software library gets the PCM data from HSS device and passes the PCM data from IP termination to HSS device.

4.1.2 HSS Device Usage in Intel® IXP400 DSP Software Version 2.6.2

Version 2.6.2 runs in kernel mode and it does not require the implementation of device driver to pass data between codelets demo application and the software library. There is no specific directory that contains the files related to HSS device usage for Version 2.6.2; however, in Version 1.1, the HSS device driver files are stored in `IDS/drivers/HSS` directory.

Using the NPE initialization process as a comparison, Version 2.6.2 codelets demo application initializes the NPEs directly in `demostart` function in `IxDspCodeletMain`.c by calling `ixDspCodeletNPEInit` function.

Call-back functions (such as `xHssCallBack` function) and the mechanism to pass data between Version 2.6.2 software library and HSS device are defined in the software binary file, `dsr.o`. PCM data is not passed between user space and kernel space. For example, there is no need to copy data from Version 2.6.2 software library using `hssdriver_write` function (through `copy_from_user` function) to HSS buffer in kernel space. Such implementation is not required since Version 2.6.2 runs in kernel space.

4.2 SLIC Device Driver

SLIC device driver implementation provides access of SLIC functionality to codelets demo application. Both Version 2.6.2 and Version 1.1 provide sample code to demonstrate the method to enable and utilize the SLICs on the platforms supported. Table 3 shows the differences in SLIC driver between Version 2.6.2 and Version 1.1.

Table 3. Comparison of SLIC driver

Intel® IXP400 DSP Software Version 2.6.2	Intel® Infrastructure DSP Solution Version 1.1
SLIC enabling source code located in `ixp400_xscale_sw/src/codec`	SLIC enabling source code located in `IDS/drivers/SLIC`
No Linux device driver associated with SLIC driver to pass/receive data to/from codelets demo application	Linux device driver, `ixSlicModule` defined to pass data/messages to/from codelets demo application
SLIC driver included in codelets module, `ixp400_codelets_dspEng.o`	SLIC driver is a standalone module, `ixp400_codec.o`
SLIC driver only enables up to 2 FXSs, because supported platform (ADI* Coyote* Platform) only supports 2 FXSs	SLIC driver supports up to 4 FXSs, because Intel® IXDPG425 Network Gateway Reference Platform and Intel® IXDP465 Development Platform support 4 FXSs, Intel® IXP435 Multi-Service Residential Gateway Reference Platform, SLIC driver supports 2 FXSs.
SLIC driver does not support FXO port because platform does not support FXO	SLIC driver supports FXO port, because Intel IXDP465 Development Platform and Intel IXP435 Reference Platform support FXO (Si3050 chip)

Codelets demo application Gateway and Fax Bypass Demo option is used to show the differences in SLIC driver between the two versions of software releases. When executing Gateway and Fax Bypass Demo, when dialing from one phone to another phone (for example, if user of phone 1 dials phone 2), codelets demo application is required to ring the FXS port of phone 2.

In Version 2.6.2, when the Gateway State Machine changes to RING state by invoking `ixDspCodeletGwInitRing` function (defined in `ixp400_xscale_sw/src/codelets/dspEng` directory `IxDspCodeletGw.c` file), it invokes `ixDspCodeletNormalRing` function (defined in `IxDspCodeletSlicUtil.c`). The `ixDspCodeletNormalRing` function contains six lines of code (invoking `ixScDirectRegWrite` function) to write to SLIC Direct Registers. With this procedure, SLIC will send the RING signal through the FXS interface to the analog phone and the analog phone will ring. Since both Gateway State Machine and SLIC driver run in kernel space, it can directly invoke the SLIC driver functions.

