Dialogic® Open System Release

Linux Device Drivers

High Level Architecture Document

TABLE OF CONTENTS

1. Introduction

This document is intended to provide the users of the Dialogic® Open System Release driver source code, a high level understanding of the Linux device drivers provided on the various media/telephony platforms. The device driver architecture will be covered for the Springware, DM3 and IPT (PMAC) platforms.

1.1 Scope

This document will provide a high level overview of the Linux drivers supported under the Dialogic® Open System Release project. An architectural overview of each driver is outlined and includes descriptions of the internal components of the drivers themselves as well as a communication flows with surrounding components. Some additional capabilities are covered including tracing and diagnostics for some of the drivers.

1.2 Glossary

Term or Acronym	Description
LiS	Linux Streams
Springware	Dialogic® older legacy product line
DM3	Dialogic Media Architecture Generation III; the newer line of Telephony/Media products
PMAC	Packet Media Access Card; Internal name for the IPT series of boards.
MDI	Mercury Driver Interface (DM3 Driver library)
LPDI	PMAC Driver Interface
IOCTL	Input Output Control Command
WW	Dialogic Werewolf Protocol
DPC	Deferred Processing/Procedure Call
SRAM	Shared Random Access Memory
I2O	Intelligent Input/Output; hardware specification for offloading I/O processing from the CPU

1.3 Revision History

Revision	Author	Reason for Changes
0.1	Dialogic® System Release team	Initial draft – 03/03/2006

2. What is a Device Driver

A Device Driver is a well defined interface to a specific piece of hardware. It is written to isolate the user from the operating system and to abstract the under lying hardware. A Linux device driver is presented as a file to the user. The driver can expose some, few or none of the file operations (open, close, read, write, etc) to the user. These operations are called user entry points to the driver (user interface). There are no other ways to access the driver, except through these entry points.

A driver is required to follow the operating system's supported driver interface. This interface allows the driver to execute in the operating system's kernel mode. Running in this mode allows the driver to use the kernel's hardware interface to control the hardware.

A driver can be written in ways where it does not require a piece of hardware, yet run in kernel mode. This type of driver is known as a kernel module. Since the driver runs in kernel mode, it is capable of crashing the system. A kernel panic (segmentation faults) is the Linux equivalent to a blue screen in Windows.

For in-depth information on Linux Device Drivers refer the *Linux Device Driver, 3rd Edition* distributed by O'Reilly Publications.

2.1 Springware Linux Device Driver

The Springware device drivers support the ability to run on the older Springware legacy platform. These are STREAMS based and use STREAMS functionality provided by LiS. Unlike DM3 or PMAC, a common library interface to access these drivers does not exist which makes their removal difficult. All User Interfaces are provided by LiS, based on the STREAMS protocol.

The Springware drivers are comprised of the following driver/modules:

- DLGN – Dialogic Generic Driver
- DVBM – Dialogic Bulk Data Module
- GPIO – Generic I/O Driver
- GNCFD – Generic Configuration Driver
- SCTMR – Timer Module

2.1.1 DLGN

The DLGN is the main driver which controls the firmware interfaces. It follows the Springware Protocol to communicate with the firmware. There is no DPC or timer in this driver and its use has not been required since SRAM access is done quickly enough in the messaging via the channel and device SRAM areas. No other processing is required in this driver. DLGN will read or write a block of data from and to the host runtime library or the Bulk Data Module.

2.1.2 DVBM

The Bulk Data Module buffers media data from and to the DLGN driver. It is configured by the host runtime libraries (which can be configured by the application via setparm API). The configuration sets the flow of data between DLGN and host.

2.1.3 GPIO

The Generic I/O driver is used to download the firmware onto the board. This driver interfaces with the GPIO Driver library (libgpio.so) and is utilized by the GENLOAD application during the start of all the Dialogic® services. The driver locates boards in the system by providing the physical address, bus, slot and other information to GENLOAD. This is used by gpiodebug for debugging the memory and I/O devices.

2.1.4 GNCFD

The Generic Configuration driver interfaces with the GNCFD Driver library (libgncf.so) and is used to setup logical devices for the Springware boards. The driver is invoked after firmware download via the VOXCTL application. After the logical devices are created (dxxxB1, dxxB1C1, etc.), the driver will start the board (enable interrupts). VOXCTL will create the /usr/dialogic/cfg/.voxcfg file which contains all the logical device information given by the GNCFD driver.

