



User Guide

Linux BSP: Kernel and Ramdisk Configuration for IXA SDK 2.0

Revision **1.0**

Last Print Date:

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 products are not intended for use in medical, life saving, or life sustaining 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.

This design 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 document 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.

Copyright © Intel Corporation, 8/16/01

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

Abstract

This document describes how to customize ARM Linux and the IXA SDK for an IXP1200 based system. It provides specific details on how to build and configure the linux kernel, the ramdisk, and how to make board specific modifications to the IXP1200 linux port and the IXA SDK.

The IXA SDK has been designed such that it obtains hardware parameters at run-time from the linux kernel and auto-configures itself. This means that most components of the SDK don't need to be recompiled just to run on a different IXP1200 board. This document explains how that is accomplished and what the limitations of the current implementation are.

The document is targeted at developers attempting to customize the IXA SDK for their specific hardware.

Table of Contents

1.	Introduction	5
1.1	Purpose.....	5
1.2	Scope.....	5
1.3	Reference Documents	5
2.	Building and Configuring the Linux Kernel	6
2.1	Building the ARM Linux kernel	6
2.2	Current configuration of the linux kernel	6
3.	Building and configuring the ramdisk	8
3.1	Building the ramdisk	8
3.2	Current configuration of the ramdisk.....	8
3.3	Adding a utility to the ramdisk	9
4.	Customizing Linux for specific hardware	10
4.1	Adding a new board.....	10
4.2	Modifying the setup files	10
5.	Customizing the IXA SDK	12
5.1	Linux API for IXA SDK BSP	12
5.1.1	get_virtual_mem_map()	12
5.1.2	get_physical_mem_map()	13
5.1.3	get_virt_from_phys()	13
5.1.4	get_phys_from_virt()	14
5.1.5	get_board_config()	14
5.2	Customizing the ASL	14
5.3	Customizing the Resource Manager.....	15

1. Introduction

1.1 Purpose

This document describes how to customize the IXA SDK for a specific hardware configuration.

1.2 Scope

This document describes

- The current configuration of the linux kernel for the IXM1200 board.
- How to build and change the configuration of the linux kernel
- The current configuration of the ramdisk for the IXM1200 board
- How to build and change the configuration of the ramdisk
- The current configuration of IXA SDK components
- How to change the configuration of the various IXA SDK components.

1.3 Reference Documents

	Document Name	Revision	Doc Number
[SRM]	IXP1200 Network Processor Software Reference Manual	6/2001	278306-007
[PRM]	IXP1200 Network Processor Programmer's Reference	6/2001	278304-008
[HRM]	IXP1200 Network Processor Hardware Reference Manual	6/2001	278303-007

2. Building and Configuring the Linux Kernel

This section describes how to build and configure the linux kernel for the IXP1200. It also describes the current configuration of the linux kernel for the IXM1200 system.

2.1 Building the ARM Linux kernel

To build the linux kernel on an x86 linux host

- If it does not already exist, create the directory `/tftpboot`.
- `cd` to `$IXROOT/executive` directory, where `$IXROOT` is `/opt/ixasdk`. Copy the kernel source file `linux.src.tgz` from CD number 2.

```
cd /opt/ixasdk/executive
cp /cdrom/linux.src.tgz .
```
- Unpack the kernel source file:

```
tar -xvzpf linux.src.tgz
```
- Change to the directory with the linux kernel source:

```
cd /opt/ixasdk/executive/linux
```
- Create a configuration file:

```
make config.
```

From the displayed list of configuration options, choose the functionality appropriate to your environment. If you wish to select the default configuration shipped with IXA 2.0, either hit return on all options or, instead, type `make config < default.config` to create a configuration file `.config` in the same directory.
- Resolve dependencies in the configuration file:

```
make dep
```
- Compile the kernel to create a self extracting binary file called `zImage`:

```
make zImage
```

`zImage` can be downloaded to the IXP1200. The binary is created in `/opt/ixasdk/executive/linux/arch/arm/kernel/boot/compressed` and copied by the makefile into the directory `/tftpboot`. From this directory, the kernel `zImage` may be downloaded via `tftp` to the IXP1200 target (refer [IXPLINUXHOWTO]).

