TERIDIAN SEMICONDUCTOR CORP.

Simplifying System Integration™

73M1822/73M1922
Control Module User Guide

December 23, 2009
Rev. 1.0
UG_1x22_053

Teridian Semiconductor Corp., 6440 Oak Canyon, Suite 100, Irvine, CA 92618
TEL (714) 508-8800, FAX (714) 508-8877, http://www.teridian.com

Table of Contents

Figures

Tables

1 Introduction

This document describes the capabilities of the 73M1822/73M1922 Control Module. This driver software is provided for use and integration by Teridian customers on their individual platforms. The intention of this Control Module is to provide a customizable framework that is independent of processor and operating system.

Throughout this document the 73M1x22 Reference Device Driver will be simply referred to as “driver” or “device driver”. The 73M1822 and 73M1922 will be collectively referred to as the 73M1x22.

1.1 Purpose and Scope

The 73M1x22 Control Module provides the necessary system interfaces for the control and management of the 73M1x22. The driver supports API calls from the application and translates these to and from the device. The driver can be used as is, in whole or in part, or customized to accommodate a customer’s unique environment.

The scope of this document includes discussion of driver’s architecture and design, interface to the user application, the driver internal state machine, and the interface to the 73M1x22 hardware module.

The driver model is intended to be independent of processor and operating system. Layers above the Control Module address software interfaces which may pre-exist for a given application and the layer below addresses hardware related interfaces between the processor and the 73M1x22 devices.

1.2 Conventions Used in this Guide

This document uses the following conventions:

- Software code, IOCTL names, modem events, data types, and Linux® commands are presented in Courier font.
- A table with a blue header is a summary table. A table with a gray header is a detail table.

1.3 Acronyms

MAFE – Modem Analog Front End
DAA – Data Access Arrangement
IOCTL – I/O Control
ISR – Interrupt Service Routine
BSP – Board Support Package
GPIO – General Purpose Input/Output
POH – Phone Off Hook
NOPOH – No Phone Off Hook
APOH – Another Phone Off Hook

2 Overview

2.1 Driver Architecture

The driver provides a framework by which applications can leverage the features of the chipset. The main interface of the driver (IOCTLs) provides an abstraction layer for monitoring and control of the device status. Figure 1 depicts the general driver architecture.

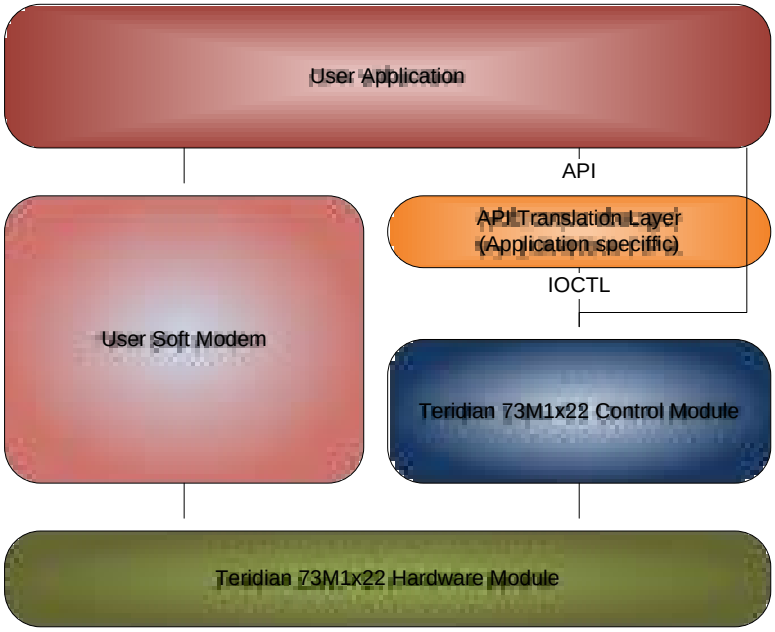

Figure 1: General Driver Architecture

Device status is analyzed and processed by an independently running process based on predefined algorithms. When an event or an even sequence is recognized, the driver posts the corresponding event to be retrieved by the user application. Figure 2 depicts the driver functional block diagram.

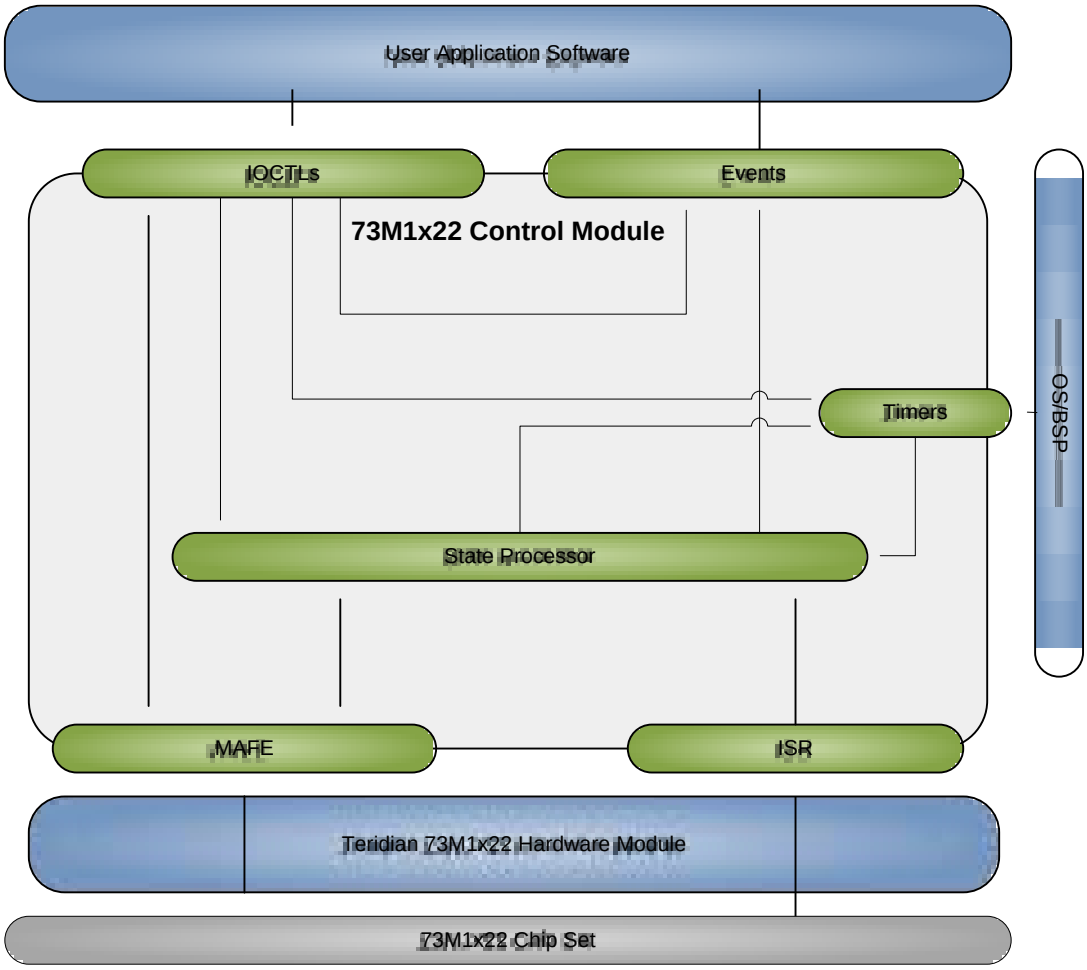

Figure 2: Driver Functional Block Diagram

The driver provides an interface to user application through the IOCTL and Events interface. Through this interface, the user application communicates with the driver via a standard device interface (`open`, `close`, `select`, `ioctl` functions). Using this interface the application will be able to control the operation of the device and to retrieve the status of the modem line. Section 6 describes the details of this interface.

The 73M1x22 driver is built as a Linux loadable module (for Linux) or integrated with the operating system kernel. It will be brought into operation by a user application or by an operating system start up script. For Linux, the `insmod` command is used to insert the driver into the kernel. The `insmod` command invokes the `module_init()` macro, which in turn runs the one-time initialization function of the driver. Before exiting the initialization the driver enters its main operational state via the scheduling of one of many timers that make up the driver's main processing.

2.2 Functional Overview

Once installed, the driver is a self-contained module running independently along with the kernel processes. Its main purpose is to monitor the modem line for various conditions, generates appropriated event when they occur, and to provide access to the 73M1x22 device for management purposes, via standard driver access methods such as `open`, `close`, `select`, `ioctl`, etc. The following sections provide an overview of that functionality.

2.2.1 Event Generation

During operation the driver constantly monitors the line for the following conditions:

1. APOH condition.
2. Transition back to no APOH condition (NOPOH).
3. Ring start detection.
4. End of Ring condition.
5. Battery disconnected condition.
6. Battery restored condition.
7. Polarity reversal condition.
8. Line State condition – monitoring of voltage and current.
9. Synchronization loss condition.
10. Synchronization recovered event.
11. Over voltage, Over current, or Over load condition.
12. Pulse dial complete event.
13. Pulse dial aborted event.

Section 5 provides detail descriptions of each event.

If any of these conditions occur the driver creates an event entry in the event queue and notifies the application layer via file descriptor ready mechanism (or via file descriptor `select()` function). Upon receiving this "wakeup" notification, the application can then test the file descriptor "ready" status with the `FD_ISSET` macro to confirm, and then retrieve the event from the driver via IOCTL event `get` command.

2.2.2 Modem Channel Configuration and Management

Configuration of the modem parameters and other management command such as hook switch operation are done via standard device driver IOCTLs. The 73M1x22 Reference Driver provides an extended list of IOCTLs for this purpose. The IOCTL command descriptions in Section 6 provide details of how they work.

2.2.3 Line State Analysis via Current and Voltage Measurements

The 73M1x22 Control Module can be programmed to provide extended line status information and line monitoring capability. The driver operates autonomously, under the application layer control, to detect line condition specified in pre-set criteria and automatically report status change. This alleviates the burden of constant polling from the application layer. The line status consists of the line voltage and line current measurements. Each measurement entity is operated independently as described below.

2.2.3.1 Interval and Even Table

The Interval and Event Table (IET) is a table that consists of multiple rows of the following information:

1. Row number.
2. Lower bound threshold.
3. Upper bound threshold.
4. Application defined event.

The application is responsible for building up the IET using the IET table update IOCTL – `M1X22_MEASURE_UPDATE`. This IOCTL can be used to create/update or to read back the current table entry (see Section 6.6.3 for details).

2.2.3.2 Measurement Procedure

The driver is responsible for reading the raw value required at a given sample rate from the device. The result is computed and averaged over a specified number of those previously read values, and then compare to the interval and event table (IET) described below. Based on these comparisons dedicated events may be sent to the application.

The driver compares the calculated average of the reading value with the lower and upper bound threshold in each IET table entry. If the value falls in between the ranges and it is the first transition into these new ranges the specific application defined event (4) will be sent to the application. The lower and upper bound ranges are expressed in milliamps for current intensity, or in volts for line voltage.

2.2.3.3 Management of the Procedure

Controlling the operation of this current/voltage measurement is exclusively done by the application layer. The application layer uses the `M1X22_MEASURE_START` and `M1X22_MEASURE_STOP` to start and stop the measurement, respectively. The simply stops the measurement and requires no additional parameter, while `M1X22_MEASURE_START` starts the measuring process and it consists of two parameters:

1. Sample time interval.
2. Average sample count.

The sample time interval is the time interval between two consecutive reading samples expressed in milliseconds, and the average sample count is the number of reading samples to be used for average calculation. Any one of these parameters can be ZERO indicates a no change. The application layer can adjust one or both parameters anytime using `M1X22_MEASURE_START`.

2.2.3.4 Even Handling

The driver emits the `M1X22_LINE_STATE` event when line state condition changes from one interval to another. The application is expected to receive the notification and can request for the event from the driver using the `M1X22_GET_EVENT` IOCTL. Refer to Section 5 for details on how to get notified and to retrieve the event from the driver's queue. The following supporting data will accompany the line state event:

1. The state of the modem channel – on or off-hook.
2. Line current or line voltage.
3. IET row index.
4. Application defined event.

2.2.4 GPIO Support

Four General Purpose I/O pins (GPIOs) pins can be managed independently and used for carrying input or output signal to and from the 73M1x22 device. If used as input, signal transition on the pin can be detected and trigger interrupt to the host CPU. The driver provides the ability to program each GPIO pin as input or output port, the ability to read and write data to the GPIO pin as well as generating interrupt event correspond the signal transition. Section 6.7 describes the GPIO related IOCTL.

Note: The GPIO feature exists only on certain 73M1x22 device packages.

2.2.5 Loopback and Testing Modes

The 1x22 devices support several variations of loopback modes. Refer to the "Loopback and Testing Modes" section of the *73M1822/73M1922 Data Sheet* for more detail. Each loopback mode is designed to test connectivity at various points in the system. Systematically use of the loopback feature in conjunction with external application that control data stream in and out of the system can be an effective tool to isolate faults. While the driver provides IOCTLs to manage those loopback test points, it does not have a way to inject or intercept data flow through the system to perform diagnostic. It relies on external application for those capabilities. The IOCTL for managing the loopback can be found in Section 6.8.1.

2.2.6 Call Progress Monitor

The 1x22 device provides the ability to monitor activity on the line via feature called the Call Progress Monitor. The gain setting of its audio path can be adjusted using the `M1X22_CPROG_MONITOR` IOCTL detailed in Section 6.9.3. For more detail on this subject, refer to the "Call Progress Monitor" section of the *73M1822/73M1922 Data Sheet*.

2.2.7 Billing Tone Filter

Some countries use a large amplitude out-of-band tone to measure call duration and to allow remote central offices to determine the duration of a call for billing purposes. To avoid saturation and distortion of the input caused by these tones, it is important to be able to reject them. These frequencies are typically 12 kHz and 16 kHz. Refer to section "Billing Tone Rejection" in the *73M1822/73M1922 Data Sheet* for more detail.

