TERIDIAN SEMICONDUCTOR CORP.

Simplifying System Integration™

73M1866B/73M1966B FXOAPI User Guide

November 2, 2009
Rev. 2.1
UG_1x66B_046

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 application programming interface (API) of the Teridian M1x66 Reference Driver. The API is defined as a simplified functional interface allows an application program written in "C" language to communicate with the driver without going through complicate and cumbersome Linux® driver IOCTL. In addition, the API subsystem also encapsulates the FXO asynchronous event notification and provides a much simpler mechanism to the user application via a callback function.

The API subsystem resides between the user application and the driver layer as shown in Figure 1.

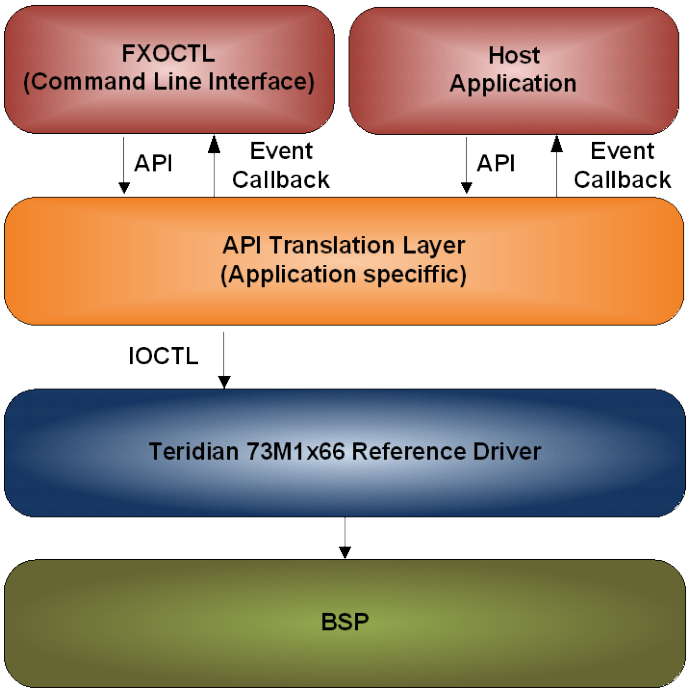

Figure 1: Driver API

1.1 Overview

The FXOAPI is a user-friendly programming conduit used to access and manage the FXO device/channel via the Reference Driver. This section describes the programming flow and how the application program can be implemented using the FXOAPIs.

First, the concept of device versus channel, the Teridian FXO driver supports multiple FXO channels. The application layer communicates with the driver using this channel descriptor. However, the driver reports asynchronous FXO channel events through the device descriptor. At this point, the only use for the device descriptor is for conveying event notification and for retrieving events from the driver.

The application program must first initialize the FXOAPI by calling FXO API Initialization and Termination M1x66_FxoApiInit(). This initializes internal data structures and prepares the API subsystem for subsequent calls to other APIs.

After the FXO API is initialized, the application program opens the FXO device and channel descriptors to access and manage the FXO device and channel(s) by calling Device/Channel Open, Close and Initialization APIs
M1x66_OpenDevice() and M1x66_OpenChannel() respectively. If more than one FXO channel exists in the hardware platform, each channel is opened separately by calling M1x66_OpenChannel().

To simplify the usage of the driver, the API encapsulates all relevant data components required to operate the FXO device and the channels into a data structure whose pointer is referred to as a “handle”. This handle is returned from a successful call to open device/channel APIs and is used throughout the life of the FXO device and channels.

The PCM Interface Config data is programmed by the M1x66_PcmIfConfig() API, and the PCM timeslot config data by the M1x66_PcmConfig() API. Generally, the PCM Interface config data remains static while the PCM timeslot may be dynamically allocated and set for each call establishment.

By default, the PCM channel is disabled and should remain disabled when there is no voice call on the channel. The application program can use the M1x66_PcmActivation() to enable and M1x66_PcmDeActivation() to disable the PCM channel as needed.

Upon exiting the application program it is recommended that all device and channels descriptors be closed, using
M1x66_CloseDevice() and M1x66_CloseChannel(), and the API subsystem be shutdown using M1x66_FxoApiRelease() to ensure that the system resources are free up when the application program terminates.

As mentioned earlier, the asynchronous FXO events from each channel are reported through the opened device descriptor. The event message contains the channel ID from which the event occurred on. The application program must distinguish the event for each channel and act accordingly. The way this event is conveyed to the application program is via the callback function provided as one of the parameters passed into Device/Channel Open, Close and Initialization APIs
M1x66_OpenDevice(). The callback function serves as a “hook” for the API subsystem and notifies the application whenever an FXO event is generated.

Section 5 provides a sample application using FXOAPI.

1.2 Conventions Used in this Guide

This document uses the following conventions:

- Software code, IOCTL names and data types 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

API – Application Programming Interface
APOH – Another Phone Off Hook
BSP – Board Support Package
DAA – Data Access Arrangement
FXO – Foreign eXchange Office
GPIO – General Purpose Input/Output
IET – Interval Event Table
IOCTL – I/O Control
NOPOH – No Phone Off Hook
PCM – Pulse-code Modulation
POH – Phone Off Hook
SPI – Serial Peripheral Interface

2 API Descriptions

This section contains the detail description of each API. Table 1 provides a summary of the APIs.

Table 1: API Overview

IOCTL Name	Description
FXO API Initialization and Termination M1x66_FxoApiInit	API subsystem initialization.
M1x66_FxoApiRelease	API subsystem release or termination.
Device/Channel Open, Close and Initialization APIs M1x66_OpenDevice	Open FXO device of event reporting.
M1x66_CloseDevice	Close an opened device.
M1x66_OpenChannel	Open FXO channel for operation.
M1x66_CloseChannel	Close an opened channel.
M1x66_InitChannel	Initialize channel for specific region.
M1x66_PCMIfConfig	Configure PCM interface parameter.
M1x66_PcmConfig	Configure PCM timeslots and codec.
M1x66_PcmActivation	Activate PCM channel.
M1x66_PcmDeActivation	De-activate PCM channel.
M1x66_HookSwitch	Perform hook switching – on/off-hook.
Hook Switch and Pulse Dial Related APIs **2.1.1 M1x66_HookSwitch** **Description** Perform on/off-hook on the FXO channel. **Prototype** `M1x66_RET M1x66_HookSwitch ( M1x66_HANDLE channel, M1x66_HOOK_SWITCH hookSwitch );` **Parameters** **Data Type** / **Name** / **Description** M1x66_HANDLE / channel / Channel handle from FXO open channel. M1x66_HOOK_SWITCH / hookSwitch / Type of hook switch requested. **Return Values** **Data Type** / **Description** M1x66_RET / M1x66_SUCCESS – Successfully performed hook switch.	Pulse dialing.

IOCTL Name	Description
M1x66_FAILED – Failure. M1x66_PulseDial	
M1x66_PulseDialCancel	Terminate pulse dial session.
M1x66_SetPulseDialParam	Adapt to the line condition instead of default values.
GPIO Management APIs M1x66_GetGpioConfig	Read a GPIO configuration.
M1x66_SetGpioConfig	Configure the GPIO.
M1x66_SetGpioData	Write data to GPIO pin.
M1x66_GetGpioData	Read data from GPIO pin.
M1x66_GpioEnable	Enable the GPIO.
M1x66_GpioDisable	Disable the GPIO.
M1x66_GetPulseDialParam	Read the pulse dial timing parameters.
Loopback Control APIs M1x66_LoopbackGet	Read the status of the loopback.
M1x66_LoopbackSet	Set the transmit/receive data path into loopback mode.
M1x66_LoopbackClear	Disable the loopback.
Country Default Parameter and Override APIs M1x66_GetCountryParam	Read the default country parameter.
M1x66_SetCountryParam	Change the default country parameter.
M1x66_GetHWRevision	Read the device hardware revision number.
Billing Tone Filter Control APIs M1x66_BillingToneFilterEnable	Filters out the billing tone.
M1x66_BillingToneFilterDisable	Disable the billing tone filter.
M1x66_ThresholdOverride	Override the FXO channel parameters.
1x66 H/W Register Access and Debugging aid APIs M1x66_HWRegisterRead	Read 1x66 register.
M1x66_HWRegisterReadAll	Read all registers.
M1x66_HWRegisterWrite	Write to 1x66 register.
M1x66_SetDebugTrace	Set debug trace mask.
M1x66_SetPhoneVolume	Set speaker phone and micro phone volume.
M1x66_CallProgressMonitor	Monitor activities on the line.
M1x66_SelectSampleRate	Set new PCM sample rate – 8 or 16 kHz.
	Get file descriptor from a handle.