2.2 Current configuration of the linux kernel

As described earlier, the linux kernel can be configured during the build process using `make config`. For every configuration option, the makefile defines a pre-processor macro. For e.g. for the configuration option `CONFIG_ARCH_IXP1200`, the makefile defines a macro `CONFIG_ARCH_IXP1200`. The source can thus use `#ifdef CONFIG_ARCH_IXP1200` and conditionally compile the source code for the kernel. This scheme is used with all the options to turn off/on various features in the kernel.

Below we list some of the important configuration options selected for the IXP1200 port.

Configuration option	Description
CONFIG_ARCH_IXP1200	The ARM linux port supports a variety of ARM CPU's including SA110, SA 1100, ARM7 etc. This option selects the IXP1200 architecture
CONFIG_NWFPE	Turns on floating point emulation. This is required because a number of ramdisk utilities including the shell (bash) have some floating point code. The ARM has no floating point support in hardware and hence the need for software emulation.
CONFIG_SPECTACLE_ISLAND	Indicates that the IXM1200 board is being used. The list of boards can be changed by editing <code>config.in</code> in <code>executive/linux/arch/arm/kernel</code>
CONFIG_ALIGNMENT_TRAP	Turns on the kernel mode alignment handler.
CONFIG_MODULES	Turns on module support in the kernel
CONFIG_NET	Configures the networking support in the kernel
CONFIG_SYSVIPC	Includes System V IPC support
CONFIG_BLK_DEV_RAM	Includes block device support for the ramdisk
CONFIG_DEV_INITRD	Includes a ramdisk as part of the initial bootup
CONFIG_SERIAL_IXP1200_CONSOLE	Includes serial ixp1200 console device support
CONFIG_SERIAL_IXP1200	Includes serial driver for the ixp1200
CONFIG_INET	Configures INET type socket support
CONFIG_NET_ETHERNET	Configures Ethernet support
CONFIG_NET_PCI	Configures Networking PCI device support
CONFIG_EEPRO100	Includes eeepro100 driver support
CONFIG_EXT2_FS	Includes EXT2 file system support
CONFIG_NFS_FS	Includes NFS support

3. Building and configuring the ramdisk

This section describes how to build and configure the ramdisk for the IXP1200. It also describes the current configuration of the ramdisk shipped with IXA SDK 2.0.

3.1 Building the ramdisk

To build the ramdisk on an x86 linux host

- If it does not already exist, create the directory `/tftpboot`.
- `cd` to `$/IXROOT/executive` directory, where `$/IXROOT` is `/opt/ixasdk`. Copy the ramdisk source file `utils.src.tgz` from CD number 2.

```
cd /opt/ixasdk/executive
cp /cdrom/utils.src.tgz .
```
- Unpack the ramdisk source file:

```
tar -xvzpf utils.src.tgz
```
- To build the ramdisk, one needs to be logged in as root, so login as root.
- The source for the command line utilities that go into the ramdisk are located in `/opt/ixasdk/executive/utils`. Change directory:

```
cd /opt/ixasdk/executive/utils/ramdisk
```
- Type `make`. This makes all the utilities and creates a compressed ramdisk – `ramdisk_img.gz`. It also copies it into the directory `/tftpboot` from which it can be downloaded via `tftp` to the IXP1200 target.