To enable or disable the billing tone filter, the driver offers the `M1X22_BTONE_FILTER` IOCTL. Its description can be found in Section 6.9.2.

3 Driver Service Interface

The Driver Service provides the link between the modem device and the user application. First, the driver must be loaded and bound into the operating system environment before this service can be provided. Access to the driver is done via two file descriptors – the device and channel file descriptors. The device file descriptor provides access to device level management interface while the channel descriptor is used to manage at the channel level interface. The driver supports multiple modem channels through separated channel descriptors; however, only one device descriptor is used.

The following sections describe how the driver is brought into action based on the operating system environment.

3.1 Linux Operating System

This description is valid for Linux 2.4 and 2.6. The 73M1x22 driver takes the form of a Linux standard character device driver. It is brought into operation by a user application or by Linux startup script using the `insmod` command. This command inserts the driver module into the kernel which in turn registers with the kernel using the default major number of 251. Multiple modem channels are supported via the use of minor number which can varies from 0 to 16. This minor number associated with the device and channel descriptors created using mknod command. The driver expects the minor number 0 to be associated with the device descriptor and the number from 1 to 16 with the channel descriptors. Device major and minor numbers are configurable at build time as described in Section 8.2.

The device and channel descriptors can be created in the `/dev` directory at the same time when the driver is insmod into the kernel. The `mknod` command is used to create those descriptors as illustrated below:

```
mknod -m 660 /dev/tsc_1x22_ctl_device   c 251 0
mknod -m 660 /dev/tsc_1x22_ctl_channel  c 251 1
```

In this example, one device descriptor (`tsc_1x22_ctl_device`) is created with major number 251, minor number 0, and one channel descriptor (`tsc_1x22_ctl_channel`) is created with major number 251, minor number 1. The minor number base (0) can be changed (see the compile time configurable parameter in Section 8.2).

Once the driver is installed and the device/channel descriptors are created, the driver service can be accessed via standard C library `open()`, and subsequently with `select()`, `close()`, and `ioctl()` functions.

The following illustrates how the device and channel are opened, closed, and the IOCTL access:

```
devfd  = open("/dev/tsc_1x22_ctl_device",  O_RDONLY|O_WRONLY);
chanfd = open("/dev/tsc_1x22_ctl_channel", O_RDONLY|O_WRONLY);
ioctl (devfd, M1X22_EVENT_GET, &event_structure);
ioctl (chanfd, M1X22_ATH1, NULL);
close (devfd);
close (chanfd);
```

Accessing the driver using IOCTL must be done via an opened descriptor. There are two types of IOCTL commands – the device level commands, which can be accessed by an opened device descriptor, and channel level commands, which can be accessed using an opened channel descriptor. Section 6 describes the IOCTL commands.

3.2 Other Operating Systems

To be provided.

4 Country Specific Settings

The 73M1x22 Control Module supports global compliance parameters for each DAA device it manages. When selected for a specific country code using `M1X22_CH_INIT`, the following predefined parameters will be applied:

1. AC termination impedance – AC impedance register value.
2. DC termination mask – DC mask value.
3. Ring Detection – Ring detection threshold value.
4. Automatic CID Enable – Automatically enter CID state when on hook.
5. Use Seize State – If set, the driver enter seize state for 350 ms before setting ENNOM (refer to the 73M1822/*73M1922 Data Sheet* for a detailed explanation of the seize state).

These parameters are defined in the Country Code Parameter files (`tsc_1x22_ctl_cntry_tbl.c`) and can be changed as required. The list of the country codes supported can be found in Appendix A.

5 Modem Events

The driver provides event service to the high level application by maintaining a FIFO queue of event structures, `M1X22_MDM_EVENT_t`. Events are created by the driver to reflect various conditions as described in Section 5.2. Once created, this new event is added to the FIFO queue and the driver notifies the application layer via file descriptor status change mechanism. This in turn triggers the application to request for the event via the `M1X22_EVENT_GET` IOCTL. Upon retrieval, each event structure is removed from the FIFO queue after its information is conveyed to the high level application.

To receive this modem event notification the application must register for file descriptor status change using the standard UNIX `select()` function. When this function returns the modem event availability status will reflect in the file descriptor parameter. The `FD_ISSET` macro can be used for checking the status, and if available, the application can request for the event using `M1X22_EVENT_GET`.

5.1 M1X22_MDM_EVENT_t

Description

This structure is used by `M1X22_EVENT_GET` to retrieve an event from the event queue. The event structure consists of event ID indentifying the event, the channel ID identifying the modem channel where the event was generated, the number of remaining events in the queue, and up to four event data that carries additional information pertaining to that specific event.

Prototype

```
typedef struct {
    unsigned int         event_id;    /* Event ID          */
    unsigned int         channel_id;  /* Channel ID        */
    unsigned int         event_cnt;   /* number of remaining queued events */
    unsigned int         event_data1; /* additional data 1 */
    unsigned int         event_data2; /* additional data 2 */
    unsigned int         event_data3; /* additional data 3 */
    unsigned int         event_data4; /* additional data 4 */
}
M1X22_MDM_EVENT_t;
```

Parameters

Data Type	Name	Description
`unsigned int`	`event_id`	Event ID (see Section 5.2).
`unsigned int`	`channel_id`	Channel ID.
`unsigned int`	`event_cnt`	Number of events that remain in the queue.
`unsigned int`	`event_data1`	Event data 1.
`unsigned int`	`event_data2`	Event data 2.
`unsigned int`	`event_data3`	Event data 3.
`unsigned int`	`event_data4`	Event data 4.

5.2 Event Identification

5.2.1 M1X22_BATTERY_DROPPED

This event occurs when the modem line is disconnected from the telephone network.

Parameters

Data Type	Name	Description
unsigned int	event_id	M1X22_BATTERY_DROPPED.
unsigned int	channel_id	Channel ID.
unsigned int	event_cnt	Number of events that remain in the queue.
unsigned int	event_data1	N/A.
unsigned int	event_data2	N/A.
unsigned int	event_data3	N/A.
unsigned int	event_data4	N/A.

5.2.2 M1X22_BATTERY_FEEDED

This event is generated when the line is connected to the telephone network and the voltage is restored to normal operating level.

Parameters

Data Type	Name	Description
unsigned int	event_id	M1X22_BATTERY_FEEDED.
unsigned int	channel_id	Channel ID.
unsigned int	event_cnt	Number of events that remain in the queue.
unsigned int	event_data1	N/A.
unsigned int	event_data2	N/A.
unsigned int	event_data3	N/A.
unsigned int	event_data4	N/A.

5.2.3 M1X22_APOH_DETECT

This event is generated when a parallel phone goes off hook.

Parameters

Data Type	Name	Description
unsigned int	event_id	M1X22_APOH_DETECT.
unsigned int	channel_id	Channel ID.
unsigned int	event_cnt	Number of events that remain in the queue.
unsigned int	event_data1	N/A.
unsigned int	event_data2	N/A.
unsigned int	event_data3	N/A.
unsigned int	event_data4	N/A.

5.2.4 M1X22_NOPOH_DETECT

This event is generated when a parallel phone goes on hook.

Parameters

Data Type	Name	Description
unsigned int	event_id	M1X22_NOPOH_DETECT.
unsigned int	channel_id	Channel ID.
unsigned int	event_cnt	Number of events that remain in the queue.
unsigned int	event_data1	N/A.
unsigned int	event_data2	N/A.
unsigned int	event_data3	N/A.
unsigned int	event_data4	N/A.

5.2.5 M1X22_POLARITY_CHG

This event is generated when there is a voltage reversal occurs on the line.

Parameters

Data Type	Name	Description
unsigned int	event_id	M1X22_POLARITY_CHG.
unsigned int	channel_id	Channel ID.
unsigned int	event_cnt	Number of events that remain in the queue.
unsigned int	event_data1	N/A.
unsigned int	event_data2	N/A.
unsigned int	event_data3	N/A.
unsigned int	event_data4	N/A.

5.2.6 M1X22_RING_DETECT

This event is generated at the beginning of the ring burst.

Parameters

Data Type	Name	Description
unsigned int	event_id	M1X22_RING_DETECT.
unsigned int	channel_id	Channel ID.
unsigned int	event_cnt	Number of events that remain in the queue.
unsigned int	event_data1	Ring burst frequency (in Hz)
unsigned int	event_data2	N/A.
unsigned int	event_data3	N/A.
unsigned int	event_data4	N/A.

5.2.7 M1X22_RING_DETECT_END

This event is generated when the driver detects the end of the ring burst.

Parameters

Data Type	Name	Description
unsigned int	event_id	M1X22_RING_DETECT_END.
unsigned int	channel_id	Channel ID.
unsigned int	event_cnt	Number of events that remain in the queue.
unsigned int	event_data1	Ring burst frequency (in Hz)
unsigned int	event_data2	Ring burst duration (in ms)
unsigned int	event_data3	N/A.
unsigned int	event_data4	N/A.

5.2.8 M1X22_SYNC_LOSS_DETECT

This event is generated when the driver detects the failure of synchronization across the barrier path.

Parameters

Data Type	Name	Description
unsigned int	event_id	M1X22_SYNC_LOSS_DETECT.
unsigned int	channel_id	Channel ID.
unsigned int	event_cnt	Number of events that remain in the queue.
unsigned int	event_data1	N/A.
unsigned int	event_data2	N/A.
unsigned int	event_data3	N/A.
unsigned int	event_data4	N/A.

5.2.9 M1X22_OV_DETECT

This event is generated on detection of an over voltage line condition.

Parameters

Data Type	Name	Description
unsigned int	event_id	M1X22_OV_DETECT.
unsigned int	channel_id	Channel ID.
unsigned int	event_cnt	Number of events that remain in the queue.
unsigned int	event_data1	N/A.
unsigned int	event_data2	N/A.
unsigned int	event_data3	N/A.
unsigned int	event_data4	N/A.

5.2.10 M1X22_OI_DETECT

This event is generated on detection of an over current line condition.

Parameters

Data Type	Name	Description
unsigned int	event_id	M1X22_OI_DETECT.
unsigned int	channel_id	Channel ID.
unsigned int	event_cnt	Number of events that remain in the queue.
unsigned int	event_data1	N/A.
unsigned int	event_data2	N/A.
unsigned int	event_data3	N/A.
unsigned int	event_data4	N/A.

5.2.11 M1X22_LINE_STATE

This event is generated when the line state analysis process detects a change in the state of the line. The line state analysis process will monitor (under user control) the line current and/or voltage in both On and off hook states. This event indicates that either the user has requested the current line state or one of the programmable threshold states has become active or non active.

Parameters

Data Type	Name	Description
unsigned int	event_id	M1X22_LINE_STATE.
unsigned int	channel_id	Channel ID.
unsigned int	event_cnt	Number of events that remain in the queue.
unsigned int	event_data1	M1X22_ON_HOOK, M1X22_OFF_HOOK.
unsigned int	event_data2	M1X22_VOLTAGE, M1X22_CURRENT.
unsigned int	event_data3	IET row number.
unsigned int	event_data4	Event identifier.

5.2.12 M1X22_DIAL_COMPLETE

This event is generated when a pulse dial session is completed successfully.

Parameters

Data Type	Name	Description
unsigned int	event_id	M1X22_DIAL_COMPLETE.
unsigned int	channel_id	Channel ID.
unsigned int	event_cnt	Number of events that remain in the queue.
unsigned int	event_data1	N/A.
unsigned int	event_data2	N/A.
unsigned int	event_data3	N/A.
unsigned int	event_data4	N/A.

5.2.13 M1X22_DIAL_ABORTED

This event is generated when a pulse dial session is aborted or canceled by the user application.

Parameters

Data Type	Name	Description
unsigned int	event_id	M1X22_DIAL_ABORTED.
unsigned int	channel_id	Channel ID.
unsigned int	event_cnt	Number of events that remain in the queue.
unsigned int	event_data1	N/A.
unsigned int	event_data2	N/A.
unsigned int	event_data3	N/A.
unsigned int	event_data4	N/A.

5.2.14 M1X22_SYNC_RECOVERED

This event is generated after a successful recovery from the barrier sync failure.

Parameters

Data Type	Name	Description
unsigned int	event_id	M1X22_SYNC_RECOVERED.
unsigned int	channel_id	Channel ID.
unsigned int	event_cnt	Number of events that remain in the queue.
unsigned int	event_data1	N/A.
unsigned int	event_data2	N/A.
unsigned int	event_data3	N/A.
unsigned int	event_data4	N/A.

5.2.15 M1X22_GPIO_INTERRUPT

This event occurs when an input GPIO interrupt is triggered. The “data1” field contains the source GPIO that generates the interrupt.

Parameters

Data Type	Name	Description
unsigned int	event_id	M1X22_GPIO_INTERRUPT.
unsigned int	channel_cid	Channel CID.
unsigned int	event_cnt	Number of events that remain in the queue.
unsigned int	event_data1	The GPIO that triggers the interrupt event.
unsigned int	event_data2	N/A.
unsigned int	event_data3	N/A.
unsigned int	event_data4	N/A.

6 IOCTL Commands Description

Upon successful opening of a device or channel descriptor the application layer can control the operation of the device and the modem channel. The application in user space communicates with the driver via standard Linux driver interface IOCTL calls.

Unless specified, in general, the IOCTL function returns zero (0) for successful request, or a negative value of EFAULT (-EFAULT) if any error is detected during the processing of the IOCTL. The more detail cause of error, or also known as error code, is stored internally and can be retrieved with the `M1X22_ERROR_CODE_GET` IOCTL. The driver keeps only the last error code of the last IOCTL per device or channel descriptor. Therefore, if required, the error code must be retrieved immediately after the execution of the IOCTL or before the next IOCTL is executed. The error codes are listed in Section 7.4.

The following sections describe the detail of each IOCTL command. The driver IOCTL belongs to one of the following categories:

- Initialization
- Event and Status Services
- modem Hook Switch Control Services
- Caller-ID Services
- Ring Detection Services
- Line State Analysis Services
- GPIO Services
- Miscellaneous Service

6.1 Initialization and Configuration IOCTLs

The following IOCTLs are defined to configure the 73M1x22. Table 1 provides a summary of the IOCTLs.