IOCTL Name	Description
M1x66_GetFileDrescriptor	
Interval Event Table Management APIs M1x66_GetCurrentIET	Read current IET entry.
M1x66_UpdateCurrentIET	Update current IET entry.
M1x66_ClearCurrentIET	Delete the current IET table entry.
M1x66_GetVoltageIET	Read the voltage IET table entry.
M1x66_UpdateVoltageIET	Update voltage IET entry.
M1x66_ClearVoltageIET	Delete the voltage IET table entry.
Current/Voltage Measurement APIs M1x66_StartMeasureCurrent	Start current measurement.
M1x66_StartMeasureVoltage	Start voltage measurement.
M1x66_StopMeasureCurrent	Stop current measurement.
M1x66_StopMeasureVoltage	Stop voltage measurement.
Miscellaneous Channel Setting APIs M1x66_SetRingCadence	Set Ring Cadence filtering criteria.

2.2 FXO API Initialization and Termination

2.2.1 M1x66_FxoApiInit

Description

This API allocates and initializes required internal data structures; make necessary preparation before other API members can be invoked. The FXO API Initialization and Termination M1x66_FxoApiInit() must be called once during application program's initialization, or prior to accessing other API members. Upon exiting of the application program a reversed operation must be performed to release resources using M1x66_FxoApiRelease.

Prototype

```
M1x66_RET M1x66_FxoApiInit (void);
```

Parameters

Data Type	Name	Description
void	void	No parameter is required.

Return Values

Data Type	Description
M1x66_RET	M1x66_SUCCESS – Successful FXO API initialization. M1x66_FAILED – Failed.

2.2.2 M1x66_FxoApiRelease

Description

This API de-allocates the resources acquired during initialization - FXO API Initialization and Termination M1x66_FxoApiInit. It must be called prior to application program termination, or when access to FXOAPI is no longer needed. Any FXO channels or device remain open when this API is called will be closed.

Prototype

```
M1x66_RET M1x66_FxoApiRelease (void);
```

Parameters

Data Type	Name	Description
void	void	No parameter is required.

Return Values

Data Type	Description
M1x66_RET	M1x66_SUCCESS – Successful FXO API termination. M1x66_FAILED – Unsuccessful.

2.3 Device/Channel Open, Close and Initialization APIs

2.3.1 M1x66_OpenDevice

Description

This API performs the low level opening of the device descriptor. By default, the device descriptor name is known as "/dev/ter10"; however, since this is configurable during driver installation, a different device descriptor name may be used. Refer to driver configuration and installation more detail. Calling to this open device API is mandatory before any channel descriptor can be opened, and it must be called after the FXO API Init function.

In addition to opening the device for accessing this API also requires a callback function. This function is called by the API subsystem to convey the FOX events to the application program. Six parameters are returned when this callback function is invoked:

`eventId`: This parameter is the event identification of type `M1x66_EVENT_ID`.
`chanId`: This parameter contains the FXO channel ID from which the event is generated.
`evtData1`: are reserved for special purpose in event handling.
`evtData2`:
`evtData3`:
`evtData4`:

A successful open device returns a non-zero device handle that can be used to access other device related APIs.

Prototype

```
M1x66_HANDLE M1x66_OpenDevice (char *devName, void *callback);
```

Parameters

Data Type	Name	Description
`char *`	`devName`	FXO device descriptor name (default: /dev/ter10).
`void`	`(*eventCallback)( M1x66_EVENT_ID eventId, unsigned int chanId, unsigned int evtData1, unsigned int evtData2, unsigned int evtData3, unsigned int evtData4)`	Event status callback function. The parameters returned by this callback function are event ID, channel ID and 4 event data that are specific to the event ID. Most of these event data are not applicable except those listed in the "EventCallback Parameters" table below.

EventCallback Parameters

Event Name	Parameters
`M1x66_EVENT_RING_START`	evtData1 – Ring burst frequency (in Hz)
`M1x66_EVENT_RING_END`	evtData1 – Ring burst frequency (in Hz) evtData2 – Ring burst duration (in milliseconds)
`M1x66_EVENT_LINE_STATE`	evtData1 – Channel Hook state (0-on-hook, 1-off-hook) evtData2 – 0-line current, 1- voltage evtData3 – IET index evtData4 – Application defined event
`M1x66_EVENT_GPIO_INTERRUPT`	evtData1 – GPIO number

Return Values

Data Type	Description
M1x66_HANDLE	Non-ZERO value if the device is opened successfully. ZERO if the device fails to open.

2.3.2 M1x66_CloseDevice

Description

This API performs the low level closing of the device handle. It should be called before exiting application program to release all resource associated with the open device. Closing this device will automatically force the closing of any opened channel handles.

Prototype

```
M1x66_RET M1x66_CloseDevice (M1x66_HANDLE devHandle);
```

Parameters

Data Type	Name	Description
M1x66_HANDLE	devHandle	Device handle obtained from open device.

Return Values

Data Type	Description
M1x66_RET	M1x66_SUCCESS – Successfully close the device handle. M1x66_FAILED – Failed to close the handle.

2.3.3 M1x66_OpenChannel

Description

This API performs the low level opening of the channel descriptor. By default, the channel descriptor is known as "/dev/terXX", where XX varies from 11 to 26 representing channel 0 to 15 respectively. Calling to the open channel API is mandatory before any access to the channel can be performed, and it must be called after the open device as this API requires the device handle to be passed in as the second parameter. A successful open channel returns a non-zero channel handle that can be used to access other channel related APIs.

Prototype

```
M1x66_RET M1x66_OpenChannel (
      char *pChanName,
      M1x66_HANDLE devHandle,
      int chanNum);
```

Parameters

Data Type	Name	Description
char *	pChanName	FXO channel descriptor name. (Default: /dev/ter11...26).
M1x66_HANDLE	devHandle	Device handle obtained from FXO open device.
int	chanNum	FXO channel number from 0 to 15.

Return Values

Data Type	Description
M1x66_HANDLE	0 – Failed to open channel. None-zero handle – Successful.

2.3.4 M1x66_CloseChannel

Description

This API performs the low level closing of the channel handle. It should be called before exiting application program, or when the channel is no longer needed, to release all resources associated with the open channel.

Prototype

```
M1x66_RET M1x66_CloseChannel (M1x66_HANDLE chanHandle);
```

Parameters

Data Type	Name	Description
M1x66_HANDLE	chanHandle	Channel handle obtained from FXO open channel.

Return Values

Data Type	Description
M1x66_RET	M1x66_SUCCESS – Successfully close the channel. M1x66_FAILED – Failed to close the channel.

2.3.5 M1x66_InitChannel

Description

This API performs the channel initialization and prepares the FXO channel to operate in conformance with specific region where the FXO line is being deployed. Teridian FXO device supports global-compliance parameter in countries listed in `M1x66_COUNTRY_CODE`.

Prototype

```
M1x66_RET M1x66_InitChannel (
      M1x66_HANDLE chanHandle,
      M1x66_COUNTRY_CODE cntryCode );
```

Parameters

Data Type	Name	Description
`M1x66_HANDLE`	`chanHandle`	Channel handle obtained from FXO open channel.
`M1x66_COUNTRY_CODE`	`cntryCode`	Country code as listed in M1x66_COUNTRY_CODE.

Return Values

Data Type	Description
`M1x66_RET`	M1x66_SUCCESS – Successfully initialize the channel. M1x66_FAILED – Failed to initialize the channel.

2.4 PCM Management and Control APIs

2.4.1 M1x66_PcmIfConfig

Description

This API is called to program the PCM interface configuration. It is anticipated that this PCM interface configuration should never change, thus, only need to program once during initialization for each channel.

Prototype

```
M1x66_RET M1x66_PcmIfConfig (
      M1x66_HANDLE channel,
      M1x66_PCM_IF_CFG *pIfConfig );
```

Parameters

Data Type	Name	Description
M1x66_HANDLE	channel	Channel handle from FXO open channel.
M1x66_PCM_IF_CFG	*pIfConfig	Pointer to the PCM interface config data structure.

Return Values

Data Type	Description
M1x66_RET	M1x66_SUCCESS – Successfully configured the PCM interface. M1x66_FAILED – Failure.

2.4.2 M1x66_PcmConfig

Description

This API is called to program the PCM configuration. The PCM configuration contains the actual PCM timeslots, thus, may change from session to session depend on the peer termination. The PCM transmit and receive timeslots are used to establish connection to another device on the PCM bus.

Note: this PCM config API only sets up the PCM connection for both direction (tx and rx) but it does not enable it until the M1x66_PcmActivation is called.

Prototype

```
M1x66_RET M1x66_PcmConfig (
      M1x66_HANDLE channel,
      M1x66_PCM_CFG *pConfig );
```

Parameters

Data Type	Name	Description
M1x66_HANDLE	channel	Channel handle from FXO open channel.
M1x66_PCM_CFG	*pConfig	Pointer to PCM config data structure.

Return Values

Data Type	Description
M1x66_RET	M1x66_SUCCESS – Successfully program the PCM config. M1x66_FAILED – Failure.