3.2 Current configuration of the ramdisk

The ramdisk consists of the following directories at the top level

- `/dev` Contains various devices created using `mknod`
- `/etc` Contains various startup scripts
- `/var` Contains various temporary directories
- `/usr` User level directory
- `/lib` Contains various shared libraries (`.so`)
- `/sbin` Contains system binaries
- `/bin` Contain user binaries
- `/root` Root user's home directory
- `/rd` Temporary directory created by initial ramdisk
- `/tmp` Temporary directory for system utilities
- `/nfs` Mount point for nfs
- `/mnt` Generic mount point

- `/home` Home directory for non root users
- `/proc` proc directory

The ramdisk includes a set of utilities called busybox and tinylogin. These utilities provide a number of popular unix shells commands for linux. Also included are a shell (bash), gdbserver, various networking utilities such ftp, telnetd etc. The source for these is located in `/opt/ixasdk/executive/utils/`.

3.3 Adding a utility to the ramdisk

To add a utility to the ramdisk, download the source and put it into `/opt/ixasdk/executive/utils/`. Then compile the utility and test it on the target. Once it is verified as working, add it to the ramdisk build. To do this add it to the `mkramdisk` script under `/opt/ixasdk/executive/utils/ramdisk`.

4. Customizing Linux for specific hardware

This section describes how to customize the IXP1200 linux port for specific hardware.

4.1 Adding a new board

To add a new board to the supported configurations,

- Modify the file `config.in` under `/opt/ixasdk/executive/linux/arch/arm`. Currently two board configurations are supported: `CONFIG_SPECTACLE_ISLAND` (IXM1200) and `CONFIG_EVAL_BOARD`. Add your board configuration to this file.
- Modify `board.h` under `/opt/ixasdk/executive/linux/include/asm/arch`. Add board specific information to this file. The next section describes this in more detail.

4.2 Modifying the setup files

This section describes the important files that need to be modified for a specific board.

- `/opt/ixasdk/executive/linux/asm/arch/hardware.h`. This file contains the virtual memory map for the IXP1200 board. Most of the constants defined in this file do not change from board to board. So unless the user wants to change the virtual memory mapping for a specific reason, it is not necessary to modify this file.
- `/opt/ixasdk/executive/linux/asm/arch/ixp1200eb.h`. This file contains the physical memory map for the IXP1200 board. Most of these constants do not change from board to board.
- `/opt/ixasdk/executive/linux/include/asm/arch/board.h`. This file contains board specific information. The following constants are important

Macro	Description and example
<code>IX_BOARD_NAME</code>	Name of the board, e.g "SPECTACLE_ISLAND"
<code>NUMBER_OF_IX_DEVICES</code>	Number of devices on the IXPBUS. For the IXM1200 this is 3 (2 100 Bit MAC's and 1 gigabit MAC)
<code>DEVICE_TYPE_0</code>	Device type for device 0. For the IXM1200 this is <code>DEVICE_TYPE_FAST_ETHERNET</code>
<code>DEVICE_0_PORTS</code>	Number of ports for device 0. For the IXM1200 board this is 8
<code>DEVICE_TYPE_1, DEVICE_TYPE_2</code> etc	...
<code>DEVICE_1_PORTS, DEVICE_2_PORTS</code> etc	...

IX_BUS_MODE	Mode for the IXPBUS. For the IXM1200 board it is IX_BUS_MODE_32_BIT
MAX_CTRL_STORE	Size of control store. 2048 words for the IXM1200.
SRAM_BASE	Base address in virtual memory of all system SRAM.
SRAM_SIZE	Total size of SRAM
RM_SRAM_BASE	Base address in virtual memory of SRAM allocated to the Resource Manager
RM_SRAM_SIZE	Size of SRAM allocated to the Resource Manager
ASL_SRAM_SIZE	Size of SRAM allocated to the ASL
ASL_SRAM_BASE	Base address in virtual memory of uncached SRAM allocated to the ASL
ASL_CACHEABLE_SRAM_BASE	Base address in virtual memory of cached SRAM allocated to the ASL.
DRAM_BASE	Base address in virtual memory of system DRAM
DRAM_SIZE	Size of system DRAM
RM_DRAM_BASE	Base address in virtual memory of DRAM allocated to the Resource Manager
RM_DRAM_SIZE	Size of Resource Manager DRAM
ASL_DRAM_BASE	Base address in virtual memory of uncached DRAM allocated to the ASL
ASL_DRAM_SIZE	Size of uncached ASL DRAM
ASL_CACHEABLE_DRAM_BASE	Base address in virtual memory of cached ASL DRAM
ASL_CACHEABLE_DRAM_SIZE	Size of cached ASL DRAM
PHYS_FLASH_SIZE	Size of flash

All the sizes above are in bytes.