Table 1: Summary of Initialization IOCTLs

IOCTL Name	Description	Descriptor
`M1X22_CH_INIT`	Initializing modem channel for operation.	Channel
`M1X22_CNTRY_NMBR_GET`	Convert ASCII two-character country code into country code index used by the driver.	Channel
`M1X22_GET_COUNTRY_CONFIG`	Read the current default setting for a given country code.	Channel
`M1X22_SET_COUNTRY_CONFIG`	Overwrites the current default setting for a particular country.	Channel
`M1X22_PHONE_VOLUME_SET`	Set transmit and receive gain.	Channel
`M1X22_SET_SAMPLING_FREQ`	Select modem sample rate	Channel
`M1X22_GET_SAMPLING_FREQ`	Get current modem sample rate	Channel

6.1.1 M1X22_CH_INIT

Description

Performs 73M1x22 channel initialization. This includes initialize all default registers and country specific threshold parameters. This IOCTL requires the country code listed in `M1X22_COUNTRY_CODE` as input. The modem channel will be initialized according to the default setting parameter of the input country code. The default setting parameter can be found in the `tsc_1x22_ctl_cntry_tbl.c` file. See also `M1X22_GET_COUNTRY_CONFIG` and for run-time modification of country default setting.

```
#define M1X22_CH_INIT         _IOWR(0xA4,0xC8,unsigned int)
```

Prototype

```
int ioctl (
      int chan_fd,
      int M1X22_CH_INIT,
      unsigned long param );
```

Parameters

Data Type	Name	Description
`int`	`chan_fd`	Channel descriptor.
`int`	`M1X22_CH_INIT`	I/O control identifier for this operation.
`unsigned long`	`param`	Country code as listed in `M1X22_COUNTRY_CODE`.

Return Values

Data Type	Description
`int`	-1 – Failed to initialize device. 0 – Successful.

6.1.2 M1X22_CNTRY_NMBR_GET

Description

Converts a null terminated ASCII string into a country code. This country code can be used in the `M1X22_CH_INIT` IOCTL.

```
#define M1X22_CNTRY_NMBR_GET  _IOWR(0xA4,0xB5,unsigned int)
```

Prototype

```
int ioctl (
      int chan_fd,
      int M1X22_CNTRY_NMBR_GET,
      unsigned long param );
```

Parameters

Data Type	Name	Description
`int`	`chan_fd`	Channel descriptor.
`int`	`M1X22_CNTRY_NMBR_GET`	I/O control identifier for this operation.
`unsigned long`	`param`	Address of the null terminated character string.

Return Values

Data Type	Description
`int`	Return country code – M1X22_COUNTRY_CODE. 255 – Invalid country code.

6.1.3 M1X22_GET_COUNTRY_CONFIG

This IOCTL allows an application program to read the current default setting for a particular country using the country code as an input.

Description

Reads the current default setting for a given country. The country code is passed in via the `cnum` field of the `M1X22_CNTRY_STRUCT_t` structure. This structure is also used by the driver to return the parameter.

```
#define M1X22_GET_COUNTRY_CONFIG     _IOWR(0xA4, 0xF4, unsigned int)
```

Prototype

```
int ioctl (
      int chan_fd,
      int M1X22_GET_COUNTRY_CONFIG,
      unsigned long param );
```

Parameters

Data Type	Name	Description
`int`	`chan_fd`	Channel descriptor.
`int`	`M1X22_GET_COUNTRY_CONFIG`	I/O control identifier for this operation.
`unsigned long`	`param`	Pointer to structure `M1X22_CNTRY_STRUCT_t`.

Return Values

Data Type	Description
`int`	0 – Successful. -EFAULT – Failed to get country config parameter.

Example

The following example illustrates the reading of the default setting for UK.

```
M1X22_CNTRY_STRUCT_t country_config;

country_config.cnum = M1X22_CNTRY_CODE_UK;
ioctl (fd, M1X22_GET_COUNTRY_CONFIG, (unsigned long) &country_config);
printf ("\nCountry Code       : %s", country_config.ccode);
printf ("\nac_impedance       : %d", country_config.ac_impedance);
printf ("\ndc_vi_mask         : %d", country_config.ac_vi_mask);
printf ("\nrgth_value         : %d", country_config.rgth_value);
printf ("\nauto_cid_enable    : %d", country_config.auto_cid_enable);
printf ("\nuse_seize_state    : %d", country_config.use_seize_state);
printf ("\n\n");
```

6.1.4 M1X22_SET_COUNTRY_CONFIG

This IOCTL allows application program to write the current default setting for a particular country using the country code as an input. Once written this becomes the new default setting for the country code until the system is rebooted. For persistence change of default country parameter the `tsc_1X22_ctl_cntry_tbl.c` must be change and rebuilt.

Description

Write to the current default setting of a given country code. The new country config parameter is passed in via the `M1X22_CNTRY_STRUCT_t` structure.

```
#define M1X22_SET_COUNTRY_CONFIG     _IOWR(0xA4, 0xF5, unsigned int)
```

Prototype

```
int ioctl (
      int chan_fd,
      int M1X22_SET_COUNTRY_CONFIG,
      unsigned long param );
```

Parameters

Data Type	Name	Description
`int`	`chan_fd`	Channel descriptor.
`int`	`M1X22_SET_COUNTRY_CONFIG`	I/O control identifier for this operation.
`unsigned long`	`param`	Pointer to structure `M1X22_CNTRY_STRUCT_t`.

Return Values

Data Type	Description
`int`	Always returns 0.

Example

The following example illustrates the writing of the new country config setting for UK.

```
M1X22_CNTRY_STRUCT_t country_config;

country_config.cnum                = M1X22_CNTRY_CODE_UK;
strcpy ((void *) &country_config.ccode, “UK”);
strcpy ((void *) &country_config.country, “United Kingdom”);
country_config.ac_impedance        = 3;
country_config.dc_vi_mask          = 2;
country_config.rgth_value          = 1;
country_config.auto_cid_enable     = FALSE;
country_config.use_seize_state     = FALSE;

ioctl (fd, M1X22_SET_COUNTRY_CONFIG, (unsigned long) &country_config);
```

6.1.5 M1X22_PHONE_VOLUME_SET

Description

The gain of both transmit and receive path can be controlled by digital and/or analog means. This IOCTL provides an easy way to set the transmit and receive gain.

```
#define M1X22_PHONE_VOLUME_SET      _IOWR(0xA4,0xCA,unsigned int)
```

Prototype

```
int ioctl (
      int chan_fd,
      int M1X22_PHONE_VOLUME_SET,
      unsigned long param );
```

Parameters

Data Type	Name	Description
`int`	`chan_fd`	Channel descriptor.
`int`	`M1X22_PHONE_VOLUME_SET`	I/O control identifier for this operation.
`unsigned long`	`param`	Pointer to structure `txrx_gain` (see Section 7.4).

Return Values

Data Type	Description
`int`	0 – Successful. (-1) – Failed.

6.1.6 M1X22_SET_SAMPLING_FREQ

Description

Select modem sample rate. The 73M1x22 device can operate at many different sample rates ranging from 7.2 kHz to 16 kHz. The device defaults to 7.2 kHz operation with a 24.576 MHz crystal upon startup and can be changed using this ioctl.

Notes: Changing of the sample rate will affect the following:

1. Barrier interface operation – a momentarily lost of synchronization on the barrier interface is expected. However, the driver will automatically attempt to recover barrier synchronization. The SYNC lost event is sent and should be followed by SYNC restored event when the barrier is synced up again.

```
#define M1X22_SET_SAMPLING_FREQ     _IOWR(0xA4, 0xA8, unsigned int)
```

Prototype

```
int ioctl (
      int chan_fd,
      int M1X22_SET_SAMPLING_FREQ,
      unsigned long param );
```

Parameters

Data Type	Name	Description
int	chan_fd	Channel descriptor.
int	M1X22_SET_SAMPLING_FREQ	I/O control identifier for this operation.
unsigned long	param	Sample rate selection as defined in M1X22_SAMPLE_RATE_SELECTION.

Return Values

Data Type	Description
int	0 – Successful. -EFAULT– Failed to set sample rate.

6.1.7 M1X22_GET_SAMPLING_FREQ

Description

Return current modem sample rate. The 73M1x22 device can operate at many different sample rates ranging from 7.2 kHz to 16 kHz. The device defaults to 7.2 kHz operation with a 24.576 MHz crystal upon startup.

```
#define M1X22_GET_SAMPLING_FREQ     _IOWR(0xA4, 0xA8, unsigned int)
```

Prototype

```
int ioctl (
      int chan_fd,
      int M1X22_GET_SAMPLING_FREQ)
      unsigned long param );
```

Parameters

Data Type	Name	Description
int	chan_fd	Channel descriptor.
int	M1X22_GET_SAMPLING_FREQ	I/O control identifier for this operation.
unsigned long	param	N/A.

Return Values

Data Type	Description
int	Sample rate selection as defined in M1X22_SAMPLE_RATE_SELECTION.

6.2 Events and Status Service

The driver can be queried for a variety of current status of the modem line via IOCTLs. Table 2 summarizes these services.

Table 2: Modem Line Status Services

Events and Status	Description	Descriptor
M1X22_RNG_GET	Modem line ring status commands.	Channel
M1X22_POL_GET	Modem line polarity status commands.	Channel
M1X22_BAT_GET	Modem line battery status command.	Channel
M1X22_POH_GET	Modem line POH status command.	Channel
M1X22_EVENT_GET	Event retrieval command.	Device
M1X22_ERROR_CODE_GET	Retrieve last error code command.	Device/Channel

6.2.1 M1X22_RNG_GET

Description

Gets the current ring status of the modem line. The current ring status is maintained in the driver internal variable. This variable is being retrieved by use application using this command.

```
#define M1X22_RNG_GET   _IOWR(0xA4,0xB8,unsigned int)
```

Prototype

```
int ioctl (
      int chan_fd,
      int M1X22_RNG_GET,
      unsigned long param );
```

Parameters

Data Type	Name	Description
int	chan_fd	Channel descriptor.
int	M1X22_RNG_GET	I/O control identifier for this operation.
unsigned long	param	N/A.

Return Values

Data Type	Description
int	The return value can be either of the following: 0 – No ring signal occurred on the modem line. 1 – The modem line is ringing, signaling of incoming call.

6.2.2 M1X22_POL_GET

Description

Gets the current polarity reversal status of the modem line. The driver maintains this status in its local variable and it is being retrieved by user application using this command.

```
#define M1X22_POL_GET   _IOWR(0xA4,0xB7,unsigned int)
```

Prototype

```
int ioctl (
      int chan_fd,
      int M1X22_POL_GET,
      unsigned long param );
```

Parameters

Data Type	Name	Description
int	chan_fd	Channel descriptor.
int	M1X22_POL_GET	I/O control identifier for this operation.
unsigned long	param	N/A.

Return Values

Data Type	Description
int	The return value can be either of the following: 0 – No polarity reversal event. 1 – polarity reversal event occurred on the modem line.

6.2.3 M1X22_BAT_GET

Description

Gets the current battery status of the modem line. The driver maintains this status in its local variable and it is being retrieved by user application using this command.

```
#define M1X22_BAT_GET   _IOWR(0xA4,0xB6,unsigned int)
```

Prototype

```
int ioctl (
      int chan_fd,
      int M1X22_BAT_GET,
      unsigned long param );
```

Parameters

Data Type	Name	Description
int	chan_fd	Channel descriptor.
int	M1X22_BAT_GET	I/O control identifier for this operation.
unsigned long	param	N/A.

Return Values

Data Type	Description
int	The return value can be either of the following: 0 – The modem line is not powered. 1 – The modem line is connected to the PSTN and being powered.

6.2.4 M1X22_POH_GET

Description

Gets the current “parallel phone off-hook” status of the modem line. The driver maintains this status in its local variable and it is being retrieved by user application using this command.

```
#define M1X22_POH_GET   _IOWR(0xA4,0xB9,unsigned int)
```

Prototype

```
int ioctl (
      int chan_fd,
      int M1X22_POH_GET,
      unsigned long param );
```

Parameters

Data Type	Name	Description
int	chan_fd	Channel descriptor.
int	M1X22_POH_GET	I/O control identifier for this operation.
unsigned long	param	N/A.

Return Values

Data Type	Description
int	The return value can be either of the following: 0 – No parallel phone off-hook on the modem line. 1 – A parallel phone off-hook on the modem line.

6.2.5 M1X22_EVENT_GET

Description

Returns an event from the FIFO queue. The driver records various events in its internal FIFO queue. Access to this event on this queue by user application is accomplished using this command, and the event will be removed permanently from the queue.

```
#define M1X22_EVENT_GET _IOWR(0xA4,0xB1,unsigned int)
```

Prototype

```
int ioctl (
      int dev_fd,
      int M1X22_EVENT_GET,
      unsigned long param );
```

Parameters

Data Type	Name	Description
`int`	`dev_fd`	Device descriptor.
`int`	`M1X22_EVENT_GET`	I/O control identifier for this operation.
`unsigned long`	`param`	Pointer to structure `M1X22_MDM_EVENT_t` (see Section 5.1).

Return Values

Data Type	Description
`int`	0 – Successful. -EFAULT Failed to retrieve event data.

6.2.6 M1X22_ERROR_CODE_GET

Description

This IOCTL returns the error code of the last IOCTL command. The driver records only the last error code and applicable to device and channel descriptor.

```
#define M1X22_ERROR_CODE_GET _IOWR(0xA4,0xB2,unsigned int)
```

Prototype

```
int ioctl (
      int dev_fd,
      int M1X22_ERROR_CODE_GET,
      unsigned long param );
```

Parameters

Data Type	Name	Description
int	dev_fd/chan_fd	Device or Channel descriptor.
int	M1X22_ERROR_CODE_GET	I/O control identifier for this operation.
unsigned long	param	Pointer to the error code of type unsigned int.