2.4.3 M1x66_PcmActivation

Description

This API activates the PCM timeslot configured for this channel. It enables the PCM data stream for both directions.

Prototype

```
M1x66_RET M1x66_PcmActivation (
      M1x66_HANDLE channel );
```

Parameters

Data Type	Name	Description
M1x66_HANDLE	channel	Channel handle from FXO open channel.

Return Values

Data Type	Description
M1x66_RET	M1x66_SUCCESS – Successfully activated the PCM timeslots. M1x66_FAILED – Failure.

2.4.4 M1x66_PcmDeActivation

Description

This API de-activates the PCM timeslot configured for this channel.

Prototype

```
M1x66_RET M1x66_PcmActivation (
      M1x66_HANDLE channel );
```

Parameters

Data Type	Name	Description
M1x66_HANDLE	channel	Channel handle from FXO open channel.

Return Values

Data Type	Description
M1x66_RET	M1x66_SUCCESS – Successfully de-activated the PCM timeslots. M1x66_FAILED – Failure.

2.5 Hook Switch and Pulse Dial Related APIs

2.5.1 M1x66_HookSwitch

Description

Perform on/off-hook on the FXO channel.

Prototype

```
M1x66_RET M1x66_HookSwitch (
      M1x66_HANDLE channel,
      M1x66_HOOK_SWITCH hookSwitch );
```

Parameters

Data Type	Name	Description
M1x66_HANDLE	channel	Channel handle from FXO open channel.
M1x66_HOOK_SWITCH	hookSwitch	Type of hook switch requested.

Return Values

Data Type	Description
M1x66_RET	M1x66_SUCCESS – Successfully performed hook switch. M1x66_FAILED – Failure.

2.5.2 M1x66_PulseDial

Description

Perform pulse dialing on the FXO channel. This API will bring the FXO channel off-hook then perform pulse dialing with the digit string provided. Once the dialing is completed the FXO channel will remain off-hook indefinitely, or until it is manually brought back on-hook.

Up to 30 numeric digits is supported. Valid numeric digit is from ASCII 0 to 9 (i.e., 0x30 to 0x39).

Prototype

```
M1x66_RET M1x66_PulseDial (
      M1x66_HANDLE channel,
      char *dialSting );
```

Parameters

Data Type	Name	Description
M1x66_HANDLE	channel	Channel handle from FXO open channel.
char	*dialString	Numeric dial string of up to 30 characters max.

Return Values

Data Type	Description
M1x66_RET	M1x66_SUCCESS – Successfully dialed. M1x66_FAILED – Failure.

2.5.3 M1x66_PulseDialCancel

Description

2.6 This API terminates any active pulse dial session previously initiated by the Hook Switch and Pulse Dial Related APIs

2.6.1 M1x66_HookSwitch

Description

Perform on/off-hook on the FXO channel.

Prototype

```
M1x66_RET M1x66_HookSwitch (
      M1x66_HANDLE channel,
      M1x66_HOOK_SWITCH hookSwitch );
```

Parameters

Data Type	Name	Description
M1x66_HANDLE	channel	Channel handle from FXO open channel.
M1x66_HOOK_SWITCH	hookSwitch	Type of hook switch requested.

Return Values

Data Type	Description
M1x66_RET	M1x66_SUCCESS – Successfully performed hook switch. M1x66_FAILED – Failure.

M1x66_PulseDial ioctl. The actual cancelation of the pulse dial session occurs in the background and when complete the application should receive a pulse dial aborted event.

Prototype

```
M1x66_RET M1x66_PulseDialCancel (M1x66_HANDLE chanHandle);
```

Parameters

Data Type	Name	Description
M1x66_HANDLE	chanHandle	Channel handle from FXO open channel.

Return Values

Data Type	Description
M1x66_RET	M1x66_SUCCESS – Command successful. M1x66_FAILED – Failure.

2.6.2 M1x66_SetPulseDialParam

Description

The pulse dial API employs three timing parameters to perform dialing. Upon initialization these timers are set to their default values, however, they can be changed to adapt to the line condition using this API: These parameters are as follow:

1. On-hook duration (default 40 ms)
2. Off-hook duration and (default 60 ms)
3. Inter-digits duration (default 1 sec)

Prototype

```
M1x66_RET M1x66_SetPulseDialParam (M1x66_HANDLE chanHandle,
                                   int onHookDuration,
                                   int offHookDuration,
                                   int interDigitDuration);
```

Parameters

Data Type	Name	Description
M1x66_HANDLE	chanHandle	Channel handle from FXO open channel.
int	onHookDuration	On-hook duration.
int	offHookDuration	Off-hook duration.
int	interDigitDuration	Inter-digits duration.

Return Values

Data Type	Description
M1x66_RET	M1x66_SUCCESS – Successful. M1x66_FAILED – Failure.

2.6.3 M1x66_GetPulseDialParam

Description

This API reads the pulse dial timing parameters. If reading is successful the pulse dial timing parameters will be returned in those three passed in parameters.

Prototype

```
M1x66_RET M1x66_GetPulseDialParam (M1x66_HANDLE chanHandle,
                                   int *pOnHookDuration,
                                   int *pOffHookDuration,
                                   int *pInterDigitDuration)
```

Parameters

Data Type	Name	Description
M1x66_HANDLE	chanHandle	Channel handle from FXO open channel.
M1966_GPIO_NUMBER	gpio	GPIO number
int	*pOnHookDuration	Pointer to on-hook duration variable (output)
int	*pOffHookDuration	Pointer to off-hook duration variable (output)
int	*pInterDigitDuration	Pointer to inter-digit duration variable (output)

Return Values

Data Type	Description
M1x66_RET	M1x66_SUCCESS – Successful. M1x66_FAILED – Failure.

2.7 GPIO Management APIs

2.7.1 M1x66_GetGpioConfig

Description

This is an API for reading the configuration of a GPIO. If reading is successful the driver returns the direction and the polarity of the GPIO parameter.

Prototype

```
M1x66_RET M1x66_GetGpioConfig (M1x66_HANDLE chanHandle,
                               M1966_GPIO_NUMBER gpio,
                               M1966_GPIO_SIGNAL_DIR *direction,
                               M1966_GPIO_INTR_POLARITY *polarity)
```

Parameters

Data Type	Name	Description
M1x66_HANDLE	chanHandle	Channel handle from FXO open channel.
M1966_GPIO_NUMBER	gpio	GPIO number
M1966_GPIO_SIGNAL_DIR	*direction	Points to direction variable (output)
M1966_GPIO_INTR_POLARITY	*polarity	Points to polarity variable (output)

Return Values

Data Type	Description
M1x66_RET	M1x66_SUCCESS – Successful. M1x66_FAILED – Failure.

2.7.2 M1x66_SetGpioConfig

Description

This API configures the selected GPIO per “direction” and “polarity” selected. Configuring the GPIO does not activate it; the GPIO must be “enabled” using M1x66_GpioEnable.

When configured as input (direction), the GPIO signal transition can cause interrupt which in turn sends an M1966_EVENT_GPIO_INTERRUPT event to the upper layer application. This depends on the interrupt detection polarity. For example if the polarity is set for detection on the rising edge (M1966_GPIO_POL_RISING), then interrupt will occur when the GPIO data pin signal transitions from low to high, and vice-versa.

Prototype

```
M1x66_RET M1x66_SetGpioConfig (M1x66_HANDLE chanHandle,
                               M1966_GPIO_NUMBER gpio,
                               M1966_GPIO_SIGNAL_DIR direction,
                               M1966_GPIO_INTR_POLARITY polarity)
```

Parameters

Data Type	Name	Description
M1x66_HANDLE	chanHandle	Channel handle from FXO open channel.
M1966_GPIO_NUMBER	gpio	GPIO number.
M1966_GPIO_SIGNAL_DIR	direction	GPIO signal direction.
M1966_GPIO_INTR_POLARITY	polarity	GPIO interrupt edge transition polarity.

Return Values

Data Type	Description
M1x66_RET	M1x66_SUCCESS – Successful. M1x66_FAILED – Failure.

2.7.3 M1x66_SetGpioData

Description

This API writes data to a GPIO pin. This applies only to a GPIO configured with output direction. The GPIO data signal (data) will be written to the selected GPIO pin.

Prototype

```
M1x66_RET M1x66_SetGpioData   (M1x66_HANDLE chanHandle,
                               M1966_GPIO_NUMBER gpio,
                               M1966_GPIO_DATA_TYPE data)
```

Parameters

Data Type	Name	Description
M1x66_HANDLE	chanHandle	Channel handle from FXO open channel.
M1966_GPIO_NUMBER	gpio	GPIO number
M1966_GPIO_DATA_TYPE	data	GPIO data signal to be written

Return Values

Data Type	Description
M1x66_RET	M1x66_SUCCESS – Successful. M1x66_FAILED – Failure.