5. Customizing the IXA SDK

This section describes how to customize the IXA SDK. The various components of the IXA SDK get system configuration from the linux kernel via various API calls. This chapter describes how to use the BSP API.

5.1 Linux API for IXA SDK BSP

The following functions are supported in the IXA SDK BSP API. The API is only exported to drivers and other modules in the linux kernel. It is not available to applications. Typically these functions will only be used by IXA SDK components such as the Resource Manager and the ASL. These functions are available in the directories `/opt/ixasdk/executive/linux/include/arch/mm-ixp1200.h` and `/opt/ixasdk/executive/linux/include/arch/board.h`.

5.1.1 `get_virtual_mem_map()`

```
typedef enum {
    REQ_SRAM_PUSH_POPQ_BASE,
    REQ_RM_DRAM_BASE,
    REQ_ASL_DRAM_BASE,
    REQ_PCIMEM_BASE,
    REQ_ARMCSR_BASE,
    REQ_PCI_CSR_BASE,
    REQ_SRAM_CSR_BASE,
    REQ_SDRAM_CSR_BASE,
    REQ_MICROENGINE_FBI_BASE,
    REQ_PCIO_BASE,
    REQ_SRAM_SLOW_PORT_BASE,
    REQ_PCICFG0_BASE,
    REQ_PCICFG1_BASE,
    REQ_FLASH_BASE,
    REQ_SRAM_BASE,
    REQ_RM_SRAM_BASE,
    REQ_ASL_SRAM_BASE,
    REQ_SRAM_READ_LOCK,
    REQ_SRAM_WRITE_UNLOCK,
    REQ_SRAM_CAM_UNLOCK,
    REQ_SRAM_CLEAR_BITS,
    REQ_SRAM_SET_BITS,
    REQ_SRAM_TEST_CLEAR_BITS,
```

```
REQ_SRAM_TEST_SET_BITS,  
REQ_UNCACHEABLE_ADDR,  
REQ_LINUX_START_ADDRESS,  
REQ_NON_LINUX_BLK_START_ADDRESS,  
REQ_VMON_SDRAM_BASE,  
REQ_TOTAL_SDRAM_SIZE,  
REQ_PHYSICAL_VMON_SDRAM_BASE,  
REQ_PHYSICAL_LINUX_START_ADDRESS,  
REQ_PHYSICAL_NON_LINUX_START_ADDRESS  
} req_mem_map;
```

```
#include <asm/arch/mm-ixp1200.h>
```

```
extern unsigned long get_virtual_mem_map(int req, void ** non_cacheable_base,  
void ** cacheable_base);
```

The parameters are defined as follows:

req	A value from the req_mem_map enumeration.
non_cacheable_base	Returns the virtual address in non_cached memory.
cacheable_base	Returns the virtual address in cached memory. Valid only for ASL DRAM and SRAM; for all others, the value is zero.

This function gets the base offset in virtual memory for “req” as specified from req_mem_map. ASL packet memory may be mapped cached or uncached. Both base offsets are returned. For example, if req is specified as REQ_ASL_DRAM_BASE, then the cacheable_base points to the start of cacheable DRAM for the ASL and non_cacheable_base points to the start of non_cacheable DRAM for the ASL. For most requests, these point to the same address.