Return Values

Data Type	Description
int	0 – Successful. -EFAULT – Failed to retrieve the error code.

6.3 Modem Hook Switch Control Services

The configuration and control of the modem Hook Switch is accomplished by using the IOCTLs summarized in Table 3.

Table 3: Modem Hook Switch Control Services

Events and Status	Description	Descriptor
M1X22_ENNOM_DELAY_TIMER	Allows the developer to tune the EnNom response timer.	Channel
M1X22_ATH1	Issue off-hook in the modem interface.	Channel
M1X22_ATH0	Issue on-hook in the modem interface.	Channel
M1X22_ATDP	Pulse dial.	Channel
M1X22_ATDP_CANCEL	Pulse dial abort or cancel.	Channel
M1X22_ATDP_PARAM	Pulse dial parameters.	Channel
M1X22_FLSH_CFG	Configure of flash-hook parameter for the modem interface.	Channel
M1X22_FLSH_SET	Perform hook flashing on the modem interface.	Channel
M1X22_SEND_WETTING_PULSE	Perform a wetting pulse on the modem interface.	Channel

6.3.1 M1X22_ENNOM_DELAY_TIMER

This IOCTL allows an application program to change the ENNOM delay timer from the default value of 350 ms to any value within the valid ranges of 10 to 350 ms, inclusive

Description

Delaying of ENNOM bit setting is required for loop stabilization during off hook operation. However, the duration depends largely on the quality of the hook circuit design. The driver uses the default setting of 350 ms an optimized choice for response time and audio quality. This IOCTL allows developers to tune this timing value to suit their specific needs.

```
#define M1X22_ENNOM_DELAY_TIMER     _IOWR(0xA4, 0xF8, unsigned int)
```

Prototype

```
int ioctl (
      int chan_fd,
      int M1X22_ENNOM_DELAY_TIMER,
      unsigned long param );
```

Parameters

Data Type	Name	Description
int	chan_fd	Channel descriptor.
int	M1X22_ENNOM_DELAY_TIMER	I/O control identifier for this operation.
unsigned long	param	The desired ENNOM delay duration. Range from 10 to 350 ms, inclusive.

Return Values

Data Type	Description
int	Always returns 0.

6.3.2 M1X22_ATH1

Description

Issues the off-hook signal to the modem interface.

```
#define M1X22_ATH1      _IOWR(0xA4,0xA2,unsigned int)
```

Note: The driver provides an option to generate a battery status event upon completion of this off-hook procedure. This option is controlled by the following macro defined in a header file. By default this macro is disabled:

```
#define SEND_BAT_STATUS_OFFHOOK
```

It is envisioned that the application layer will use this event to determine if the call establishment should be proceeded.

Prototype

```
int ioctl (
      int chan_fd,
      int M1X22_ATH1,
      unsigned long param );
```

Parameters

Data Type	Name	Description
`int`	`chan_fd`	Channel descriptor.
`int`	`M1X22_ATH1`	I/O control identifier for this operation.
`unsigned long`	`param`	N/A.

Return Values

Data Type	Description
`int`	Always returns 0.

6.3.3 M1X22_ATH0

Description

Issues on-hook in the modem interface.

```
#define M1X22_ATH0      _IOWR(0xA4,0xA1,unsigned int)
```

Prototype

```
int ioctl (
      int chan_fd,
      int M1X22_ATH0,
      unsigned long param );
```

Parameters

Data Type	Name	Description
int	chan_fd	Channel descriptor.
int	M1X22_ATH0	I/O control identifier for this operation.
unsigned long	param	N/A.

Return Values

Data Type	Description
int	Always returns 0.

6.3.4 M1X22_ATDP

Description

Performs pulse dialing on the modem channel. As a pulse dial procedure can take more than a second per digit, it is absolutely essential that this session be carried out transparently in the background without locking up the caller during the process. Therefore, this IOCTL is a non-blocking call and it returns immediately after scheduling the background process to start the pulse dialing. For that reason, the return code does not reflect the status of the pulse dial, but rather the status of the scheduling of the pulse dial session.

An active dial session can be aborted using the `M1X22_ATDP_CANCEL` IOCTL. If done before its completion, the driver stops the dialing and sends the `M1X22_DIAL_ABORTED` event. However, upon a successful completion of the dialing, the driver sends an `M1X22_DIAL_COMPLETE` event to notify the application layer of the status. It is recommended that the application monitor the pulse dial status event (`M1X22_DIAL_COMPLETE` or `M1X22_DIAL_ABORTED`) to synchronize with the driver as to when the dial session is completed.

Note: The driver rejects all IOCTLs while this pulse dial session is in progress, except `M1X22_ATDP_CANCEL`.

```
#define M1X22_ATDP      _IOWR(0xA4,0xA3,unsigned int)
```

Prototype

```
int ioctl (
      int chan_fd,
      int M1X22_ATDP,
      unsigned long param );
```

Parameters

Data Type	Name	Description
`int`	`chan_fd`	Channel descriptor.
`int`	`M1X22_ATDP`	I/O control identifier for this operation.
`unsigned long`	`param`	Pointer to the `M1X22_PULSE_DIAL_t` structure.

Return Values

Data Type	Description
`int`	Always returns 0.

6.3.5 M1X22_ATDP_CANCEL

Description

Aborts or cancels an active pulse session requested previously by using the `M1X22_ATDP` IOCTL. The cancellation occurs in the background and, when it is done, the driver sends an `M1X22_DIAL_ABORTED` event.

```
#define M1X22_ATDP_CANCEL     _IOWR(0xA4,0xDD,unsigned int)
```

Prototype

```
int ioctl (
      int chan_fd,
      int M1X22_ATDP_CANCEL,
      unsigned long param );
```

Parameters

Data Type	Name	Description
`int`	`chan_fd`	Channel descriptor.
`int`	`M1X22_ATDP_CANCEL`	I/O control identifier for this operation.
`unsigned long`	`param`	N/A.

Return Values

Data Type	Description
`int`	Always returns 0.

6.3.6 M1X22_ATDP_PARAM

Description

This IOCTL is used to modify or read the following default pulse dial parameters:

- On hook duration (default = 60 ms)
- Off hook duration (default = 40 ms)
- Inter-digit duration (default = 1 sec)

The command field in the `M1X22_PULSE_DIAL_PARAM_t` structure indicates whether it is a read or a write operation. For reading the driver returns the parameters in the structure, while for writing the driver expects the new pulse dial parameters to be written in the structure.

```
#define M1X22_ATDP_PARAM      _IOWR(0xA4,0xAD,unsigned int)
```

Prototype

```
int ioctl (
      int chan_fd,
      int M1X22_ATDP_PARAM,
      unsigned long param );
```

Parameters

Data Type	Name	Description
`int`	`chan_fd`	Channel descriptor.
`int`	`M1X22_ATDP_PARAM`	I/O control identifier for this operation.
`unsigned long`	`param`	Pointer to the `M1X22_PULSE_DIAL_PARAM_t` structure.

Return Values

Data Type	Description
`int`	Always returns 0.

6.3.7 M1X22_FLSH_CFG

Description

This is the configuration of flash-hook timing parameter for the modem interface.

```
#define M1X22_FLSH_CFG  _IOWR(0xA4,0xBA,unsigned int)
```

Prototype

```
int ioctl (
      int chan_fd,
      int M1X22_FLSH_CFG,
      unsigned long param );
```

Parameters

Data Type	Name	Description
int	chan_fd	Channel descriptor.
int	M1X22_FLSH_CFG	I/O control identifier for this operation.
unsigned long	param	The desired flash duration. Range from 5 to 50 ms, inclusive. If out of range it is forced to 10 ms.

Return Values

Data Type	Description
int	Always returns 0.

6.3.8 M1X22_FLSH_SET

Description

Performs hook flashing on the modem interface for the duration set by M1X22_FLSH_CFG.

```
#define M1X22_FLSH_SET  _IOWR(0xA4,0xBC,unsigned int)
```

Prototype

```
int ioctl (
      int chan_fd,
      int M1X22_FLSH_SET,
      unsigned long param );
```

Parameters

Data Type	Name	Description
int	chan_fd	Channel descriptor.
int	M1X22_FLSH_SET	I/O control identifier for this operation.
unsigned long	param	N/A.

Return Values

Data Type	Description
int	Always returns 0.

6.3.9 M1X22_SEND_WETTING_PULSE

Description

Performs hook flashing on the modem interface for the duration set by `param`.

```
#define M1X22_FLSH_SET  _IOWR(0xA4,0xBC,unsigned int)
```

Prototype

```
int ioctl (
      int chan_fd,
      int M1X22_SEND_WETTING_PULSE,
      unsigned long param );
```

Parameters

Data Type	Name	Description
`int`	`chan_fd`	Channel descriptor.
`int`	`M1X22_SEND_WETTING_PULSE`	I/O control identifier for this operation.
`unsigned long`	`param`	Integer representing the desired wetting pulse duration. Range from 1 to 1000 ms. Out of range behavior is undefined.

Return Values

Data Type	Description
`int`	0 – Successful. -1 – Modem channel is not off-hook.

6.4 Caller-ID Services

The following service control how the modem manages Type 1 Caller-ID.

Table 4: Call ID Services

Name	Description	Descriptor
M1X22_ENABLE_CALLER_ID	Enable automatic Caller ID enabling mode.	Channel
M1X22_DISABLE_CALLER_ID	Disable automatic Caller ID enabling mode.	Channel
M1X22_ENTER_CID_MODE	Manually enter Caller ID mode.	Channel
M1X22_EXIT_CID_MODE	Manually exit Caller ID mode.	Channel

6.4.1 M1X22_ENABLE_CALLER_ID

Description

Enables the automatic Caller ID processing. By default the driver will start Caller ID mode with an on hook transition. This can be enabled by upper layer application using this service.

```
#define M1X22_DISABLE_CALLER_ID     _IOWR(0xA4, 0xF2, unsigned int)
```

Prototype

```
int ioctl (
      int chan_fd,
      int M1X22_ENABLE_CALLER_ID,
      unsigned long param );
```

Parameters

Data Type	Name	Description
int	chan_fd	Channel descriptor.
int	M1X22_ENABLE_CALLER_ID	I/O control identifier for this operation.
unsigned long	param	N/A.

Return Values

Data Type	Description
int	Always returns 0.

6.4.2 M1X22_DISABLE_CALLER_ID

Description

Disables the automatic Caller ID processing. By default the driver will start Caller ID mode with an on hook transition. This can be disabled by upper layer application using this service.

```
#define M1X22_DISABLE_CALLER_ID     _IOWR(0xA4, 0xF2, unsigned int)
```

Prototype

```
int ioctl (
      int chan_fd,
      int M1X22_DISABLE_CALLER_ID,
      unsigned long param );
```

Parameters

Data Type	Name	Description
int	chan_fd	Channel descriptor.
int	M1X22_DISABLE_CALLER_ID	I/O control identifier for this operation.
unsigned long	param	N/A.

Return Values

Data Type	Description
int	Always returns 0.

6.4.3 M1X22_ENTER_CID_MODE

Description

Manually enters Caller ID mode regardless of the state of the automatic CID service.

```
#define M1X22_ENTER_CID_MODE     _IOWR(0xA4, 0xF2, unsigned int)
```

Prototype

```
int ioctl (
      int chan_fd,
      int M1X22_ENTER_CID_MODE,
      unsigned long param );
```

Parameters

Data Type	Name	Description
int	chan_fd	Channel descriptor.
int	M1X22_ENTER_CID_MODE	I/O control identifier for this operation.
unsigned long	param	N/A.

Return Values

Data Type	Description
int	0 – Successful. -1 – If modem channel is not on-hook.

6.4.4 M1X22_EXIT_CID_MODE

Description

Exits Caller ID mode.

```
#define M1X22_EXIT_CID_MODE     _IOWR(0xA4, 0xF2, unsigned int)
```

Prototype

```
int ioctl (
      int chan_fd,
      int M1X22_EXIT_CID_MODE,
      unsigned long param );
```

Parameters

Data Type	Name	Description
int	chan_fd	Channel descriptor.
int	M1X22_EXIT_CID_MODE	I/O control identifier for this operation.
unsigned long	param	N/A.

Return Values

Data Type	Description
int	Always returns 0.

6.5 Ring Detection Services

The following services control how the modem manages Ring Detection.

Table 5: Ring Detection Services

Name	Description	Descriptor
M1X22_SET_MIN_INTER_RING_GAP	Set minimum inter-ring timer value.	Channel
M1X22_SET_RING_MIN_FREQ	Set min frequency threshold for ring filter.	Channel
M1X22_SET_RING_MAX_FREQ	Set max frequency threshold for ring filter.	Channel

6.5.1 M1X22_SET_MIN_INTER_RING_GAP

Description

Upon detection of a RGDT interrupt the driver will start a timer with a minimum duration specified by this IOCTL (default is 150 ms). If no more RGDT interrupts are detected in that time period, then the driver will attempt to interpret the existing RGDT interrupts (polarity reversal or ring burst). Polarity reversals and/or ring bursts separated by less than this time period will be considered to be one event and will be interpreted as such. RGDT interrupts farther apart than this will be interpreted as separate events.

```
#define M1X22_SET_MIN_INTER_RING_GAP     _IOWR(0xA4, 0xE2, unsigned int)
```

Prototype

```
int ioctl (
      int chan_fd,
      int M1X22_SET_MIN_INTER_RING_GAP,
      unsigned long param );
```

Parameters

Data Type	Name	Description
int	chan_fd	Channel descriptor.
int	M1X22_SET_MIN_INTER_RING_GAP	I/O control identifier for this operation.
unsigned long	param	Time in ms.

Return Values

Data Type	Description
int	Always returns 0.