2.7.4 M1x66_GetGpioData

Description

This is an API for reading data from a GPIO pin. This applies only to a GPIO configured with input direction. The selected GPIO pin signal level will be read into the variable pointed to by the parameter *data.

Prototype

```
M1x66_RET M1x66_GetGpioData   (M1x66_HANDLE chanHandle,
                               M1966_GPIO_NUMBER gpio,
                               M1966_GPIO_DATA_TYPE *data)
```

Parameters

Data Type	Name	Description
M1x66_HANDLE	chanHandle	Channel handle from FXO open channel.
M1966_GPIO_NUMBER	gpio	GPIO number
M1966_GPIO_DATA_TYPE	*data	Pointer to data signal read from the GPIO

Return Values

Data Type	Description
M1x66_RET	M1x66_SUCCESS – Successful. M1x66_FAILED – Failure.

2.7.5 M1x66_GpioEnable

Description

This is an API for enable the GPIO. After a GPIO pin is configured using M1x66_SetGpioConfig it must be enabled to start the operation. Once enabled the GPIO signal can be manipulated using M1x66_SetGpioData for those configured for output direction, or the signal can be read for those configured for input direction.

Prototype

```
M1x66_RET M1x66_GpioEnable    (M1x66_HANDLE chanHandle,
                               M1966_GPIO_NUMBER gpio)
```

Parameters

Data Type	Name	Description
M1x66_HANDLE	chanHandle	Channel handle from FXO open channel.
M1966_GPIO_NUMBER	gpio	GPIO number

Return Values

Data Type	Description
M1x66_RET	M1x66_SUCCESS – Successful. M1x66_FAILED – Failure.

2.7.6 M1x66_GpioDisable

Description

This API disables the GPIO. The GPIO will stop it internal operation,

Prototype

```
M1x66_RET M1x66_GpioEnable    (M1x66_HANDLE chanHandle,
                               M1966_GPIO_NUMBER gpio)
```

Parameters

Data Type	Name	Description
M1x66_HANDLE	chanHandle	Channel handle from FXO open channel.
M1966_GPIO_NUMBER	gpio	GPIO number

Return Values

Data Type	Description
M1x66_RET	M1x66_SUCCESS – Successful. M1x66_FAILED – Failure.

2.8 Loopback Control APIs

2.8.1 M1x66_LoopbackGet

Description

This API reads the current status of the loopback. The active loopback mode will be returned to the variable pointed by the passing parameter.

Prototype

```
M1x66_RET M1x66_LoopbackGet (M1x66_HANDLE chanHandle,
                             M1966_LOOPBACK_MODE *pLoopbackType)
```

Parameters

Data Type	Name	Description
M1x66_HANDLE	chanHandle	Channel handle from FXO open channel.
M1966_LOOPBACK_MODE	*pLoopbackType	Pointer to the loopback mode variable (output)

Return Values

Data Type	Description
M1x66_RET	M1x66_SUCCESS – Successful. M1x66_FAILED – Failure.

2.8.2 M1x66_LoopbackSet

Description

This API sets the transmit and receive data path into a loopback mode. The desired loopback mode will be set on the transmit and receive data path.

Prototype

```
M1x66_RET M1x66_LoopbackSet (M1x66_HANDLE chanHandle,
                             M1966_LOOPBACK_MODE loopbackType)
```

Parameters

Data Type	Name	Description
M1x66_HANDLE	chanHandle	Channel handle from FXO open channel.
M1966_LOOPBACK_MODE	loopbackType	Loopback mode

Return Values

Data Type	Description
M1x66_RET	M1x66_SUCCESS – Successful. M1x66_FAILED – Failure.

2.8.3 M1x66_LoopbackClear

Description

This API disables the loopback.

Prototype

```
M1x66_RET M1x66_LoopbackClear (M1x66_HANDLE chanHandle)
```

Parameters

Data Type	Name	Description
M1x66_HANDLE	chanHandle	Channel handle from FXO open channel.

Return Values

Data Type	Description
M1x66_RET	M1x66_SUCCESS – Successful. M1x66_FAILED – Failure.

2.9 Country Default Parameter and Override APIs

2.9.1 M1x66_GetCountryParam

Description

This API reads the default country parameter. To retrieve the country parameter set the desired country code in the field “cnum”. If the reading is successful the desired country parameter will be returned in the country structure passed in.

Prototype

```
M1x66_RET M1x66_GetCountryParam (M1x66_HANDLE chanHandle,
                                 M1966_CNTRY_STRUCT_t *pParam)
```

Parameters

Data Type	Name	Description
M1x66_HANDLE	chanHandle	Channel handle from FXO open channel.
M1966_CNTRY_STRUCT_t	*pParam	Pointer to structure returning the country parameter.

Return Values

Data Type	Description
M1x66_RET	M1x66_SUCCESS – Successful. M1x66_FAILED – Failure.

2.9.2 M1x66_SetCountryParam

Description

The country parameters are set to their default value upon startup. This default country parameter can be found in ref [1]. This API can be used to change the default parameter.

Prototype

```
M1x66_RET M1x66_SetCountryParam (M1x66_HANDLE chanHandle,
                                 M1966_CNTRY_STRUCT_t *pParam)
```

Parameters

Data Type	Name	Description
M1x66_HANDLE	chanHandle	Channel handle from FXO open channel.
M1966_CNTRY_STRUCT_t	*pParam	Pointer to structure containing the new default country parameters.

Return Values

Data Type	Description
M1x66_RET	M1x66_SUCCESS – Successful. M1x66_FAILED – Failure.

2.9.3 M1x66_ThresholdOverride

Description

Various FXO channel parameters are conveniently grouped and predefined in the default country parameter. These parameters are programmed during channel initialization API (`M1x66_InitChannel`). However, these parameters can be overridden at runtime to fine tune to the desired threshold for the specific installation using this threshold override API.

Note: This API must be invoked after the `M1x66_InitChannel` API to prevent the parameter from over written.

Prototype

```
M1x66_RET M1x66_ThresholdOverride (M1x66_HANDLE chanHandle,
                                   M1966_THRESH_OVERRIDE_t *threshold)
```

Parameters

Data Type	Name	Description
`M1x66_HANDLE`	`chanHandle`	Channel handle from FXO open channel.
`M1966_THRESH_OVERRIDE`	`*threshold`	Pointer to the structure contains the new threshold parameters to be written.

Return Values

Data Type	Description
`M1x66_RET`	M1x66_SUCCESS – Successful. M1x66_FAILED – Failure.

2.10 Billing Tone Filter Control APIs

2.10.1 M1x66_BillingToneFilterEnable

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 API provides the ability to filter out the billing tone.

Prototype

```
M1x66_RET M1x66_BillingToneFilterEnable (M1x66_HANDLE chanHandle,
                                         M1966_BTONE_FREQUENCY frequency)
```

Parameters

Data Type	Name	Description
M1x66_HANDLE	chanHandle	Channel handle from FXO open channel.
M1966_BTONE_FREQUENCY	frequency	Frequency – 12 or 16 KHz.

Return Values

Data Type	Description
M1x66_RET	M1x66_SUCCESS – Successful. M1x66_FAILED – Failure.

2.10.2 M1x66_BillingToneFilterDisable

Description

This API disables the billing tone filter.

Prototype

```
M1x66_RET M1x66_BillingToneFilterDisable (M1x66_HANDLE chanHandle)
```

Parameters

Data Type	Name	Description
M1x66_HANDLE	chanHandle	Channel handle from FXO open channel.

Return Values

Data Type	Description
M1x66_RET	M1x66_SUCCESS – Successful. M1x66_FAILED – Failure.

2.11 1x66 H/W Register Access and Debugging aid APIs

2.11.1 M1x66_HWRegisterRead

Description

Perform H/W register reading from the 73M1x66B device that carries the FXO channel. This API is not recommended for use in normal operation but only for diagnostic or testing purpose.

The channel handle contains the corresponding SPI channel ID (cid) of the 73M1x66B device in the daisy chain. This SPI cid is passed to the driver to retrieve the value of the specified register.

Prototype

```
M1x66_RET M1x66_HWRegisterRead (
      M1x66_HANDLE channel,
      M1x66_REG register,
      char *pRetData );
```

Parameters

Data Type	Name	Description
M1x66_HANDLE	channel	Channel handle from FXO open channel.
M1x66_REG_TYPE	Register	1x66 Register
char	*pRetData	Pointer to a single byte of returned data.

Return Values

Data Type	Description
M1x66_RET	M1x66_SUCCESS – Successfully read the register. M1x66_FAILED – Failure.

2.11.2 M1x66_HWRegisterReadAll

Description

This API is similar to the 1x66 H/W Register Access and Debugging aid APIs M1x66_HWRegisterRead, but all 73M1x66B registers will be read. The register contents will be returned via the pRetData. It is the responsibility of the caller application to insure that this data pointer contains enough space for the return data. A 0x25 bytes buffer is required to store all returned register where the first byte corresponds to the first register (RG00) and so on.