2.1.5 SCTMR

The Timer module is used by the Global Call libraries to keep time in kernel mode. No other functionality is performed by this driver. It has no interaction with the board or the other drivers.

2.1.6 Message Tracing Capability

The Springware driver has message tracing capabilities useful for debugging. This feature is not optimized and thus restricted by the Linux kernel logging mechanism. By default tracing is disabled, but can be enabled via the following two parameters:

- ft_pc: enables PC commands i.e. the tracing of host ⇔ firmware commands.
- ft_dl: enables DL commands i.e. the tracing of firmware ⇔ host commands

Both the above parameters take a setting “0” to trace all commands, or a non-zero value to enable the tracing of a particular class of messages (e.g. ft_pc = 2 will filter on all Play commands going down) The list of supported commands is available in the file /usr/dialogic/inc/d40low.h.

To actually modify these parameters, one needs to edit the /usr/dialogic/bin/drvload script file find the location where the streams-dlgnDriver.o is called and add the above parameters on the command line.
For e.g. **/sbin/insmod -f streams-dlgnDriver.o <current parameters> ft_pc=<set> ft_dl=<set>**

The print statements go to the system console and system log /var/log/messages. Also, the device area commands from the firmware are not enabled, because they will interfere with system logging. This can be enabled easily by editing the DLGN driver.

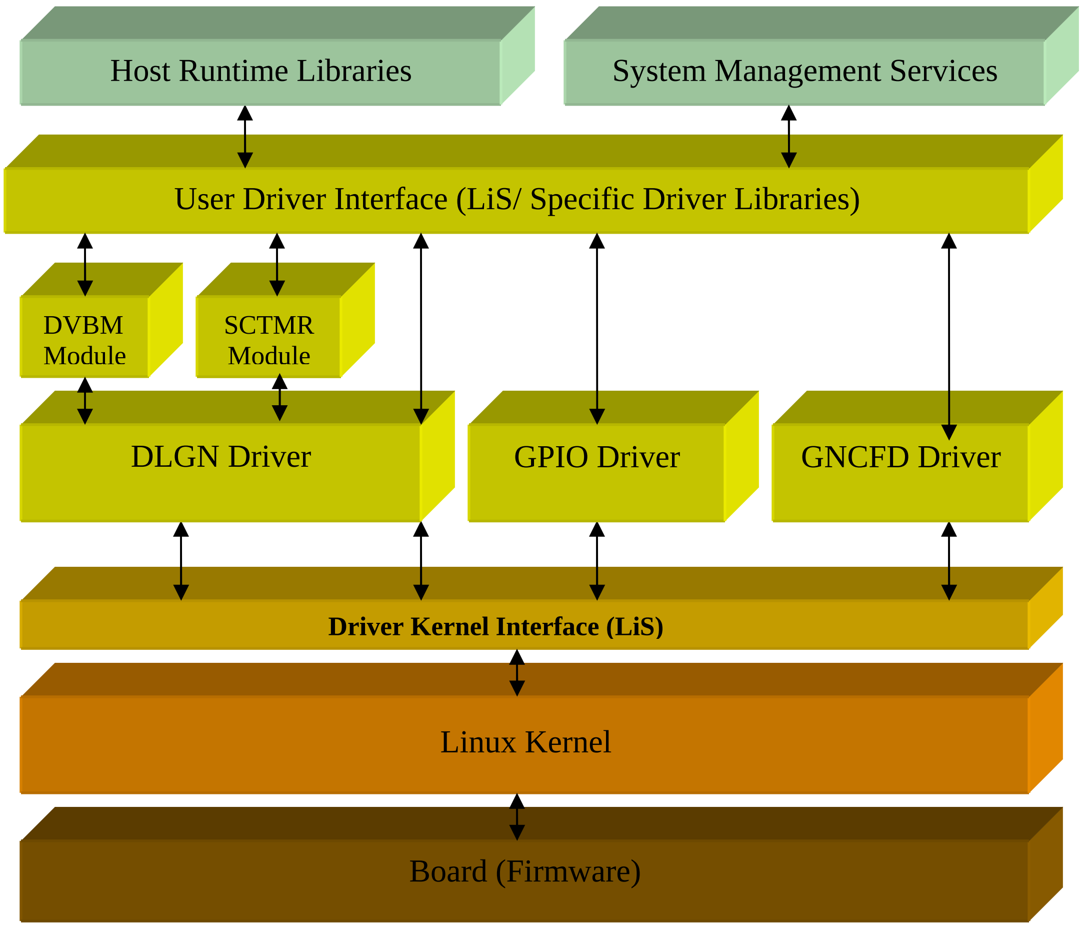
Host Runtime Libraries
System Management Services
User Driver Interface (LiS/ Specific Driver Libraries)
DVBM Module
SCTMR Module
DLGN Driver
GPIO Driver
GNCFD Driver
Driver Kernel Interface (LiS)
Linux Kernel
Board (Firmware)