6.5.2 M1X22_SET_RING_MIN_FREQ

Description

Upon detection of a ring burst the driver will attempt to determine the ring burst frequency. If the ring burst frequency is above the minimum ring frequency and below the maximum frequency then the driver will report an `M1X22_RING_DETECT` event.

```
#define M1X22_SET_RING_MIN_FREQ     _IOWR(0xA4, 0xE3, unsigned int)
```

Prototype

```
int ioctl (
      int chan_fd,
      int M1X22_SET_RING_MIN_FREQ,
      unsigned long param );
```

Parameters

Data Type	Name	Description
`int`	`chan_fd`	Channel descriptor.
`int`	`M1X22_SET_ RING_MIN_FREQ`	I/O control identifier for this operation.
`unsigned long`	`param`	Frequency in Hz.

Return Values

Data Type	Description
`int`	Always returns 0.

6.5.3 M1X22_SET_RING_MAX_FREQ

Description

Upon detection of a ring burst the driver will attempt to determine the ring burst frequency. If the ring burst frequency is above the minimum ring frequency and below the maximum frequency, then the driver will report an `M1X22_RING_DETECT` event.

```
#define M1X22_SET_RING_MAX_FREQ     _IOWR(0xA4, 0xE4, unsigned int)
```

Prototype

```
int ioctl (
      int chan_fd,
      int M1X22_SET_RING_MIN_FREQ,
      unsigned long param );
```

Parameters

Data Type	Name	Description
`int`	`chan_fd`	Channel descriptor.
`int`	`M1X22_SET_ RING_MAX_FREQ`	I/O control identifier for this operation.
`unsigned long`	`param`	Frequency in Hz.

Return Values

Data Type	Description
`int`	Always returns 0.

6.6 Line State Analysis Services

The following services control the modem line state via line current and line voltage measurements. Table 6 provides the summary of each IOCTL. These IOCTLs can be used for both measuring entities – the line current and line voltage.

Table 6: Line State Analysis Services

Name	Description	Descriptor
M1X22_MEASURE_START	Start line measurement.	Channel
M1X22_MEASURE_STOP	Stop line measurement.	Channel
M1X22_MEASURE_UPDATE	Update line measurement parameter (IET).	Channel

6.6.1 M1X22_MEASURE_START

Description

Starts the measurement of a measuring entity (current or voltage) as specified by its IET parameters. The `param` parameter points to a structure that contains the requested measurement criteria.

```
#define M1X22_MEASURE_START   _IOWR(0xA4, 0xE5, unsigned int)
```

Prototype

```
int ioctl (
      int chan_fd,
      int M1X22_MEASURE_START,
      unsigned long param );
```

Parameters

Data Type	Name	Description
int	chan_fd	Channel descriptor.
int	M1X22_MEASURE_START	I/O control identifier for this operation.
unsigned long	param	Pointer to M1X22_MEASURE_START_STOP_t.

Return Values

Data Type	Description
unsigned int	0 – Successful. -1 – Failed to start measurement.

Example

The following example code illustrates the starting of the line current monitoring.

```
M1X22_MEASURE_START_t current;
int ret;

current.entity = M1X22_MEASURE_ENTITY_CURRENT;    /* start current monitor */
current.sample_time = 100;                        /* sampling at 100ms interval */
current.average_sample_count = 10;                /* averaging over 10 samples  */

ret = ioctl (fd, M1X22_MEASURE_START, &current);
if (ret < 0)
   printf ("Failed to start line current monitoring");
else
   printf ("Successful");
```

6.6.2 M1X22_MEASURE_STOP

Description

Stops an on-going measurement (current or voltage). The `param` parameter points to a structure that contains the requested stop measuring entity.

```
#define M1X22_MEASURE_STOP   _IOWR(0xA4, 0xE6, unsigned int)
```

Prototype

```
int ioctl (
      int chan_fd,
      int M1X22_MEASURE_STOP,
      unsigned long param );
```

Parameters

Data Type	Name	Description
`int`	`chan_fd`	Channel descriptor.
`int`	`M1X22_MEASURE_STOP`	I/O control identifier for this operation.
`unsigned long`	`param`	Pointer to `M1X22_MEASURE_START_STOP_t`.

Return Values

Data Type	Description
`unsigned long`	0 – Successful. -1 – Failed to stop measurement.

Example

The following example code illustrates the stopping of an on-going voltage measurement process.

```
M1X22_MEASURE_START_t voltage;
int ret;

voltage.entity = M1X22_MEASURE_ENTITY_VOLTAGE; /* stop voltage monitor */

ret = ioctl (fd, M1X22_MEASURE_STOP, &voltage);
if (ret < 0)
   printf (“Failed to stop line voltage monitoring”);
else
   printf (“Successful”);
```

6.6.3 M1X22_MEASURE_UPDATE

Description

Reads an IET entry or updates an IET entry based on provided parameters. The param parameter points to a structure that contains the requested action (GET or SET), the measuring entity (current or voltage), the IET table index, and its attributes, if used in the SET operation. For GET operations the IET attributes will be read from the driver.

```
#define M1X22_MEASURE_UPDATE   _IOWR(0xA4, 0xE7, unsigned int)
```

Prototype

```
int ioctl (
      int chan_fd,
      int M1X22_MEASURE_UPDATE,
      unsigned long param );
```

Parameters

Data Type	Name	Description
int	chan_fd	Channel descriptor.
int	M1X22_MEASURE_UPDATE	I/O control identifier for this operation.
unsigned long	param	Pointer to M1X22_MEASURE_UPDATE_t.

Return Values

Data Type	Description
unsigned int	0 – Successful. -1 – Failed to update IET entry.

Example

The following example code illustrates an update to a line current IET table entry.

```
M1X22_MEASURE_UPDATE_t update;
int ret;

update.row = 3;                        /* update row 3 of current IET table */
update.action = M1X22_MEASURE_ACTION_SET;     /* request for SET operation */
update.entity = M1X22_MEASURE_ENTITY_CURRENT; /* measuring entity= current */
update.interval_min = 15;                    /* lower bound current = 15ma */
update.interval_max = 21;                    /* upper bound current = 21ma */
update.event        = 0x000E1521;            /* this event to be emitted   */

ret = ioctl (fd, M1X22_MEASURE_UPDATE, &update);
if (ret < 0)
   printf (“Failed to update current IET entry”);
else
   printf (“Successful”);
```

The following example code illustrates a reading of a line voltage IET table entry.

```
M1X22_MEASURE_UPDATE_t read;
int ret;

read.row = 5;                              /* read row 5 of voltage IET table */
read.action = M1X22_MEASURE_ACTION_GET;       /* request for GET operation */
read.entity = M1X22_MEASURE_ENTITY_VOLTAGE;   /* measuring entity= voltage */

ret = ioctl (fd, M1X22_MEASURE_UPDATE, &read);
if (ret < 0)
   printf (“Failed to read voltage IET entry”);
else {
   printf (“\nReading voltage IET table entry row: %d”, read.row);
   printf (“\nInterval min: %d”, read.interval_min);
   printf (“\nInterval max: %d”, read.interval_max);
   printf (“\nEvent       : 0x%08X”, read.event);
}
```

6.7 GPIO Services

6.7.1 M1X22_GPIO_CONFIG

Description

The ioctl is used to configure the GPIO pin.

```
#define M1X22_GPIO_CONFIG   _IOWR(0xA4, 0xC0, unsigned int)
```

Prototype

```
int ioctl (
      int chan_fd,
      int M1X22_GPIO_CONFIG,
      unsigned long param );
```

Parameters

Data Type	Name	Description
`int`	`chan_fd`	Channel descriptor.
`int`	`M1X22_GPIO_CONFIG`	I/O control identifier for this operation.
`unsigned long`	`param`	Pointer to `M1X22_GPIO_CONFIG_t`.

Return Values

Data Type	Description
`unsigned int`	0 – Successful. -1 – Failed to configure GPIO.

6.7.2 M1X22_GPIO_CONTROL

Description

The ioctl is used to control the operation of the GPIO pin.

```
#define M1X22_GPIO_CONTROL   _IOWR(0xA4, 0xC1, unsigned int)
```

Prototype

```
int ioctl (
      int chan_fd,
      int M1X22_GPIO_CONTROL,
      unsigned long param );
```

Parameters

Data Type	Name	Description
int	chan_fd	Channel descriptor.
int	M1X22_GPIO_CONTROL	I/O control identifier for this operation.
unsigned long	param	Pointer to M1X22_GPIO_CONTROL_t.

Return Values

Data Type	Description
unsigned int	0 – Successful. -1 – Failed to control GPIO.

6.7.3 M1X22_GPIO_DATA

Description

This IOCTL is used to read or write data from and to the GPIO pin.

```
#define M1X22_GPIO_DATA   _IOWR(0xA4, 0xC2, unsigned int)
```

Prototype

```
int ioctl (
      int chan_fd,
      int M1X22_GPIO_DATA,
      unsigned long param );
```

Parameters

Data Type	Name	Description
int	chan_fd	Channel descriptor.
int	M1X22_GPIO_DATA	I/O control identifier for this operation.
unsigned long	param	Pointer to the M1X22_GPIO_DATA_t structure**.**

Return Values

Data Type	Description
unsigned int	0 – Successful. -1 – Failed to access GPIO data.

6.8 Loopback Services

6.8.1 M1X22_LOOPBACK

Description

This IOCTL is used for managing the loopback session – initiating, clearing or retrieving the status of current active loopback session on a given channel. Only one loopback session can be active per channel. For reading the status the loopback "mode" will be returned in the structure `M1X22_LOOPBACK_t` pointed to by the "param" field.

```
#define M1X22_LOOPBACK   _IOWR(0xA4, 0xBD, unsigned int)
```

Prototype

```
int ioctl (
      int chan_fd,
      int M1X22_LOOPBACK,
      unsigned long param );
```

Parameters

Data Type	Name	Description
`int`	`chan_fd`	Channel descriptor.
`int`	`M1X22_LOOPBACK`	I/O control identifier for this operation.
`unsigned long`	`param`	Pointer to structure `M1X22_LOOPBACK_t`.

Return Values

Data Type	Description
`unsigned int`	0 – Successful. -FAULT – Failed to perform loopback request.

6.9 Miscellaneous

6.9.1 M1X22_THRESHOLD_OVERRIDE

Description

Various modem channel parameters are conveniently grouped and predefined in the country parameter setting as specified in Section 7.2. These parameters are programmed during channel initialization ioctl (`M1X22_CH_INIT`). However, these parameters can be overridden at runtime to fine tune to the desired threshold for the specific installation using this `M1X22_THRESHOLD_OVERRIDE` IOCTL.

Note: This IOCTL must be invoked after M1X22_CH_INIT to prevent the parameter from over written by `M1X22_CH_INIT`.

```
#define M1X22_THRESHOLD_OVERRIDE    _IOWR(0xA4,0xB3,unsigned int)
```

Prototype

```
int ioctl (
      int chan_fd,
      int M1X22_THRESHOLD_OVERRIDE,
      unsigned long param );
```

Parameters

Data Type	Name	Description
`int`	`chan_fd`	Channel descriptor.
`int`	`M1X22_THRESHOLD_OVERRIDE`	I/O control identifier for this operation.
`unsigned long`	`param`	Pointer to the `M1X22_THRESH_OVERRIDE_t` structure.

Return Values

Data Type	Description
`int`	-EFAULT – Failed to perform threshold override. 0 – Successful.

6.9.2 M1X22_BTONE_FILTER

Description

Large amplitude out-of-band tones can be used to measure call duration and to allow remote central office to determine the duration of the call for billing purposes. These tones can saturate or distort the input signal, thus, it is important to be able to reject them. This IOCTL provides the ability to filter out the billing tone.

```
#define M1X22_BTONE_FILTER   _IOWR(0xA4,0xC3,unsigned int)
```

Prototype

```
int ioctl (
      int chan_fd,
      int M1X22_BTONE_FILTER,
      unsigned long param );
```

Parameters

Data Type	Name	Description
int	chan_fd	Channel descriptor.
int	M1X22_BTINE_FILTER	I/O control identifier for this operation.
unsigned long	param	Pointer to the M1X22_BTONE_FILTER_t structure.

Return Values

Data Type	Description
int	-EFAULT – Failed to perform billing tone filter. 0 – Successful.

6.9.3 M1X22_CPROG_MONITOR

Description

The Call Progress Monitor monitors activities on the line. The audio output contains both transmit and receive data with a configurable level individually can be set using this IOCTL.

```
#define M1X22_CPROG_MONITOR    _IOWR(0xA4,0xF0,unsigned int)
```

Prototype

```
int ioctl (
      int chan_fd,
      int M1X22_CPROG_MONITOR,
      unsigned long param );
```

Parameters

Data Type	Name	Description
int	chan_fd	Channel descriptor.
int	M1X22_CPROG_MONITOR	I/O control identifier for this operation.
unsigned long	param	Pointer to the M1X22_CPROG_MONITOR_t structure.

Return Values

Data Type	Description
int	-EFAULT – Failed to perform call progress monitor. 0 – Successful.

6.9.4 M1X22_DEBUG_LEVEL_SET

Description

Sets the driver trace mask to enable or disable run-time trace messages. Multiple trace masks can be ORed together.

```
#define M1X22_DEBUG_LEVEL_SET       _IOWR(0xA4,0xE1,unsigned int)
```

Prototype

```
int ioctl (
      int dev_fd,
      int M1X22_DEBUG_LEVEL_SET,
      unsigned long param );
```

Parameters

Data Type	Name	Description
int	dev_fd	Device descriptor.
int	M1X22_DEBUG_LEVEL_SET	I/O control identifier for this operation.
unsigned long	param	Debug trace mask: M1X22_DEBUG_TRACE_MASK.

Return Values

Data Type	Description
int	-1 – Failed set debug level mask. 0 – Successful.

7 Type and Structure Definition Reference

This section contains the type definitions, reference of data type and structure used in the 73M1x22 driver.

7.1 M1X22_COUNTRY_CODE

Description

List of country codes use in the `M1X22_CH_INIT` IOCTL.