Prototype

```
M1x66_RET M1x66_HWRegisterReadAll (
      M1x66_HANDLE channel,
      char *pRetData );
```

Parameters

Data Type	Name	Description
M1x66_HANDLE	channel	Channel handle from FXO open channel.
char	*pRetData	Pointer to return data for all 0x25 (byte) registers.

Return Values

Data Type	Description
M1x66_RET	M1x66_SUCCESS – Successfully read all registers.

	M1x66_FAILED – Failure.

2.11.3 M1x66_HWRegisterWrite

Description

Perform H/W register writing to the 73M1x66B device that carries the FXO channel. This API is not recommended for use in normal operation but only for diagnostic or testing purpose.

The channel handle contains the corresponding SPI channel ID (cid) of the 73M1x66B device in the daisy chain. This SPI cid is passed to the driver to set the value of the specified register.

Prototype

```
M1x66_RET M1x66_HWRegisterWrite (
      M1x66_HANDLE channel,
      char value );
```

Parameters

Data Type	Name	Description
M1x66_HANDLE	channel	Channel handle from FXO open channel.
M1x66_REG_TYPE	Register	1x66 Register.
char	value	Data value to be written.

Return Values

Data Type	Description
M1x66_RET	M1x66_SUCCESS – Successfully written the register. M1x66_FAILED – Failure.

2.11.4 M1x66_SetDebugTrace

Description

The M1966 Reference Driver provides run-time debugging facility via tracing. The application program can turn on various trace masks to display debug message for trouble shooting purposes. Due to the potential amount of messages resulting from setting this trace mask at kernel level the use of this facility is not recommended under normal operation.

Note: the trace mask occupies a unique bit field, therefore, multiple masks can be “or” together.

Prototype

```
M1x66_RET M1x66_SetDebugTrace (
      M1x66_HANDLE deviceHandle,
      M1x66_DEBUG_TRACE_MASK debugTraceMask );
```

Parameters

Data Type	Name	Description
M1x66_HANDLE	deviceHandle	Device handle from open device.
M1x66_DEBUG_TRACE_MASK	debugTraceMask	Debug trace mask.

Return Values

Data Type	Description
M1x66_RET	M1x66_SUCCESS – Successfully set the trace mask. M1x66_FAILED – Failure.

2.11.5 M1x66_GetFileDrescriptor

Description

To provide full flexibility to the driver access, this API can be called by an application program to obtain the file descriptor for a given device or channel handle. With this returned file descriptor, the user has direct access to the driver via its native IOCTL as described in the *73M1866B/73M1966B Reference Driver User Guide*.

Caution: Accessing directly to the driver via IOCTL in mix mode with API is generally not recommended. However, if this becomes necessary it is strongly advised that the user carefully review the API implementation and be aware of any potential conflicts that may exist as a result of this change.

Prototype

```
int M1x66_GetFileDescriptor (M1x66_HANDLE pHandle);
```

Parameters

Data Type	Name	Description
M1x66_HANDLE	pHandle	Device or channel handle obtains from open API.

Return Values

Data Type	Description
int	Device or channel file descriptor. A NULL value indicates invalid, or not opened file descriptor.

2.11.6 M1x66_GetHWRevision

Description

This API reads the device hardware revision number. If the reading is successful, the device revision number will be stored in the variable pointed to by the pHWRevision parameter.

Prototype

```
M1x66_RET M1x66_GetHWRevision (M1x66_HANDLE chanHandle,
                               unsigned int *pHWRevision)
```

Parameters

Data Type	Name	Description
M1x66_HANDLE	chanHandle	Channel handle from FXO open channel.
unsigned int	*pHWRevision	Pointer to hardware revision number.

Return Values

Data Type	Description
M1x66_RET	M1x66_SUCCESS – Successful. M1x66_FAILED – Failure.

2.12 Interval Event Table Management APIs

2.12.1 M1x66_GetCurrentIET

Description

Read the line current IET table entry.

Prototype

```
int M1x66_GetCurrentIET (
      M1x66_HANDLE channel,
      unsigned int ietIndex,
      M1966_IET_t  *pIET );
```

Parameters

Data Type	Name	Description
M1x66_HANDLE	channel	Channel handle obtained from FXO open channel.
unsigned int	ietIndex	IET entry index – row number (0 – 9).
M1966_IET_t	*pIET	Pointer to an output structure M1966_IET_t.

Return Values

Data Type	Description
M1x66_RET	M1x66_SUCCESS – Successfully read the IET entry. M1x66_FAILED - Failure.

2.12.2 M1x66_UpdateCurrentIET

Description

Perform the line current IET table update.

Prototype

```
int M1x66_UpdateCurrentIET (
      M1x66_HANDLE channel,
      unsigned int ietIndex,
      M1966_IET_t  *pIET );
```

Parameters

Data Type	Name	Description
M1x66_HANDLE	channel	Channel handle obtained from FXO open channel.
unsigned int	ietIndex	IET entry index – row number (0 – 9).
M1966_IET_t	*pIET	Pointer to an input structure M1966_IET_t.

Return Values

Data Type	Description
M1x66_RET	M1x66_SUCCESS – Successfully update the IET entry. M1x66_FAILED – Failure.

2.12.3 M1x66_ClearCurrentIET

Description

Delete the line current IET table entry.

Prototype

```
int M1x66_ClearCurrentIET (
      M1x66_HANDLE channel,
      unsigned int ietIndex);
```

Parameters

Data Type	Name	Description
M1x66_HANDLE	channel	Channel handle obtained from FXO open channel.
unsigned int	ietIndex	IET entry index – row number (0 – 9).

Return Values

Data Type	Description
M1x66_RET	M1x66_SUCCESS – Successfully clear the IET entry. M1x66_FAILED – Failure.

2.12.4 M1x66_GetVoltageIET

Description

Read the line voltage IET table entry.

Prototype

```
int M1x66_GetVoltageIET (
      M1x66_HANDLE channel,
      unsigned int ietIndex,
      M1966_IET_t  *pIET );
```

Parameters

Data Type	Name	Description
M1x66_HANDLE	channel	Channel handle obtained from FXO open channel.
unsigned int	ietIndex	IET entry index – row number (0 – 9).
M1966_IET_t	*pIET	Pointer to an output structure M1966_IET_t.

Return Values

Data Type	Description
M1x66_RET	M1x66_SUCCESS – Successfully read the IET entry. M1x66_FAILED – Failure.

2.12.5 M1x66_UpdateVoltageIET

Description

Perform the line voltage IET table update.

Prototype

```
int M1x66_UpdateVoltageIET (
      M1x66_HANDLE channel,
      unsigned int ietIndex,
      M1966_IET_t  *pIET );
```

Parameters

Data Type	Name	Description
M1x66_HANDLE	channel	Channel handle obtained from FXO open channel.
unsigned int	ietIndex	IET entry index – row number (0 – 9).
M1966_IET_t	*pIET	Pointer to an input structure M1966_IET_t.

Return Values

Data Type	Description
M1x66_RET	M1x66_SUCCESS – Successfully update the IET entry. M1x66_FAILED – Failure.

2.12.6 M1x66_ClearVoltageIET

Description

Delete the line voltage IET table entry.

Prototype

```
int M1x66_ClearVoltageIET (
      M1x66_HANDLE channel,
      unsigned int ietIndex);
```

Parameters

Data Type	Name	Description
M1x66_HANDLE	channel	Channel handle obtained from FXO open channel.
unsigned int	ietIndex	IET entry index – row number (0 – 9).

Return Values

Data Type	Description
M1x66_RET	M1x66_SUCCESS – Successfully clear the IET entry. M1x66_FAILED – Failure.

2.13 Current/Voltage Measurement APIs

2.13.1 M1x66_StartMeasureCurrent

Description

Start the line current measurement process. This API starts measuring the line current of the specified channel at a sampling interval time of “sampleTime” with averaging count of “averageSampleCount”.

Prototype

```
int M1x66_StartMeasureCurrent (
      M1x66_HANDLE channel,
      unsigned int sampleTime,
      unsigned int averageSampleCount );
```

Parameters

Data Type	Name	Description
M1x66_HANDLE	channel	Channel handle obtained from FXO open channel.
unsigned int	sampleTime	Time interval between two consecutive samples.
unsigned int	averageSampleCount	Sample count for average calculation.

Return Values

Data Type	Description
M1x66_RET	M1x66_SUCCESS – Successfully started. M1x66_FAILED – Failure.

2.13.2 M1x66_StartMeasureVoltage

Description

Start the line voltage measurement process.

Prototype

```
int M1x66_StartMeasureVoltage (
      M1x66_HANDLE channel,
      unsigned int sampleTime,
      unsigned int averageSampleCount );
```

Parameters

Data Type	Name	Description
M1x66_HANDLE	channel	Channel handle obtained from FXO open channel.
unsigned int	sampleTime	Time interval between two consecutive samples.
unsigned int	averageSampleCount	Sample count for average calculation.