Figure 3: Springware Drivers / Modules and surrounding communication flow.
Note: Each Springware driver or module contains a user and driver interface and is not shown in the diagram.

2.2 DM3 Linux Device Driver

The DM3 driver (mercd) is a character based device driver. This was originally designed as STREAMS based driver and used third party software LiS (Linux STREAMS). The native port was done without having any impact to host layer (e.g. DM3 driver library or host runtime libraries). There will be traces of the STREAMS protocol in the current driver. Additionally, a STREAMS version of the DM3 driver can be built by defining a LiS flag in the Makefile. This is not recommended since later versions (i.e. from Dialogic® System Release 6.0 onwards) of the LiS based DM3 driver have not been tested and hence not officially supported.

Mercd is based on the Dialogic Media III (a.k.a. DM3) Architecture. The driver comprises of both the SRAM protocol (used in the older products like DMVA) and the Werewolf (WW) protocol (used in the newer platforms like the DMVB/DMN160 boards) merged into one driver. Figure 1 shows the Mercd DM3 driver, its components and the surrounding communication flow.

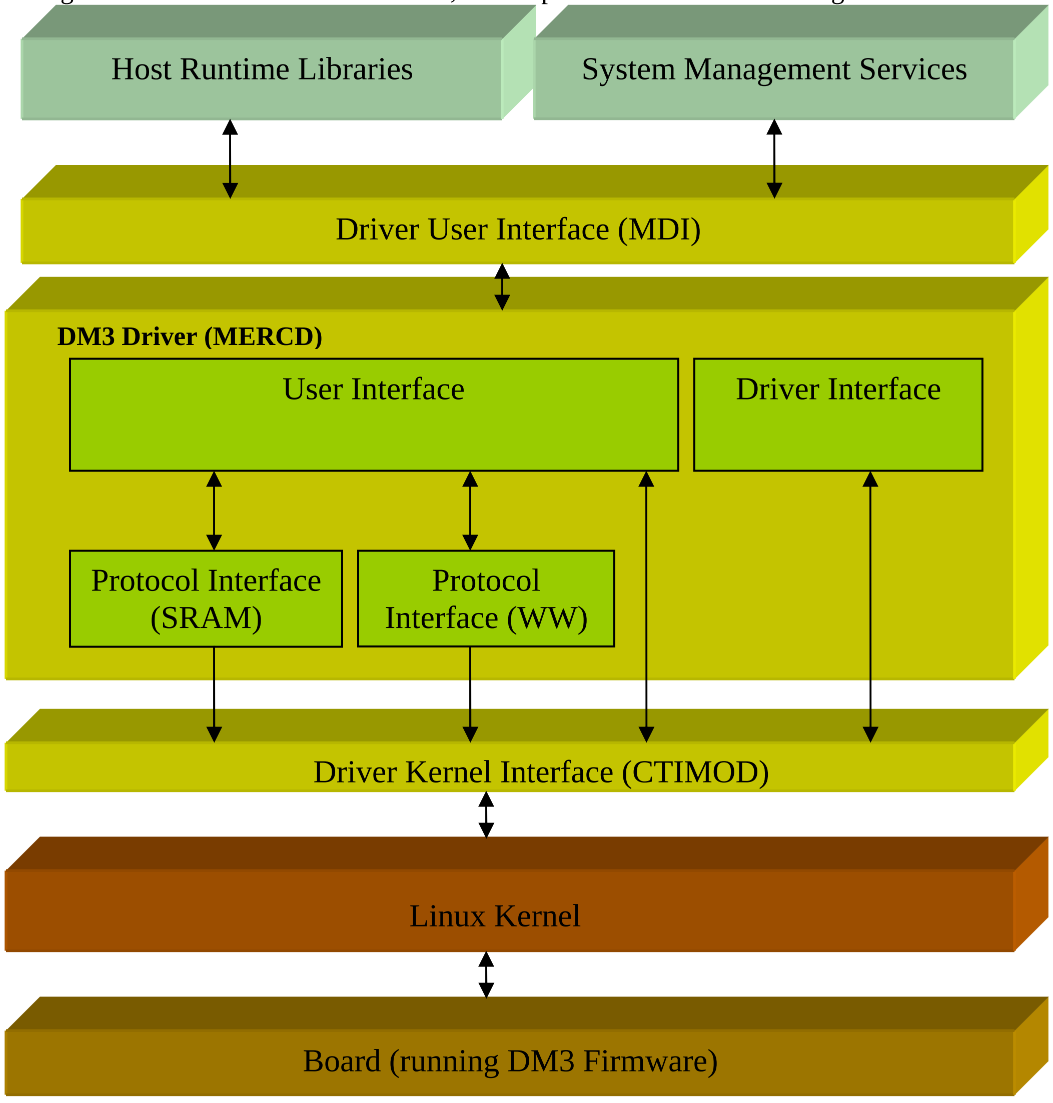
Host Runtime Libraries
System Management Services
Driver User Interface (MDI)
DM3 Driver (MERCD)
User Interface
Driver Interface
Protocol Interface (SRAM)
Protocol Interface (WW)
Driver Kernel Interface (CTIMOD)
Linux Kernel
Board (running DM3 Firmware)