Prototype

```
/*****************************************************************
**  73M1X22 Country code List - Internet Country Codes
*****************************************************************/
#define M1X22_CNTRY_CODE_AR             0       /* "Argentina"    */
#define M1X22_CNTRY_CODE_AU             1       /* "Australia"    */
#define M1X22_CNTRY_CODE_AT             2       /* "Austria"      */
#define M1X22_CNTRY_CODE_BH             3       /* "Bahrain"      */
#define M1X22_CNTRY_CODE_BE             4       /* "Belgium"      */
#define M1X22_CNTRY_CODE_BR             5       /* "Brazil"       */
#define M1X22_CNTRY_CODE_BG             6       /* "Bulgaria"     */
#define M1X22_CNTRY_CODE_CA             7       /* "Canada"       */
#define M1X22_CNTRY_CODE_CL             8       /* "Chile"        */
#define M1X22_CNTRY_CODE_C1             9       /* "ChinaData"    */
#define M1X22_CNTRY_CODE_C2             10      /* "ChinaVoice"   */
#define M1X22_CNTRY_CODE_CO             11      /* "Columbia"     */
#define M1X22_CNTRY_CODE_HR             12      /* "Croatia"      */
#define M1X22_CNTRY_CODE_TB             13      /* "TBR 21"       */
#define M1X22_CNTRY_CODE_CY             14      /* "Cyprus"       */
#define M1X22_CNTRY_CODE_CZ             15      /* "Czech Rep"    */
#define M1X22_CNTRY_CODE_DK             16      /* "Denmark"      */
#define M1X22_CNTRY_CODE_EC             17      /* "Ecuador"      */
#define M1X22_CNTRY_CODE_EG             18      /* "Egypt"        */
#define M1X22_CNTRY_CODE_SV             19      /* "El Salvador"  */
#define M1X22_CNTRY_CODE_FI             20      /* "Finland"      */
#define M1X22_CNTRY_CODE_FR             21      /* "France"       */
#define M1X22_CNTRY_CODE_DE             22      /* "Germany"      */
#define M1X22_CNTRY_CODE_GR             23      /* "Greece"       */
#define M1X22_CNTRY_CODE_GU             24      /* "Guam"         */
#define M1X22_CNTRY_CODE_HK             25      /* "Hong Kong"    */
#define M1X22_CNTRY_CODE_HU             26      /* "Hungary"      */
#define M1X22_CNTRY_CODE_IS             27      /* "Iceland"      */
#define M1X22_CNTRY_CODE_IN             28      /* "India"        */
#define M1X22_CNTRY_CODE_ID             29      /* "Indonesia"    */
#define M1X22_CNTRY_CODE_IE             30      /* "Ireland"      */
#define M1X22_CNTRY_CODE_IL             31      /* "Israel"       */
#define M1X22_CNTRY_CODE_IT             32      /* "Italy"        */
#define M1X22_CNTRY_CODE_JP             33      /* "Japan"        */
#define M1X22_CNTRY_CODE_JO             34      /* "Jordan"       */
#define M1X22_CNTRY_CODE_KZ             35      /* "Kazakhstan"   */
#define M1X22_CNTRY_CODE_KW             36      /* "Kuwait"       */
#define M1X22_CNTRY_CODE_LV             37      /* "Latvia"       */
#define M1X22_CNTRY_CODE_LB             38      /* "Lebanon"      */
#define M1X22_CNTRY_CODE_LU             39      /* "Luxembourg"   */
#define M1X22_CNTRY_CODE_MO             40      /* "Macao"        */
#define M1X22_CNTRY_CODE_MY             41      /* "Malaysia"     */
#define M1X22_CNTRY_CODE_MT             42      /* "Malta"        */
#define M1X22_CNTRY_CODE_MX             43      /* "Mexico"       */
```

```
#define M1X22_CNTRY_CODE_MA             44      /* "Morocco"      */
#define M1X22_CNTRY_CODE_NL             45      /* "Netherlands"  */
#define M1X22_CNTRY_CODE_NZ             46      /* "New Zealand"  */
#define M1X22_CNTRY_CODE_NG             47      /* "Nigeria"      */
#define M1X22_CNTRY_CODE_NO             48      /* "Norway"       */
#define M1X22_CNTRY_CODE_OM             49      /* "Oman"         */
#define M1X22_CNTRY_CODE_PK             50      /* "Pakistan"     */
#define M1X22_CNTRY_CODE_PR             51      /* "Peru"         */
#define M1X22_CNTRY_CODE_PH             52      /* "Philippines"  */
#define M1X22_CNTRY_CODE_PL             53      /* "Poland"       */
#define M1X22_CNTRY_CODE_PT             54      /* "Portugal"     */
#define M1X22_CNTRY_CODE_RO             55      /* "Romania"      */
#define M1X22_CNTRY_CODE_RU             56      /* "Russia"       */
#define M1X22_CNTRY_CODE_SA             57      /* "Saudi Arabia" */
#define M1X22_CNTRY_CODE_SG             58      /* "Singapore"    */
#define M1X22_CNTRY_CODE_SK             59      /* "Slovakia"     */
#define M1X22_CNTRY_CODE_SI             60      /* "Slovenia"     */
#define M1X22_CNTRY_CODE_ZA             61      /* "S. Africa"    */
#define M1X22_CNTRY_CODE_KR             62      /* "S. Korea"     */
#define M1X22_CNTRY_CODE_ES             63      /* "Spain"        */
#define M1X22_CNTRY_CODE_SE             64      /* "Sweden"       */
#define M1X22_CNTRY_CODE_CH             65      /* "Switzerland"  */
#define M1X22_CNTRY_CODE_SY             66      /* "Syria"        */
#define M1X22_CNTRY_CODE_TW             67      /* "Taiwan"       */
#define M1X22_CNTRY_CODE_TH             68      /* "Thailand"     */
#define M1X22_CNTRY_CODE_AE             69      /* "UAE"          */
#define M1X22_CNTRY_CODE_UK             70      /* "UK"           */
#define M1X22_CNTRY_CODE_US             71      /* "USA"          */
#define M1X22_CNTRY_CODE_YE             72      /* "Yemen"        */
```

7.2 M1X22_CNTRY_STRUCT_t

Description

This structure defines the country default parameters.

Prototype

```
typedef struct m1X22_cntry_struct
{
    unsigned int       cnum;              /* Country code      */
    unsigned char      ccode[4];          /* Two letter internet country code */
    unsigned char      country[16];       /* Country Name            */
    unsigned int       ac_impedance;
    unsigned int       dc_vi_mask;
    unsigned int       rgth_value;
    int                auto_cid_enable;   /* automatically enable CID */
    int                use_seize_state;   /* ring tone, silent duration */
} M1X22_CNTRY_STRUCT_t;
```

Parameters

Data Type	Name	Transmit
unsigned int	cnum	Country code (see M1X22_COUNTRY_CODE).
unsigned char	ccode[4]	Two letter country code.
unsigned char	country[16]	Country name.
unsigned int	ac_impedance	AC impedance.
unsigned int	dc_vi_mask	DC VI mask.
unsigned int	rgth_value	Ring voltage threshold.
int	auto_cid_enable	Automatic CID enable.
int	use_seize_mode	Seize mode enable.

7.3 M1X22_DEBUG_TRACE_MASK

Description

Trace macros used by the `M1X22_DEBUG_LEVEL_SET` IOCTL.

Prototype

```
#define M1X22_DEBUG_TRACE          0x00000001
#define M1X22_DEBUG_ERROR          0x80000002
#define M1X22_DEBUG_INIT           0x00000004
#define M1X22_DEBUG_EVENT          0x00000008
#define M1X22_DEBUG_IOCTL          0x00000010
#define M1X22_DEBUG_BARRIER        0x00000020
#define M1X22_DEBUG_INT            0x00000040
#define M1X22_DEBUG_KPROC          0x00000080
#define M1X22_DEBUG_COUNTRY_CODE   0x00000100
#define M1X22_DEBUG_RING_PATH      0x00000200
#define M1X22_DEBUG_LINE_STATE     0x00000400
#define M1X22_DEBUG_PHU            0x00000800
#define M1X22_DEBUG_LOOPBACK       0x00001000
#define M1X22_DEBUG_GPIO           0x00002000
#define M1X22_DEBUG_MAFE           0x00004000
#define M1X22_DEBUG_DIAL           0x00008000
```

7.4 M1X22_LAST_ERROR_CODE

Description

The last error code can be retrieved by this IOCTL. This is the list of the driver's last error code.

Prototype

```
/*****************************************************************
**  73M1X22 Driver Last Error Code
*****************************************************************/
#define M1X22_ERR_OK                    0x00000000  /* NO Error */
#define M1X22_ERR_INVALID_GPIO_NUM      0x00000001  /* Invalid GPIO number */
#define M1X22_ERR_INVALID_CNTRY_CODE    0x00000002  /* Invalid country code */
#define M1X22_ERR_INVALID_PARAM         0x00000003  /* Invalid parameter */
#define M1X22_ERR_INVALID_STATE         0x00000004  /* Invalid state for the command */
#define M1X22_ERR_INVALID_IOCTL         0x00000005  /* Invalid ioctl */
#define M1X22_ERR_INVALID_FD            0x00000005  /* Invalid File Descriptor */
#define M1X22_ERR_COPY_TO_USER          0x00000020  /* memcopy to user failed */
#define M1X22_ERR_COPY_FROM_USER        0x00000021  /* memcopy from user failed */
#define M1X22_ERR_PLL_NOT_LOCKED        0x00000022  /* PLL not locked */
#define M1X22_ERR_BARRIER_NOT_SYNC      0x00000023  /* Barrier not synced */
#define M1X22_ERR_NO_EVENT_DATA         0x00000024  /* No event data available */
```

7.5 struct txrx_gain

Description

This structure is used by the M1X22_PHONE_VOLUME_SET IOCTL to adjust the volume setting of the speakerphone and microphone.

Prototype

```
struct txrx_gain {
    int   tx_gain;
    int   rx_gain;
};
```

Parameters

Data Type	Name	Transmit
int	tx_gain	Transmit level in dBm.
Int	rx_gain	Receive level in dB.

7.6 M1X22_PULSE_DIAL_t

Description

This structure is used by the M1X22_ATDP IOCTLs for pulse dialing.

Prototype

```
typedef struct m1X22_pulse_dial_struct {
    unsigned int    length;              /* digit length */
    unsigned char   digits[MAX_PHONE_NMBR_DIGIT_CNT]; /* pulse dial digits */
}
M1X22_PULSE_DIAL_t;
```

Parameters

Data Type	Name	Description
unsigned int	length	Digit length.
Unsigned char	digits	Pulse dial digits.

7.7 M1X22_PULSE_DIAL_PARAM_t

Description

This structure is used by the M1X22_ATDP_PARAM IOCTLs to read or modify the pulse dial parameter. The command field indicates whether it is a read or a write operation.

Prototype

```
typedef struct m1X22_pulse_dial_param_struct {
    unsigned int    command;              /* pulse dial param command  */
    unsigned int    onhook_duration;      /* oh-hook duration    */
    unsigned int    offhook_duration;     /* off-hook duration   */
    unsigned int    inter_digit_duration; /* inter-digit duration */
}
M1X22_PULSE_DIAL_PARAM_t;
```

Parameters

Data Type	Name	Description
unsigned int	command	Pulse dial param command: 0 – Read. 1 – Write.
unsigned int	onhook_duration	On-hook duration.
unsigned int	offhook_duration	Off-hook duration.
unsigned int	intra_digit_duration	Intra-digit duration.

7.8 M1X22_THRESH_OVERRIDE_t

Description

Various modem channel parameters are conveniently grouped and predefined in the country parameter setting as specified in section x. These parameters are programmed during channel initialization IOCTL (M1X22_CH_INIT). However, these parameters can be overridden at runtime to fine tune to the desired threshold for the specific installation using this IOCTL.

Prototype

```
typedef struct
{
    unsigned char acz;          /* Active Termination Loop */
    unsigned char dciv;         /* DC current voltage charac. control */
    unsigned char rgth;         /* Ring threshold
}
M1X22_THRESH_OVERRIDE_t;
```

Parameters

Data Type	Name	Description
unsigned char	acz	Active termination loop.
unsigned char	dciv	DC current voltage characteristic control.
unsigned char	rgth	Ring threshold.

7.9 M1X22_SAMPLE_RATE_SELECTION

Description

This is the enumerated sample rate selection. It is used by the M1X22_SET_SAMPLING_FREQ IOCTL to select the PCM sample rate.