5.1.2 get_physical_mem_map()

```
#include <asm/arch/mm-ixp1200.h>
```

```
extern unsigned long get_physical_mem_map(  
int req, void ** physical_base, void ** reserved);
```

This function gets the base offset in physical memory for “req” as specified from req_mem_map and returns the physical address. The only allowed values required for this call are for ASL and RM DRAM and are listed below:

```
REQ_PHYSICAL_LINUX_START_ADDRESS  
REQ_PHYSICAL_VMON_SDRAM_BASE  
REQ_PHYSICAL_NON_LINUX_START_ADDRESS
```

5.1.3 get_virt_from_phys()

```
#include <asm/arch/mm-ixp1200.h>
```

```
extern unsigned int get_virt_from_phys(unsigned int physical_addr, int
cacheable_flag);
```

This function gets the virtual address corresponding to the specified physical address. It is valid only with DRAM; for SRAM, it is assumed that the whole SRAM is mapped contiguously. The `cacheable_flag` is valid only for ASL DRAM. If the `cacheable_flag` is set to 1, then the virtual address in cached memory is returned. If the `cacheable_flag` is set to 0, then the virtual address in non-cached memory is returned.

5.1.4 `get_phys_from_virt()`

```
#include <asm/arch/mm-ixp1200.h>
extern unsigned int get_phys_from_virt(unsigned int virtual_addr);
```

This function gets the physical address corresponding to a specified virtual address. It is valid only with DRAM; for SRAM, it is assumed that the whole SRAM is mapped contiguously.

5.1.5 `get_board_config()`

```
typedef struct __board_config {
    char    board_name[64];
    int     number_of_ix_devices;
    int     ix_device_types[MAX_NUMBER_OF_IX_DEVICES];
    int     ix_ports_on_device[MAX_NUMBER_OF_IX_DEVICES];
    int     bus_mode;
    int     max_ctrl_store;
} ixp_board_config;

/* Call to get the board configuration structure from linux - returns -1 if
error
*/
#include <asm/arch/board.h>
extern int get_board_config(ixp_board_config *config);
```

This function returns the board configuration containing information about how the IXBUS and various devices on the IXBUS are configured.

5.2 Customizing the ASL

The ASL uses the IXA SDK BSP API at initialization time to configure itself. The configurable items include the following:

- Size of SRAM/DRAM (ASL_SRAM_SIZE, ASL_DRAM_SIZE) memory used by the ASL -- used for packet buffers and other data structures maintained by the ASL
- Offset in virtual memory to the cached SRAM/DRAM (ASL_CACHEABLE_SRAM_BASE, ASL_CACHEABLE_DRAM_BASE) area used by the ASL
- Offset in virtual memory to the uncached SRAM/DRAM (ASL_SRAM_BASE, ASL_DRAM_BASE) area used by the ASL

These can be changed by modifying the file

`/opt/ixasdk/executive/linux/include/asm/arch/board.h`.

Note that the ASL keeps a cached and an uncached virtual mapping pointing to the SAME physical memory. This is done for SRAM and DRAM.

5.3 Customizing the Resource Manager

The Resource Manager uses the IXA SDK BSP API at initialization time to configure itself. The configurable items include the following:

- Size of SRAM/DRAM (RM_SRAM_SIZE, RM_DRAM_SIZE) memory owned by the Resource Manager. This memory is used by MicroACES for various data structures for e.g. route tables, hash tables etc.
- IXBUS configuration parameters for e.g. whether the bus is configured in 32 bit mode or 64 bit mode
- IXBUS devices available on the bus. For e.g. for the IXM1200 system, the number of devices is 3, 2 Octal MAC devices with 8 100 Mbit ports each, and 1 IXF1002 device with 2 gigabit ports.

These can be changed by modifying the file `/opt/ixasdk/executive/linux/asm/arch/board.h`

In addition, the Resource Manager uses the BSP API to get virtual memory offsets for various IXP1200 CSR registers. This allows the virtual memory map to be changed without the need to recompile the Resource Manager.

Note that all the memory used by the Resource Manager is uncached.