Figure 1: DM3 driver and surrounding communication flow

The Mercd driver consists of three main components:

1. User Interface providing the following operations
    - Open

  - Close
  - Ioctl
  - Read
  - Write
2. Driver Interface providing the following operations
   - Load
   - Unload
   - Probe
   - Resume (for hot-swap)
   - Suspend (for hot-swap)
3. Protocol Interface providing the following operations
   - Messaging (sending and receiving)
   - Streaming (sending and receiving)
   - Interrupt Handling (via DPC, Deferred Processing Call)
   - Timer Scheduling (via tasklets)

The User and Driver Interfaces are like other Linux character drivers and follow the same standards. But in order to protect the driver, MDI (DM3 driver library) is the only interface allowed to access the driver. One can bypass the MDI interface and directly access the driver (via file handle /dev/mercd), and invoke all the available user interfaces but the driver will reject them. The resume and suspend operation within the Driver Interface are for hot swap Linux kernels.

The Prot ocol Interfaces are the main part of the driver and are primarily used on a running system. The messaging interface sends messages to the DM3 firmware. The driver sends messages which are known only to the DM3 firmware and host runtime libraries.

A driver trace utility (drvtrace) intercepts these messages and outputs them to the screen. This utility is useful in debugging and message tracing. The streaming interface is a Dialogic developed protocol used to send and receive media data. It has been developed to keep latency to an acceptable threshold. The interrupt processing is executed via a DPC. This reduces the data transfer time for the host and firmware since the driver does not hold up the system in an interrupt. The final part of the driver is the timer scheduling interface. A constantly running timer task helps driver (and system) send and receive data/messages efficiently. The timer runs every 30 milliseconds based on the protocol.

2.3 IPT Linux Device Driver

The Packet Media Access Card (PMAC) driver is a straightforward driver for the IPT family of boards based on Intelligent Input Output (I20) technology. It provides all the driver interfaces, namely the user, driver interfaces. This driver only supports messaging based on the IPT protocol and Streaming is not supported on the IPT boards. This makes the PMAC driver simple and efficient since there is not much processing required at the driver level. The driver uses DPC and a 40 millisecond timer.

The driver library (LPDI) is the interface for the User to interact with the driver. There is no tracing facility for the driver because IPT messages are understood only by the IP host runtime library and IPT firmware.