In Version 1.1, when the Gateway State Machine changes to RING state by invoking `ixDspCodeletGwInitRing` function (defined in `IDS/codelets/dspApp` directory `IxDspCodeletGw.c` file), it invokes `ixDspCodeletNormalRing` function (defined in `IxDspCodeletSlicUtil.c`). In `ixDspCodeletNormalRing` function, it calls `ixFXSODirectRegWrite` API to write to SLIC registers to ring the analog phone. Since Version 1.1 Gateway State Machine runs in user space, `ixFXSODirectRegWrite` function utilizes SLIC device driver to pass "`ioctl`" function call to kernel space. Refer to `IxSlicDriverUsrAPIs.c` file for the definition of `ixFXSODirectRegWrite` function to write to SLIC Direct Register.

```
int ixFXSODirectRegWrite(int slot, int   board, int chip,int reg, int data)

{

... ...

SlicMsgQue = open("/dev/ixSlicModule",O_RDWR);

... ...

rc = ioctl(SlicMsgQue, IX_FXS_REG_WRITE, &IxSlicBoardIdregWr);

... ...

}
```

With the “`ioctl`” function with `IX_FXS_REG_WRITE`, the codelets demo application passes the required data using `IxSlicBoardIdregWr` structure (for example which FXS slot which register) to the SLIC device driver. The `ixSlicDirectRegWrite` function defined in `IDS/drivers/SLIC` directory `IxSlicCommon.c` file writes to the specified SLIC register.

This example shows only one functionality of SLIC, which is to ring the analog phone. Similar differences can be observed by comparing the SLIC driver source code of Version 2.6.2 with Version 1.1.

5 IP Termination

For both versions of software, there is no difference between the API to send and receive packets from IP Termination and no difference in DSP Software Release Packet Data Interface. For details on the API to send and receive IP packets and Packet Data Interface, refer to *Intel® Infrastructure DSP Solution API Reference Manual*.

Table 4. Comparison of IP Termination implementation in codelets

Intel® IXP400 DSP Software Version 2.6.2	Intel® Infrastructure DSP Solution Version 1.1
Codelets directly calls Ethernet Access Layer (`ixEthAcc`)	Intel® IXP400 Linux Ethernet Device Driver is used to manage packets to/from `ixEthAcc` Access Layer component.
IP packets do not pass through Linux Operating System socket interface.	Codelets implementation shows the method to utilize Linux Operating System socket interface.
Codelets implementation does not work with Secure Real-Time Protocol (SRTP)	Codelets implementation shows the usage of Secure Real-Time Protocol (SRTP)

However, both versions of codelets demo applications demonstrate different methods to pass the packets for IP termination. Version 2.6.2 directly calls functions in the Ethernet Access Layer, `ixEthAcc` to send and receive IP packets (RTP packets). The IP packets are passed between Access Layer and DSP Software in kernel space only. This method does not route the packet through Operating System socket interface. For Version 1.1, the IP packets pass through the Linux Operating System socket interface. The socket interface passes the data between the user space and kernel space. In the kernel space, Intel® IXP400 Linux Ethernet Device Driver is used to manage the packets to/from `ixEthAcc` Access Layer component.

For example, to send IP packets from DSP software to IP termination, Version 2.6.2 software codelets defined `ixDspCodeletEthRtpSend` function (in `ixp400_xscale_sw/src/codelets/dspEng` directory `IxDspCodeletEthIf.c` file). For Version 1.1, `ixDspCodeletRtpSocketSend` function (in `IDS/codelets/dspApp` directory `IxDspCodeletSocket.c` file) is defined to send IP packets from DSP software to IP termination. For receiving packets from IP termination to DSP software, both versions of codelets demo application use `xPacketReceive` API. The difference is, in Version 2.6.2 codelets demo application registers `ixDspCodeletRtpRxCB` function (in `IxDspCodeletIf.c` file) using `ixEthAccPortRxCallbackRegister` function. In Version 1.1, `ixDspCodeletSocketReceive` function (in `IxDspCodeletSocket.c`) is registered with Operating System socket interface to receive packets from IP termination and send to DSP software. The receive socket threads are created as real-time threads.

This example shows some of the differences between the codelets demo application of both versions of software releases to handle ingress and egress IP Packets. For detailed implementation, compare `IxDspCodeletEthIf`.c for Version 2.6.2 with `IxDspCodeletSocket`.c for Version 1.1.

Codelets in Version 1.1 support Secure Real-Time Transport Protocol (SRTP). SRTP is a set of BSD-style license library to provide confidentiality, message authentication, and replay protocol to Real-Time Transport Protocol (RTP). *Intel® Infrastructure DSP Solution Release Notes* explains the steps to build codelets with SRTP. For more details on SRTP, please refer to SRTP webpage http://srtp.sourceforge.net/.