Return Values

Data Type	Description
M1x66_RET	M1x66_SUCCESS – Successfully started. M1x66_FAILED – Failure.

2.13.3 M1x66_StopMeasureCurrent

Description

Stop the line current measurement.

Prototype

```
int M1x66_StopMeasureCurrent (
      M1x66_HANDLE channel,
      unsigned int sampleTime,
      unsigned int averageSampleCount );
```

Parameters

Data Type	Name	Description
M1x66_HANDLE	Channel	Channel handle obtained from FXO open channel.
unsigned int	sampleTime	Time interval between two consecutive samples.
unsigned int	averageSampleCount	Sample count for average calculation.

Return Values

Data Type	Description
M1x66_RET	M1x66_SUCCESS – Successfully stopped. M1x66_FAILED – Failure.

2.13.4 M1x66_StopMeasureVoltage

Description

Stop the line voltage measurement.

Prototype

```
int M1x66_StopMeasureVoltage (
      M1x66_HANDLE channel,
      unsigned int sampleTime,
      unsigned int averageSampleCount );
```

Parameters

Data Type	Name	Description
M1x66_HANDLE	Channel	Channel handle obtained from FXO open channel.
unsigned int	sampleTime	Time interval between two consecutive samples.
unsigned int	averageSampleCount	Sample count for average calculation.

Return Values

Data Type	Description
M1x66_RET	M1x66_SUCCESS – Successfully stopped. M1x66_FAILED – Failure.

2.14 Miscellaneous Channel Setting APIs

2.14.1 M1x66_SetRingCadence

Description

Set the ring cadence parameter for ring qualification process. The driver can be programmed to filter and qualify incoming ring by discarding spurious signals that can be caused by line noises or distortions. When set with these cadence parameters the driver will evaluate each incoming ring cycle based on these parameters and emits an `M1x66_EVENT_QUALIFIED_RING` event only when the ring cadence matches at the specified number of cycles.

The ring cadence parameter consists of the frequency, cycle duration, period-1 “on”, period-1 “off”, period-2 “on”, period-2 “off”, and number of cycles. This last parameter (number of cycle) is the number of ring cycles matched before the ring is qualified. The sum of the period-1 “on”, period-1 “off”, period-2 “on” and period-2 “off” must be equal to the cycle duration parameter.

Remark

The system is defaulted to ring cadence screening disabled upon startup. Enabling or disabling the filter using this API will not affect the behavior of the ring detection events (`M1x66_EVENT_RING_START` and `M1x66_EVENT_RING_END`). The application may chose to ignore these two events if the qualified ring event is preferred. To turn off this filter the same API can be used with the frequency or cycle duration parameter equal ZERO.

Prototype

```
int M1x66_SetRingCadence (
      M1x66_HANDLE channel,
      unsigned int frequency,
      unsigned int cycleDuration,
      unsigned int periodOneOn,
      unsigned int periodOneOff,
      unsigned int periodTwoOn,
      unsigned int periodTwoOff,
      unsigned int numOfCycles );
```

Parameters

Data Type	Name	Description
`M1x66_HANDLE`	`Channel`	Channel handle obtained from FXO open channel.
`unsigned int`	`frequency`	Ring frequency to be detected.
`unsigned int`	`cycleDuration`	Duration of the cadence cycle (in milliseconds).
`unsigned int`	`periodOneOn`	Period-1 “On” duration (in milliseconds).
`Unsigned int`	`periodOneOff`	Period-1 “Off” duration (in milliseconds).
`Unsigned int`	`periodTwoOn`	Period-2 “On” duration (in milliseconds).
`Unsigned int`	`periodTwoOff`	Period-2 “Off” duration (in milliseconds).
`Unsigned int`	`numOfCycles`	Detect at the end of this cycle number (1 – 5).

Return Values

Data Type	Description
`M1x66_RET`	M1x66_SUCCESS – Successful. M1x66_FAILED – Failure.

2.14.2 M1x66_SetPhoneVolume

Description

Set the speaker phone and microphone volume. This API sets transmit and receive dB gain of the FXO channel. Teridian FXO supports transmit gain from -26 to +12 dB, and receive gain from -24 to +10 dB.

Prototype

```
M1x66_RET M1x66_SetPhoneVolume (
      M1x66_HANDLE chanHandle,
      int txVolumedB,
      int rxVolumedB );
```

Parameters

Data Type	Name	Description
M1x66_HANDLE	chanHandle	Channel handle obtains from open channel.
int	txVolumedB	Transmit direction in dB (-26 to +12 dB).
int	rxVolumedB	Receive direction in dB (-24 to +10 dB).

Return Values

Data Type	Description
M1x66_RET	M1x66_SUCCESS – Successfully set phone volume. M1x66_FAILED – Failure.

2.14.3 M1x66_CallProgressMonitor

Description

For the purpose of monitoring activities on the line, a Call Progress Monitor is provided in the 73M1x66B. This audio output contains both transmit and receive data with configurable levels.

Prototype

```
M1x66_RET M1x66_CallProgressMonitor (
      M1x66_HANDLE      chanHandle,
      M1x66_CM_VOLTAGE_REF voltRef,
      M1x66_CM_GAIN     txGain,
      M1x66_CM_GAIN     rxGain);
```

Parameters

Data Type	Name	Description
M1x66_HANDLE	chanHandle	Channel handle obtains from open channel.
M1x66_CM_VOLTAGE_REF	voltRef	Voltage Reference selection.
M1x66_CM_GAIN	txGain	Transmit path gain setting.
M1x66_CM_GAIN	rxGain	Receive path gain setting.

Return Values

Data Type	Description
M1x66_RET	M1x66_SUCCESS – Successful. M1x66_FAILED – Failure.

2.14.4 M1x66_SelectSampleRate

Description

Select PCM sample rate. The 73M1x66B device can operate in one of two sample rates – 8 kHz or 16 kHz sample rate. The device defaults to 8 kHz operation upon startup and can be changed to run at 16 kHz using this API.

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

1. Data presented in PCM timeslots – reallocation of transmit and receive timeslots may be necessary or required to properly communicate with peer interface.

2. Barrier interface operation – a momentary loss 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.

Prototype

```
M1x66_RET M1x66_SelectSampleRate (
      M1x66_HANDLE chanHandle,
      M1966_SAMPLE_RATE_SELECTION rate);
```

Parameters

Data Type	Name	Description
M1x66_HANDLE	chanHandle	Channel handle obtains from open channel.
M1x66_SAMPLE_RATE_SELECTION	rate	Desired sample rate.

Return Values

Data Type	Description
M1x66_RET	M1x66_SUCCESS – Successful. M1x66_FAILED – Failure.

3 Structure Reference

This section contains the detail description of the data structure used in this API subsystem. Table 2 contains the summary of the reference structure.

Table 2: Structure Overview

Name	Description
M1x66_HANDLE_t	Device or Channel handle.
M1x66_CHAN_INIT_t	Channel Init structure.
M1x66_PCM_IF_CFG_t	PCM interface config structure.
M1x66_PCM_CFG_t	PCM timeslot config structure.
M1966_CNTRY_STRUCT_t	Defines the country default parameters.
M1966_THRESH_OVERRIDE_t	Overrides the threshold parameter of the FXO line.

3.1 M1x66_HANDLE_t

Description

The device or channel handle is a pointer to a device or channel control data structure where information pertaining to device or channel is being kept. This control data structure is created when the device or channel is opened and is used by the API internal subsystem to indentify the device or channel until it is closed.

Prototype

```
/*
*************************************************************************
** Device/Channel Control block
*************************************************************************/
typedef struct {
    int     chanNum;
    int     fd;
    void    (*eventCallback)(M1x66_EVENT_ID,
                             unsigned int,
                             unsigned int,
                             unsigned int,
                             unsigned int,
                             unsigned int);
    int     initialized;
}
m1966_DevChanHandle;

/* Device or channel handle */
typedef m1966_DevChanHandle *M1x66_HANDLE_t;
```

Parameters

Data Type	Name	Description
int	chanNum	Device or channel number.
int	fd	Device or channel corresponding file descriptor.
void	(*eventCallback)()	An event callback function pointer, used only for device.
int	Initialized	Initialization flag.

3.2 M1x66_CHAN_INIT_t

Description

Channel initialization structure. This structure contains mainly the country code. The `M1x66_InitChannel` API passes the pointer to this data structure to configure the channel to operate in a specific region. Refer to `M1x66_COUNTRY_CODE` for the list of countries supported.