Prototype

```
typedef enum
{
M1X22_XTAL_27000KHZ_FS_07200HZ =  1,      /* Xtal = 27.000 Mhz FS = 7.2KHz */
M1X22_XTAL_27000KHZ_FS_08000HZ =  2,      /* Xtal = 27.000 Mhz FS = 8.0KHz */
M1X22_XTAL_27000KHZ_FS_09600HZ =  3,      /* Xtal = 27.000 Mhz FS = 9.6KHz */
M1X22_XTAL_27000KHZ_FS_12000HZ =  4,      /* Xtal = 27.000 Mhz FS = 12.0KHz */
M1X22_XTAL_27000KHZ_FS_14400HZ =  5,      /* Xtal = 27.000 Mhz FS = 14.4KHz */
M1X22_XTAL_27000KHZ_FS_16000HZ =  6,      /* Xtal = 27.000 Mhz FS = 16.0KHz */
M1X22_XTAL_24576KHZ_FS_07200HZ =  7,      /* Xtal = 24.576 Mhz FS = 7.2KHz */
M1X22_XTAL_24576KHZ_FS_08000HZ =  8,      /* Xtal = 24.576 Mhz FS = 8.0KHz */
M1X22_XTAL_24576KHZ_FS_09600HZ =  9,      /* Xtal = 24.576 Mhz FS = 9.6KHz */
M1X22_XTAL_24576KHZ_FS_12000HZ = 10,      /* Xtal = 24.576 Mhz FS = 12.0KHz */
M1X22_XTAL_24576KHZ_FS_14400HZ = 11,      /* Xtal = 24.576 Mhz FS = 14.4KHz */
M1X22_XTAL_24576KHZ_FS_16000HZ = 12,      /* Xtal = 24.576 Mhz FS = 16.0KHz */
M1X22_XTAL_09216KHZ_FS_07200HZ = 13,      /* Xtal =  9.216 Mhz FS = 7.2KHz */
M1X22_XTAL_09216KHZ_FS_08000HZ = 14,      /* Xtal =  9.216 Mhz FS = 8.0KHz */
M1X22_XTAL_09216KHZ_FS_09600HZ = 15,      /* Xtal =  9.216 Mhz FS = 9.6KHz */
M1X22_XTAL_09216KHZ_FS_12000HZ = 16,      /* Xtal =  9.216 Mhz FS = 12.0KHz */
M1X22_XTAL_09216KHZ_FS_14400HZ = 17,      /* Xtal =  9.216 Mhz FS = 14.4KHz */
M1X22_XTAL_09216KHZ_FS_16000HZ = 18,      /* Xtal =  9.216 Mhz FS = 16.0KHz */
}
M1X22_MAFE_FREQUENCY;
```

Parameters

Name	Value	Description
M1X22_XTAL_27000KHZ_FS_07200HZ	1	Set sample rate at 27.0 MHz assuming a crystal frequency of 7.2 kHz.
M1X22_XTAL_27000KHZ_FS_08000HZ	2	Set sample rate at 27.0 MHz assuming a crystal frequency of 8.0 kHz.
M1X22_XTAL_27000KHZ_FS_09600HZ	3	Set sample rate at 27.0 MHz assuming a crystal frequency of 9.6 kHz.
M1X22_XTAL_27000KHZ_FS_12000HZ	4	Set sample rate at 27.0 MHz assuming a crystal frequency of 12.0 kHz.
M1X22_XTAL_27000KHZ_FS_14400HZ	5	Set sample rate at 27.0 MHz assuming a crystal frequency of 14.4 kHz.
M1X22_XTAL_27000KHZ_FS_16000HZ	6	Set sample rate at 27.0 MHz assuming a crystal frequency of 16.0 kHz.
M1X22_XTAL_24576KHZ_FS_07200HZ	7	Set sample rate at 24.576 MHz assuming a crystal frequency of 7.2 kHz.
M1X22_XTAL_24576KHZ_FS_08000HZ	8	Set sample rate at 24.576 MHz assuming a crystal frequency of 8.0 kHz.
M1X22_XTAL_24576KHZ_FS_09600HZ	9	Set sample rate at 24.576 MHz assuming a crystal frequency of 9.6 kHz.
M1X22_XTAL_24576KHZ_FS_12000HZ	10	Set sample rate at 24.576 MHz assuming a crystal frequency of 12.0 kHz.

Name	Value	Description
M1X22_XTAL_24576KHZ_FS_14400HZ	11	Set sample rate at 24.576 MHz assuming a crystal frequency of 14.4 kHz.
M1X22_XTAL_24576KHZ_FS_16000HZ	12	Set sample rate at 24.576 MHz assuming a crystal frequency of 16.0 kHz.
M1X22_XTAL_09216KHZ_FS_07200HZ	13	Set sample rate at 9.216 MHz assuming a crystal frequency of 7.2 kHz.
M1X22_XTAL_09216KHZ_FS_08000HZ	14	Set sample rate at 9.216 MHz assuming a crystal frequency of 8.0 kHz.
M1X22_XTAL_09216KHZ_FS_09600HZ	15	Set sample rate at 9.216 MHz assuming a crystal frequency of 9.6 kHz.
M1X22_XTAL_09216KHZ_FS_12000HZ	16	Set sample rate at 9.216 MHz assuming a crystal frequency of 12.0 kHz.
M1X22_XTAL_09216KHZ_FS_14400HZ	17	Set sample rate at 9.216 MHz assuming a crystal frequency of 14.4 kHz.
M1X22_XTAL_09216KHZ_FS_16000HZ	18	Set sample rate at 9.216 MHz assuming a crystal frequency of 16.0 kHz.

7.10 Billing Tone Filter Related Data Type and Structure

7.10.1 M1X22_BTONE_FILTER_COMMAND

Description

This is the billing tone filter commands. For the enable command the tone frequency parameter is expected in the Call Progress Monitor Data Type and Structure
.

Prototype

```
typedef enum
{
        M1X22_BTONE_FILTER_DISABLE   = 0,   /* Disable billing tone filter */
        M1X22_BTONE_FILTER_ENABLE    = 1    /* Enable billing tone filter */
}
M1X22_BTONE_FILTER_COMMAND;
```

Parameters

Name	Value	Description
M1X22_BTONE_FILTER_DISABLE	0	Disable billing tone filter.
M1X22_BTONE_FILTER_ENABLE	1	Enable billing tone filter.

7.10.2 M1X22_BTONE_FREQUENCY

Description

This is the list of tone frequencies that can be filtered out. The user must select the right frequency to effectively filter out the billing tone.

Prototype

```
typedef enum
{
        M1X22_BTONE_FREQ_12KHZ       = 0,      /* 12KHz (F1) */
        M1X22_BTONE_FREQ_16KHZ       = 1       /* 16KHz (F2) */
}
M1X22_BTONE_FREQUENCY;
```

Parameters

Name	Value	Description
M1X22_BTONE FREQ_12KHZ	0	12 kHz billing tone.
M1X22_BTONE FREQ_16KHZ	1	16 kHz billing tone.

7.10.3 M1X22_BTONE_FILTER_t

Description

This structure is used by the M1X22_BTONE_FILTER IOCTL to enable or to disable the billing tone. The filter is by default disable upon initialization and can be enabled to reject a tone of specific frequency using this IOCTL.

Prototype

```
typedef struct m1X22_btone_filter
{
        M1X22_BTONE_FILTER_COMMAND command;    /* command */
        M1X22_BTONE_FREQUENCY      frequency;  /* billing tone frequency */
}
M1X22_BTONE_FILTER_t;
```

Parameters

Data Type	Name	Description
unsigned char	acz	Active termination loop.
unsigned char	dciv	DC current voltage characteristic control.
unsigned char	rgth	Ring threshold.

7.11 Call Progress Monitor Data Type and Structure

7.11.1 M1X22_CPROG_MON_VOLT_REF

Description

This contains the list of supported voltage reference at the call progress monitor audio output. This is used in the M1X22_CPROG_MONITOR IOCTL to select the voltage reference.

Prototype

```
typedef enum
{
        M1X22_CPROG_MON_VOLT_REG_1_5       = 0, /* 1.5 Vdc   */
        M1X22_CPROG_MON_VOLT_REF_VCC_DIV_2 = 1  /* VCC/2 Vdc */
}
M1X22_CPROG_MON_VOLT_REF;
```

Parameters

Name	Value	Description
M1X22_CPROG_MON_VOLT_REG_1_5	0	1.5 Vdc reference selection.
M1X22_CPROG_MON_VOLT_REF_VCC_DIV	1	VCC/2 Vdc reference selection.

7.11.2 M1X22_CPROG_MON_GAIN

Description

This contains the list of supported gain settings of the audio path. It is used by the M1X22_CPROG_MONITOR IOCTL to set transmit and receive gain of the call progress monitor auto path.

Prototype

```
typedef enum
{
        M1X22_CPROG_MON_GAIN_0DB           = 0, /* Gain setting of 0dB   */
        M1X22_CPROG_MON_GAIN_MINUS_6DB     = 1, /* Gain setting of -6dB  */
        M1X22_CPROG_MON_GAIN_MINUS_12DB    = 2, /* Gain setting of -12dB */
        M1X22_CPROG_MON_GAIN_MUTE          = 3  /* Mute                  */
}
M1X22_CPROG_MON_GAIN;
```

Parameters

Name	Value	Description
M1X22_CPROG_MON_GAIN_0DB	0	Gain setting of 0dB.
M1X22_CPROG_MON_GAIN_MINUS_6DB	1	Gain setting of -6dB.
M1X22_CPROG_MON_GAIN_MINUS_12DB	2	Gain setting of -12dB.
M1X22_CPROG_MON_GAIN_MUTE	3	Mute.

7.11.3 M1X22_CPROG_MONITOR_t

Description

This structure is used by the M1X22_CPROG_MONITOR IOCTL to adjust (or mute) the gain setting of the call progress monitor audio path.

Prototype

```
typedef struct {
        M1X22_CPROG_MON_VOLT_REF voltage_ref;   /* Voltage reference */
        M1X22_CPROG_MON_GAIN     tx_gain;       /* Tx path gain setting */
        M1X22_CPROG_MON_GAIN     rx_gain;       /* Rx path gain setting */
}
M1X22_CPROG_MONITOR_t;
```

Parameters

Data Type	Name	Description
M1X22_CPROG_MON_VOLT_REF	voltage_ref	Voltage reference.
M1X22_CPROG_MON_GAIN	tx_gain	Transmit gain.
M1X22_CPROG_MON_GAIN	rx_gain	Receive gain.

7.12 GPIO Related Data Type and Structures

7.12.1 M1X22_GPIO_NUMBER

Description

This is the list of GPIO pin definitions.

Prototype

```
typedef enum
{
        M1X22_GPIO_NUM_4 = 0x10,                /* GPIO-4              */
        M1X22_GPIO_NUM_5 = 0x20,                /* GPIO-5              */
        M1X22_GPIO_NUM_6 = 0x40,                /* GPIO-6              */
        M1X22_GPIO_NUM_7 = 0x80                 /* GPIO-7              */
}
M1X22_GPIO_NUMBER;
```

Parameters

Name	Value	Description
M1X22_GPIO_NUM_4	0x10	GPIO4.
M1X22_GPIO_NUM_5	0x20	GPIO5.
M1X22_GPIO_NUM_6	0x40	GPIO6.
M1X22_GPIO_NUM_7	0x80	GPIO7.

7.12.2 M1X22_GPIO_CONFIG_COMMAND

Description

This is the GPIO configuration command. The GPIO can be configured using the M1X22_GPIO_CONFIG_SET command. Its configuration can be read using M1X22_GPIO_CONFIG_GET command. The GPIO must be enabled using the M1X22_GPIO_CONTROL IOCTL for the new configuration to take effect.

Prototype

```
typedef enum
{
        M1X22_GPIO_CONFIG_GET = 0,              /* GPIO config GET     */
        M1X22_GPIO_CONFIG_SET = 1               /* GPIO config SET     */
}
M1X22_GPIO_CONFIG_COMMAND;
```

Parameters

Name	Value	Description
M1X22_GPIO_CONFIG_GET	0	Read the GPIO configuration.
M1X22_GPIO_CONFIG_SET	1	Write the GPIO configuration.

7.12.3 M1X22_GPIO_CONTROL_TYPE

Description

GPIO control type definitios.

Prototype

```
typedef enum
{
        M1X22_GPIO_CONTROL_DISABLE = 0,         /* disable GPIO       */
        M1X22_GPIO_CONTROL_ENABLE  = 1          /* enable GPIO        */
}
M1X22_GPIO_CONTROL_TYPE;
```

Parameters

Name	Value	Description
M1X22_GPIO_CONTROL_DISABLE	0	Disable GPIO.
M1X22_GPIO_CONTROL_ENABLE	1	Enable GPIO.

7.12.4 M1X22_GPIO_DATA_COMMAND

Description

GPIO data access command. If the M1X22_GPIO_SIGNAL_DIRECTION is set to M1X22_GPIO_DIR_INPUT, perform the M1X22_GPIO_DATA_GET returns the logical value of type M1X22_GPIO_DATA_TYPE of the appropriate GPIO as an input. If the M1X22_GPIO_SIGNAL_DIRECTION is set to M1X22_GPIO_DIR_OUTPUT, the corresponding GPIO port outputs the logical value as written.

Prototype

```
typedef enum
{
        M1X22_GPIO_DATA_GET   = 0,              /* Read GPIO data     */
        M1X22_GPIO_DATA_SET   = 1               /* Write GPIO data    */
}
M1X22_GPIO_DATA_COMMAND;
```

Parameters

Name	Value	Description
M1X22_GPIO_DATA_GET	0	Read GPIO data.
M1X22_GPIO_DATA_SET	1	Write GPIO data.

7.12.5 M1X22_GPIO_DATA_TYPE

Description

GPIO data types – this is the GPIO data returned from the M1X22_GPIO_DATA_GET access command, or data to be written to the GPIO port using the M1X22_GPIO_DATA_SET access command.

Prototype

```
typedef enum
{
        M1X22_GPIO_DATA_LOW   = 0,              /* GPIO data - low    */
        M1X22_GPIO_DATA_HIGH  = 1               /* GPIO data - high   */
}
M1X22_GPIO_DATA_TYPE;
```

Parameters

Name	Value	Description
M1X22_GPIO_DATA_LOW	0	GPIO data – low.
M1X22_GPIO_DATA_HIGH	1	GPIO data – high.

7.12.6 M1X22_GPIO_SIGNAL_DIRECTION

Description

GPIO pin signal direction. This control bit is used to designate the GPIO pin as either input or output.

Prototype

```
typedef enum {
    M1X22_GPIO_DIR_INPUT  = 0,      /* GPIO pin signal direction – INPUT  */
    M1X22_GPIO_DIR_OUTPUT = 1       /* GPIO pin signal direction - OUTPUT */
}
M1X22_GPIO_SIGNAL_DIRECTION;
```

Parameters

Name	Value	Description
M1X22_GPIO_DIR_INPUT	0	GPIO pin signal direction – INPUT.
M1X22_GPIO_DIR_OUTPUT	1	GPIO pin signal direction – OUTPUT.

7.12.7 M1X22_GPIO_INTR_POLARITY

Description

GPIO Interrupt signal transition edge selection. The defines the interrupt source as being either on a rising or a falling edge of the corresponding GPIO pin. If configured as `M1X22_GPIO_POL_RISING` a rising edge will trigger an interrupt from the corresponding GPIO pin. If configured as `M1X22_GPIO_POL_FALLING` a falling edge will trigger an interrupt from the corresponding GPIO pin.