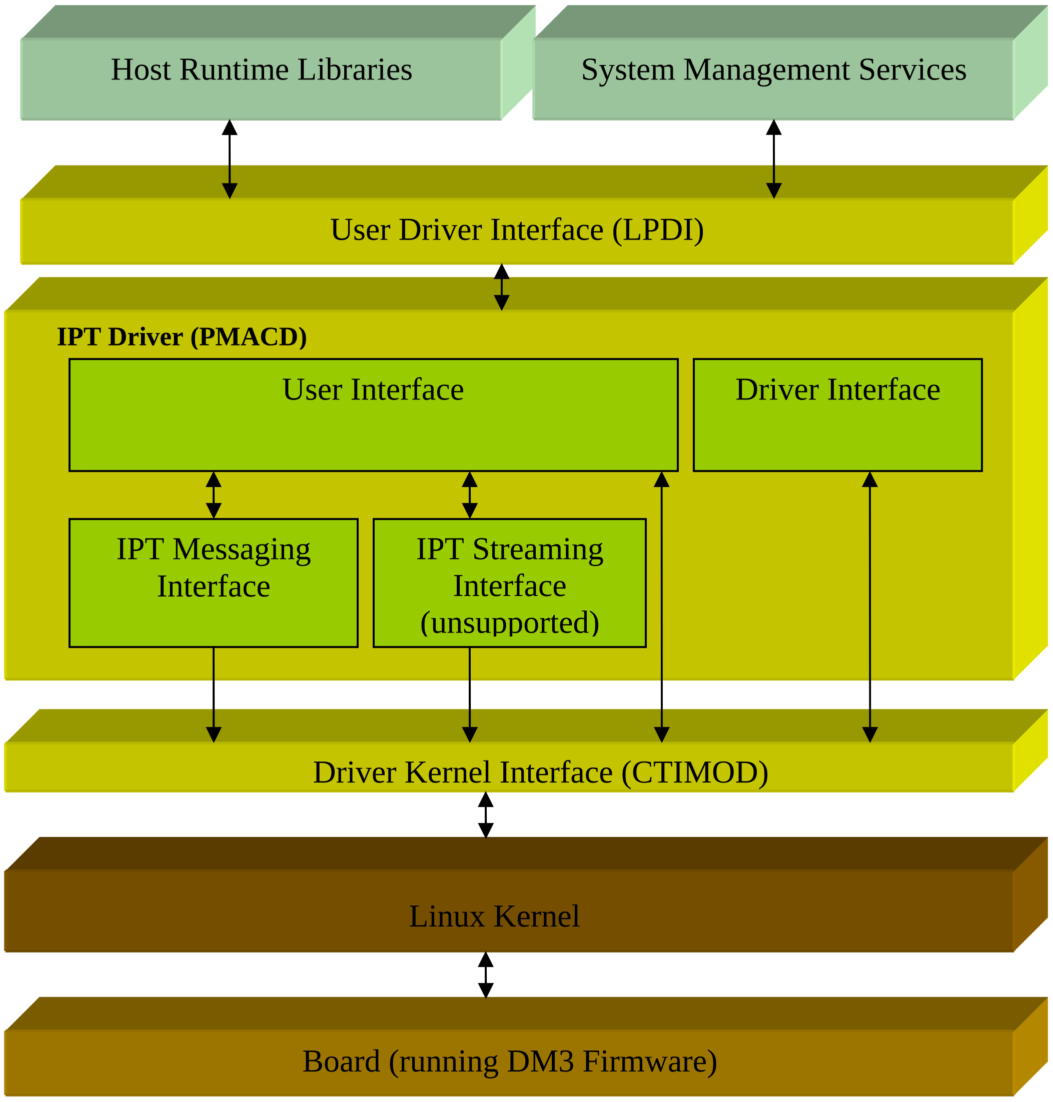
Host Runtime Libraries
System Management Services
User Driver Interface (LPDI)
IPT Driver (PMACD)
User Interface
Driver Interface
IPT Messaging Interface
IPT Streaming Interface (unsupported)
Driver Kernel Interface (CTIMOD)
Linux Kernel
Board (running DM3 Firmware)

Figure 2: PMAC driver and surrounding communication flow

2.4 Driver Kernel Interface Module (ctimod)

Ctimod provides an interface to kernel level services through a thin layer of abstraction calls. This is available only on DM3 and PMAC drivers and unlike LiS (Linux STREAMS), does not force client drivers (like PMAC, DM3) to employ specific device driver techniques (native vs. STREAMS).

This is a very important layer when loading and using pre-compiled binaries on different Linux kernel versions. In addition, ctimod provides a simple level of memory leak checking, spinlocks debugging, as wells the flexibility to add new OS features. Ctimod also provides a quicker port to new versions of the Linux kernels on different Linux Distributions without having to change the DM3 or PMAC drivers. Ctimod isolates the drivers from the Linux kernel interfaces.