Prototype

```
/*
*****************************************************************************
** Channel Init Structure
*****************************************************************************/
typedef struct {
    unsigned char      nMode;
    M1x66_COUNTRY_CODE nCountry;
    void               *pProc;
}
m1966_ChanInit;
typedef m1966_ChanInit M1x66_CHAN_INIT_t;
```

Parameters

Data Type	Name	Description
`char`	`nMode`	N/A
`M1x66_COUNTRY_CODE`	`nCountry`	Defined country codes.
`void *`	`pProc`	N/A

3.3 M1x66_PCM_IF_CFG_t

Description

PCM interface config structure. This structure contains the PCM interface configuration parameter. These parameters are usually static and do not change from session to session. The `M1x66_PcmIfConfig` API passes the pointer to this structure to configure the PCM interface.

Prototype

```
/*
*****************************************************************************
** PCM Interface Config Structure
*****************************************************************************/
typedef struct {
    int     pcmOperMode;
    int     pcmFrequency;
    int     pcmRxEdgePolarity;
    int     pcmTxEdgePolarity;
    int     pcmRxClockSlot;
    int     pcmTxClockSlot;
}
m1966_PcmIfConfig;
typedef m1966_PcmIfConfig M1x66_PCM_IF_CFG_t;
```

Parameters

Data Type	Name	Description
`int`	`pcmOperMode`	Select operation mode: master or slave mode. 1 – Slave mode. 2 – Master mode.
`M1x66_DCL_FREQUENCY`	`pcmFrequency`	DCL Frequency to be used in master or slave mode.
`int`	`pcmRxEdgePolarity`	Edge polarity to be considered for the PCM receive direction. 0 – Falling edge. 1 – Rising edge.
`int`	`pcmTxEdgePolarity`	Edge polarity to be considered for the PCM transmit direction. 0 – Falling edge. 1 – Rising edge.
`int`	`pcmRxClockSlot`	Receive clock slot bit offset.
`int`	`pcmTxClockSlot`	Transmit clock slot bit offset.

3.4 M1x66_PCM_CFG_t

Description

PCM config structure. This structure contains the PCM timeslots and codec parameter. These parameters are not static and may change from session to session. The `M1x66_PcmConfig` API passes the pointer to this structure to configure the PCM timeslots and codec.

Prototype

```
/*
*****************************************************************************
** PCM Config Structure
*****************************************************************************/
typedef struct {
    int     pcmResolution;
    int     pcmTimeslotRX;
    int     pcmTimeslotTX;
}
m1966_PcmConfig;
typedef m1966_PcmConfig M1x66_PCM_CFG_t;
```

Parameters

Data Type	Name	Description
`Int`	`pcmResolution`	Defines the PCM interface coding. 0 – 8-bit A-law. 1 – 8-bit µ-law. 2 – 16-bit linear.
`Int`	`pcmTimeslotRX`	PCM timeslot for receive direction.
`Int`	`pcmTimeslotTX`	PCM timeslot for transmit direction.

3.5 M1966_CNTRY_STRUCT_t

Description

This structure defines the country default parameters.

Prototype

```
typedef struct m1966_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 */
} M1966_CNTRY_STRUCT_t;
```

Parameters

Data Type	Name	Transmit
unsigned int	cnum	Country code (see M1x66_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.

3.6 M1966_THRESH_OVERRIDE_t

Description

This structure can be used to override the threshold parameter of the FXO line.

Prototype

```
typedef struct
{
    unsigned char acz;          /* Active Termination Loop */
    unsigned char dciv;         /* DC current voltage charac. control */
    unsigned char rgth;         /* Ring threshold
}
M1966_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

4 Enumerator Reference

This section contains the detail description of the enumerated data reference used in this API subsystem. Table 3 contains the summary of the enumerated data reference.

Table 3: Enumerator Overview

Name	Description
M1x66_RET	Return status code from API.
M1x66_EVENT_ID	FXO event identification.
M1x66_COUNTRY_CODE	List of country code supported.
M1x66_DCL_FREQUENCY	DCL frequency supported.
M1x66_HOOK_SWITCH	On and off hook.
M1x66_REG_TYPE	List of 1x66 device H/W register.
M1x66_REG_ACCESS_TYPE	Register access type.
M1966_GPIO_NUMBER	List of GPIO pin definition.
M1966_GPIO_CONFIG_COMMAND	GPIO configuration command.
M1966_GPIO_CONTROL_TYPE	GPIO control type definition.
M1966_GPIO_DATA_COMMAND	GPIO data access command.
M1966_GPIO_DATA_TYPE	GPIO data types.
M1966_GPIO_SIGNAL_DIRECTION	Designate the GPIO pin as either input or output.
M1966_GPIO_INTR_POLARITY	GPIO Interrupt signal transition edge selection.
M1x66_DEBUG_TRACE_MASK	Defines all debug trace masks.

4.1 M1x66_RET

Description

API function return code. All APIs, expect the open device and open channel, return with `M1x66_RET` code. For additional error information when an API returns `M1x66_FAILED` the application program can issue a get last error – `M1x66_GetLastError`.

Prototype

```
/*
*****************************************************************************
** Function return code
*****************************************************************************/
typedef enum {
    M1x66_SUCCESS  = 0,
    M1x66_FAILED   = (-1),
}
M1x66_RET;
```

Parameters

Name	Value	Description
M1x66_SUCCESS	0	Successful.
M1x66_FAILED	-1	Failed.

4.2 M1x66_EVENT_ID

Description

List of FXO event identification supported by the API subsystem.

Prototype

```
/*
******************************************************************************
** Event ID
******************************************************************************/
typedef enum {
    M1x66_EVENT_NOPOH         = M1966_NOPOH_DETECT,
    M1x66_EVENT_APOH          = M1966_APOH_DETECT,
    M1x66_EVENT_POLARITY_REV  = M1966_POLARITY_CHG,
    M1x66_EVENT_BATT_DROP     = M1966_BATTERY_DROPPED,
    M1x66_EVENT_BATT_FEED     = M1966_BATTERY_FEEDED,
    M1x66_EVENT_RING_END      = M1966_RING_DETECT_END,
    M1x66_EVENT_RING_START    = M1966_RING_DETECT,
    M1x66_EVENT_SYNC_LOST     = M1966_SYNC_LOSS_DETECT,
    M1x66_EVENT_SPI_ERROR     = M1966_SPI_ERROR_DETECT,
    M1x66_EVENT_ON_HOOK       = M1966_ON_HOOK_DETECT,
    M1x66_EVENT_OVER_VOLTAGE  = M1966_OV_DETECT,
    M1x66_EVENT_OVER_CURRENT  = M1966_OI_DETECT,
    M1x66_EVENT_OVER_LOAD     = M1966_OL_DETECT,
    M1x66_EVENT_UNDER_VOLTAGE = M1966_UV_DETECT,
    M1x66_EVENT_LINE_STATE    = M1966_LINE_STATE,
    M1x66_EVENT_QUALIFIED_RING= M1x66_QUALIFIED_RING,
    M1x66_EVENT_DIAL_COMPLETE = M1966_DIAL_COMPLETE,
    M1x66_EVENT_DIAL_ABORTED  = M1966_DIAL_ABORTED,
    M1x66_EVENT_SYNC_RECOVERED= M1966_SYNC_RECOVERED,
    M1x66_EVENT_GPIO_INTERRUPT= M1966_GPIO_INTERRUPT
}
M1x66_EVENT_ID;
```

Parameters

Name	Value	Description
M1x66_EVENT_GPIO_INTERRUPT	0x00800000	GPIO interrupt event.
M1x66_EVENT_NOPOH	0x00400000	No other phone off-hook.
M1x66_EVENT_APOH	0x00200000	Another phone off-hook.
M1x66_EVENT_POLARITY_REV	0x00100000	FXO line polarity changed.
M1x66_EVENT_BATT_DROP	0x00080000	Battery – FXO line is not feeded.
M1x66_EVENT_BATT_FEED	0x00040000	Battery – FXO line is feeded.
M1x66_EVENT_RING_END	0x00020000	FXO line stopped ringing.
M1x66_EVENT_RING_START	0x00010000	FXO line starts ringing.
M1x66_EVENT_SYNC_LOST	0x00008000	FXO device lost synchronization.
M1x66_EVENT_SPI_ERROR	0x00004000	SPI device error detected.
M1x66_EVENT_ON_HOOK	0x00002000	FXO line goes on-hook.
M1x66_EVENT_OVER_VOLTAGE	0x00001000	Over voltage condition detected.
M1x66_EVENT_OVER_CURRENT	0x00000800	Over current condition detected.
M1x66_EVENT_OVER_LOAD	0x00000400	Over load condition detected.
M1x66_EVENT_UNDER_VOLTAGE	0x00000200	Under voltage condition detected.

M1x66_EVENT_LINE_STATE	0x00000100	Line state event – current/voltage.
M1x66_EVENT_QUALIFIED_RING	0x00000080	Qualified ring cadence detected.

4.3 M1x66_COUNTRY_CODE

Description

List of country codes supported by the API subsystem.

Prototype