Prototype

```
typedef enum {
    M1X22_GPIO_POL_RISING  = 0,  /* Sig transition edge polarity - RISING  */
    M1X22_GPIO_POL_FALLING = 1   /* Sig transition edge polarity - FALLING */
}
M1X22_GPIO_INTR_POLARITY;
```

Parameters

Name	Value	Description
M1X22_GPIO_POL_RISING	0	Interrupt edge selection – RISING.
M1X22_GPIO_POL_FALLING	1	Interrupt edge selection – FALLING.

7.12.8 M1X22_GPIO_CONFIG_t

Description

This structure is used by the `M1X22_GPIO_CONFIG` IOCTLs to read or write GPIO configuration. The first field in the config structure is the command action field – GET or SET. For the GET command the "direction" and "polarity" fields are returned if successful.

Prototype

```
typedef struct gpio_config
{
        M1X22_GPIO_CONFIG_COMMAND command;     /* command */
        M1X22_GPIO_NUMBER         gpio;        /* GPIO number     */
        M1X22_GPIO_SIGNAL_DIR     direction;   /* signal direction */
        M1X22_GPIO_INTR_POLARITY  polarity;    /* intr edge selection */
}
M1X22_GPIO_CONFIG_t;
```

Parameters

Data Type	Name	Description
M1X22_GPIO_CONFIG_COMMAND	command	GET or SET GPIO config command.
M1X22_GPIO_NUMBER	gpio	GPIO number.
M1X22_GPIO_DIRECTION	direction	GPIO pin direction – input/output.
M1X22_GPIO_INTR_POLARITY	polarity	Interrupt on signal transition edge polarity.

7.12.9 M1X22_GPIO_DATA_t

Description

This structure is used by the M1X22_GPIO_DATA IOCTLs to read or write GPIO signal from or to the GPIO pin.

Prototype

```
typedef struct gpio_data
{
        M1X22_GPIO_DATA_COMMAND   command;     /* command */
        M1X22_GPIO_NUMBER         gpio;        /* GPIO number */
        M1X22_GPIO_DATA_TYPE      data;        /* data */
}
M1X22_GPIO_DATA_t;
```

Parameters

Data Type	Name	Description
M1X22_GPIO_DATA_COMMAND	command	Read or Write command.
M1X22_GPIO_NUMBER	gpio	GPIO number.
M1X22_GPIO_DATA_TYPE	data	Signal level to be written to the GPIO pin.

7.12.10 M1X22_GPIO_CONTROL_t

Description

This structure is used by the M1X22_GPIO_CONTROL IOCTLs to enable or disable the operation of the GPIO.

Prototype

```
typedef struct gpio_control
{
        M1X22_GPIO_CONTROL_TYPE   control;     /* control - enable/disable */
        M1X22_GPIO_NUMBER         gpio;        /* gpio */
}
M1X22_GPIO_CONTROL_t;
```

Parameters

Data Type	Name	Description
M1X22_GPIO_CONTROL_TYPE	control	Control – enable/disable GPIO.
M1X22_GPIO_NUMBER	gpio	GPIO number.

7.13 Loopback Related Data Type and Structure

7.13.1 M1X22_LOOPBACK_COMMAND

Description

These loopback commands can be used in the M1X22_LOOPBACK IOCTL to manage loopback sessions.

Prototype

```
typedef enum
{
        M1X22_LOOPBACK_CMD_GET   = 0,  /* Get the current loopback session */
        M1X22_LOOPBACK_CMD_SET   = 1,  /* Set (initiate) a loopback session */
        M1X22_LOOPBACK_CMD_CLEAR = 2   /* Clear (terminate) a loopback session */
}
M1X22_LOOPBACK_COMMAND;
```

Parameters

Name	Value	Description
M1X22_LOOPBACK_CMD_GET	0	Get the type of current loopback session.
M1X22_LOOPBACK_CMD_SET	1	Initiate a loopback session.
M1X22_LOOPBACK_CMD_CLEAR	2	Terminate a loopback session.

7.13.2 M1X22_LOOPBACK_MODE

Description

The driver supports five loopback modes. Refer to the *73M1822/73M1922 Data Sheet* or those loopback reference points.

Prototype

```
typedef enum
{
        M1X22_LOOPBACK_MODE_NONE     = 0,       /* No loopback         */
        M1X22_LOOPBACK_MODE_DIGLB1   = 1,       /* Digital Loopback-1  */
        M1X22_LOOPBACK_MODE_INTLB1   = 2,       /* Internal Loopback-1 */
        M1X22_LOOPBACK_MODE_DIGLB2   = 3,       /* Digital Loopback-2  */
        M1X22_LOOPBACK_MODE_INTLB2   = 4,       /* Internal Loopback-2 */
        M1X22_LOOPBACK_MODE_ALB      = 5        /* Analog Loopback     */
}
M1X22_LOOPBACK_MODE;
```

Parameters

Name	Value	Description
M1X22_LOOPBACK_MODE_NONE	0	No Loopback.
M1X22_LOOPBACK MODE DIGLB1	1	Digital Loopback-1.
M1X22_LOOPBACK MODE INTLB1	2	Internal Loopback-1.
M1X22_LOOPBACK MODE DIGLB2	3	Digital Loopback-2.
M1X22_LOOPBACK MODE INTLB2	4	Internal Loopback-2.
M1X22_LOOPBACK MODE ALB	5	Analog Loopback.

7.13.3 M1X22_LOOPBACK_t

Description

This is the structure used for managing the loopback with the M1X22_LOOPBACK IOCTL.

Prototype

```
typedef struct m1X22_loopback_struct
{
        M1X22_LOOPBACK_COMMAND command;
        M1X22_LOOPBACK_MODE    mode;
}
M1X22_LOOPBACK_t;
```

Parameters

Name	Value	Description
M1X22_LOOPBACK_COMMAND	Command	Loopback command.
M1X22_LOOPBACK_MODE	type	Loopback mode.

7.14 Line Measurement Related Data Types and Structures

7.14.1 M1X22_MEASURE_ENTITY

Description

This is the enumerated list of line measuring entities that can be monitored by the driver.

Prototype

```
typedef enum
{
    M1X22_MEASURE_ENTITY_CURRENT = 0,
    M1X22_MEASURE_ENTITY_VOLTAGE = 1,
}
M1X22_MEASURE_ENTITY;
```

Parameters

Name	Value	Description
M1X22_MEASURE_ENTITY_CURRENT	0	Line current measuring entity.
M1X22_MEASURE_ENTITY_VOLTAGE	1	Line voltage measuring entity.

7.14.2 M1X22_MEASURE_ACTION

Description

This is the enumerated list of possible request actions that can be performed on Interval and Event Table (IET). See `M1X22_IET_t` for a description of the IET table.

Prototype

```
typedef enum
{
    M1X22_MEASURE_ACTION_GET = 0,
    M1X22_MEASURE_ACTION_SET = 1,
    M1X22_MEASURE_ACTION_SET = 2
}
M1X22_MEASURE_ACTION;
```

Parameters

Name	Value	Description
M1X22_MEASURE_ACTION_GET	0	Read the IET table entry.
M1X22_MEASURE_ACTION_SET	1	Update the IET table entry.
M1X22_MEASURE_ACTION_CLEAR	2	Disable the IET table entry.

7.14.3 M1X22_MEASURE_START_STOP_t

Description

This structure is used by the M1X22_MEASURE_START and M1X22_MEASURE_STOP IOCTLs to start and stop the current or voltage measurement.

Prototype

```
typedef struct
{
    M1X22_MEASURE_ENTITY entity;
    unsigned int         sample_time;
    unsigned int         average_sample_count;
}
M1X22_MEASURE_START_t;
```

Parameters

Data Type	Name	Description
M1X22_MEASURE_ENTITY	entity	Measuring entity – current or voltage.
unsigned int	sample_time	Interval between two samplings in milliseconds (not applicable for the M1X22_MEASURE_STOP IOCTL).
unsigned int	average_sample_count	Sample count for average calculation (not applicable for the M1X22_MEASURE_STOP IOCTL.

7.14.4 M1X22_MEASURE_UPDATE_t

Description

This structure is used by the M1X22_MEASURE_UPDATE IOCTL to send the IET table entry parameters for update, or to read its content. The action parameter in the structure indicates the desired operation. The GET operation performs the read function while the SET operation performs write or modify function.

Prototype

```
typedef struct
{
    unsigned int row;
    M1X22_MEASURE_ENTITY entity;
    M1X22_MEASURE_ACTION action;
    unsigned int inverval_min;
    unsigned int interval_max;
    unsigned int event;
}
M1X22_MEASURE_UPDATE_t;
```

Parameters

Data Type	Name	Description
unsigned int	row	IET table index (0 to 9).
M1X22_MEASURE_ENTITY	entity	Measuring entity – current or voltage.
M1X22_MEASURE_ACTION	action	Requested action – GET, SET or CLEAR.
unsigned int	interval_min	Lower bound range (in milliamps for current, or millivolts in voltage).
unsigned int	interval_max	Upper bound range (in milliamps for current, or millivolts in voltage).
unsigned int	event	Event identifier sent to application layer.

7.14.5 M1X22_IET_t

Description

This Interval and Event Table structure (IET) is internally used by the driver to store threshold parameters for monitoring line current and voltage. Each measuring entity (current and voltage) has its own separate IET table of up to 10 entries, or rows with the following attributes: the IET row number, lower and upper bound thresholds, and an event identifier that will be sent in `M1X22_LINE_STATE` event to the application when this IET entry is entered.

The IET table is managed by the application layer via a group of line measurement IOCTLs. Management of IET table entry consists of runtime update of IET table, and the start and stop measurement processing which access these IET tables.

Prototype

```
typedef struct
{
    unsigned int row;
    unsigned int inverval_min;
    unsigned int interval_max;
    unsigned int event;
}
M1X22_IET_t;
```

Parameters

Data Type	Name	Description
`unsigned int`	`row`	IET table index (0 to 9).
`unsigned int`	`interval_min`	Lower bound range (in milliamps for current, or millivolts for voltage).
`unsigned int`	`interval_max`	Upper bound range (in milliamps for current, or millivolts for voltage).
`unsigned int`	`event`	Event identifier sent to application layer.

8 Driver Source and Include Files

The driver software is written exclusively in the C programming language and consists of the following files.

Table 7: Driver Source Code Files

File Name	Directory	Description
drv_config.h	user	User defined parameters.
drv_version.h	include	Defines the driver version number.
drv_sys_os.h	include	Include OS specific files.
drv_sys_os_linux.h	include	Include Linux specific macros/defines.
tsc_1x22_ctl_ioctls.h	include	Type definition for external interface.
tsc_1x22_ctl.h	include	Type definition for internal interface.
tsc-1x22_ctl_regs.h	include	1x22 Hardware Register definitions.
tsc_1x22_ctl_module.c	src	Main Linux module source code.
tsc_1x22_ctl_linux.c	src	Linux specific functions.
tsc_1x22_ctl_register.c	src	Wrapper functions to access device registers.
tsc_1x22_ctl_iproc.c	src	Interrupt support thread/function code.
tsc_1x22_ctl_kproc.c	src	Event/IOCTL support thread/function code.
tsc_1x22_ctl_functions.c	src	Helper functions supporting all functional aspects.
tsc_1x22_ctl_cntry_tbl.c	src	Country Code related structures.

9 Related Documentation

The following 73M1x22 documents are available from Teridian Semiconductor Corporation:

73M182273M1X22 Data Sheet
73M182273M1X22 Layout Guidelines
73M1x22 Worldwide Design Guide
73M1822/73M1922 Control Module User Guide
73M1822/73M1922 Hardware Module for SMDK412 User Guide
73M1822/73M1922 Modem API User Guide
73M1822/73M1922 Modem CTL Application User Guide
73M1822/73M1922 MicroDAA Software Architecture

10 Contact Information

For more information about Teridian Semiconductor products or to check the availability of the 73M1822 and 73M1922, contact us at:

6440 Oak Canyon Road
Suite 100
Irvine, CA 92618-5201

Telephone: (714) 508-8800
FAX: (714) 508-8878
Email: modem.support@teridian.com

For a complete list of worldwide sales offices, go to http://www.teridian.com.

Appendix A – Country Codes

Table 8 provides all the defined countries, their corresponding two-character codes and their indexes.

Table 8: Country Code Table

Country Name	Code	Index	Country Name	Code	Index
Argentina	AR	0	Latvia	LV	37
Australia	AU	1	Lebanon	LB	38
Austria	AT	2	Luxembourg	LU	39
Bahrain	BH	3	Macao	MO	40
Belgium	BE	4	Malaysia	MY	41
Brazil	BR	5	Malta	MT	42
Bulgaria	BG	6	Mexico	MX	43
Canada	CA	7	Morocco	MA	44
Chili	CL	8	Netherlands	NL	45
China1	C1	9	New Zealand	NZ	46
China2	C2	10	Nigeria	NG	47
Columbia	CO	11	Norway	NO	48
Croatia	HR	12	Oman	OM	49
TBR 21	TB	13	Pakistan	PK	50
Cyprus	CY	14	Peru	PR	51
Czech Rep	CZ	15	Philippines	PH	52
Denmark	DK	16	Poland	PL	53
Ecuador	EC	17	Portugal	PT	54
Egypt	EG	18	Romania	RO	55
El Salvador	SV	19	Russia	RU	56
Finland	FI	20	Saudi Arabia	SA	57
France	FR	21	Singapore	SG	58
Germany	DE	22	Slovakia	SK	59
Greece	GR	23	Slovenia	SI	60
Gum	GU	24	S. Africa	ZA	61
Hong Kong	HK	25	S. Korea	KR	62
Hungary	HU	26	Spain	ES	63
Iceland	IS	27	Sweden	SE	64
India	IN	28	Switzerland	CH	65
Indonesia	ID	29	Syria	SY	66
Ireland	IE	30	Taiwan	TW	67
Israel	IL	31	Thailand	TH	68
Italy	IT	31	UAE	AE	69
Japan	JP	33	UK	UK	70
Jordan	JO	34	USA	US	71
Kazakhstan	KZ	35	Yemen	YE	72
Kuwait	KW	36			

Revision History

Revision	Date	Description
1.0	12/23/2009	First publication.