```
/*
**************************************************************************
**  Country code List - Internet Country Codes
**************************************************************************/
typedef enum {
    M1x66_CNTRY_CODE_AR          = 0,      /* "Argentina"    */
    M1x66_CNTRY_CODE_AU          = 1,      /* "Australia"    */
    M1x66_CNTRY_CODE_AT          = 2,      /* "Austria"      */
    M1x66_CNTRY_CODE_BH          = 3,      /* "Bahrain"      */
    M1x66_CNTRY_CODE_BE          = 4,      /* "Belgium"      */
    M1x66_CNTRY_CODE_BR          = 5,      /* "Brazil"       */
    M1x66_CNTRY_CODE_BG          = 6,      /* "Bulgaria"     */
    M1x66_CNTRY_CODE_CA          = 7,      /* "Canada"       */
    M1x66_CNTRY_CODE_CL          = 8,      /* "Chile"        */
    M1x66_CNTRY_CODE_C1          = 9,      /* "ChineData"    */
    M1x66_CNTRY_CODE_C2          = 10,     /* "ChinaVoice"   */
    M1x66_CNTRY_CODE_CO          = 11,     /* "Columbia"     */
    M1x66_CNTRY_CODE_HR          = 12,     /* "Croatia"      */
    M1x66_CNTRY_CODE_TB          = 13,     /* "TBR 21"       */
    M1x66_CNTRY_CODE_CY          = 14,     /* "Cyprus"       */
    M1x66_CNTRY_CODE_CZ          = 15,     /* "Czech Rep"    */
    M1x66_CNTRY_CODE_DK          = 16,     /* "Denmark"      */
    M1x66_CNTRY_CODE_EC          = 17,     /* "Equador"      */
    M1x66_CNTRY_CODE_EG          = 18,     /* "Egypt"        */
    M1x66_CNTRY_CODE_SV          = 19,     /* "El Salvador"  */
    M1x66_CNTRY_CODE_FI          = 20,     /* "Finland"      */
    M1x66_CNTRY_CODE_FR          = 21,     /* "France"       */
    M1x66_CNTRY_CODE_DE          = 22,     /* "Germany"      */
    M1x66_CNTRY_CODE_GR          = 23,     /* "Greece"       */
    M1x66_CNTRY_CODE_GU          = 24,     /* "Guam"         */
    M1x66_CNTRY_CODE_HK          = 25,     /* "Hong Kong"    */
    M1x66_CNTRY_CODE_HU          = 26,     /* "Hungary"      */
    M1x66_CNTRY_CODE_IS          = 27,     /* "Iceland"      */
    M1x66_CNTRY_CODE_IN          = 28,     /* "India"        */
    M1x66_CNTRY_CODE_ID          = 29,     /* "Indonesia"    */
    M1x66_CNTRY_CODE_IE          = 30,     /* "Ireland"      */
    M1x66_CNTRY_CODE_IL          = 31,     /* "Israel"       */
    M1x66_CNTRY_CODE_IT          = 32,     /* "Italy"        */
    M1x66_CNTRY_CODE_JP          = 33,     /* "Japan"        */
    M1x66_CNTRY_CODE_JO          = 34,     /* "Jordan"       */
    M1x66_CNTRY_CODE_KZ          = 35,     /* "Kazakhstan"   */
    M1x66_CNTRY_CODE_KW          = 36,     /* "Kuwait"       */
    M1x66_CNTRY_CODE_LV          = 37,     /* "Latvia"       */
    M1x66_CNTRY_CODE_LB          = 38,     /* "Lebanon"      */
    M1x66_CNTRY_CODE_LU          = 39,     /* "Luxembourg"   */
    M1x66_CNTRY_CODE_MO          = 40,     /* "Macao"        */
    M1x66_CNTRY_CODE_MY          = 41,     /* "Malaysia"     */
    M1x66_CNTRY_CODE_MT          = 42,     /* "Malta"        */
    M1x66_CNTRY_CODE_MX          = 43,     /* "Mexico"       */
    M1x66_CNTRY_CODE_MA          = 44,     /* "Morocco"      */
    M1x66_CNTRY_CODE_NL          = 45,     /* "Netherlands"  */
    M1x66_CNTRY_CODE_NZ          = 46,     /* "New Zealand"  */
```

```
    M1x66_CNTRY_CODE_NG          = 47,     /* "Nigeria"      */
    M1x66_CNTRY_CODE_NO          = 48,     /* "Norway"       */
    M1x66_CNTRY_CODE_OM          = 49,     /* "Oman"         */
    M1x66_CNTRY_CODE_PK          = 50,     /* "Pakistan"     */
    M1x66_CNTRY_CODE_PR          = 51,     /* "Peru"         */
    M1x66_CNTRY_CODE_PH          = 52,     /* "Philippines"  */
    M1x66_CNTRY_CODE_PL          = 53,     /* "Poland"       */
    M1x66_CNTRY_CODE_PT          = 54,     /* "Portugal"     */
    M1x66_CNTRY_CODE_RO          = 55,     /* "Romainia"     */
    M1x66_CNTRY_CODE_RU          = 56,     /* "Russia"       */
    M1x66_CNTRY_CODE_SA          = 57,     /* "Saudi Arabia" */
    M1x66_CNTRY_CODE_SG          = 58,     /* "Singapore"    */
    M1x66_CNTRY_CODE_SK          = 59,     /* "Slovakia"     */
    M1x66_CNTRY_CODE_SI          = 60,     /* "Slovenia"     */
    M1x66_CNTRY_CODE_ZA          = 61,     /* "S. Africa"    */
    M1x66_CNTRY_CODE_KR          = 62,     /* "S. Korea"     */
    M1x66_CNTRY_CODE_ES          = 63,     /* "Spain"        */
    M1x66_CNTRY_CODE_SE          = 64,     /* "Sweden"       */
    M1x66_CNTRY_CODE_CH          = 65,     /* "Switzerland"  */
    M1x66_CNTRY_CODE_SY          = 66,     /* "Syria"        */
    M1x66_CNTRY_CODE_TW          = 67,     /* "Taiwan"       */
    M1x66_CNTRY_CODE_TH          = 68,     /* "Thailand"     */
    M1x66_CNTRY_CODE_AE          = 69,     /* "UAE"          */
    M1x66_CNTRY_CODE_UK          = 70,     /* "UK"           */
    M1x66_CNTRY_CODE_US          = 71,     /* "USA"          */
    M1x66_CNTRY_CODE_YE          = 72,     /* "Yemen"        */
}
M1x66_COUNTRY_CODE;
```

4.4 M1x66_DCL_FREQUENCY

Description

Valid DCL Frequency used in `M1x66_InitChannel` API.

Prototype

```
/*
*****************************************************************************
** DCL Frequency
*****************************************************************************/
typedef enum {
    M1x66_256KHZ   = 0x01,    /* 256 KHz */
    M1x66_512KHZ   = 0x02,    /* 512 KHz */
    M1x66_768KHZ   = 0x03,    /* 768 KHz */
    M1x66_1024KHZ  = 0x04,    /* 1.024 MHz */
    M1x66_1536KHZ  = 0x05,    /* 1.536 MHz */
    M1x66_1544KHZ  = 0x06,    /* 1.544 MHz */
    M1x66_2048KHZ  = 0x07,    /* 2.048 MHz */
    M1x66_3088KHZ  = 0x08,    /* 3.088 MHz */
    M1x66_4096KHZ  = 0x09,    /* 4.096 MHz */
    M1x66_6176KHZ  = 0x0A,    /* 6.176 MHz */
    M1x66_8192KHZ  = 0x0B,    /* 8.192 MHz */
}
M1x66_DCL_FREQUENCY;
```

Parameters

Name	Value	Description
M1x66_256KHZ	0x01	256 KHz
M1x66_512KHZ	0x02	512 KHz
M1x66_768KHZ	0x03	768 KHz
M1x66_1024KHZ	0x04	1024 KHz
M1x66_1536KHZ	0x05	1536 KHz
M1x66_1544KHZ	0x06	1544 KHz
M1x66_2048KHZ	0x07	2048 KHz
M1x66_3088KHZ	0x08	3088 KHz
M1x66_4096KHZ	0x09	4096 KHz
M1x66_6176KHZ	0x0A	6176 KHz
M1x66_8192KHZ	0x0B	8192 KHz

4.5 M1x66_HOOK_SWITCH

Description

Hook switch command for on-hook and off-hook.

Prototype

```
/*
*****************************************************************************
** Hook switch
*****************************************************************************/
typedef enum {
    M1x66_ON_HOOK  = 0,
    M1x66_OFF_HOOK = 1,
}
M1x66_HOOK_SWITCH;
```

Parameters

Name	Value	Description
M1x66_ON_HOOK	0x00	On hook.
M1x66_OFF_HOOK	0x01	Off hook.

4.6 M1x66_REG_TYPE

Description

List of 1x66 device register type.

Note: Register 0x00, 0x01, 0x0A, 0x0B are not available. Reading from or writing to these registers will not have any effect.

Prototype

```
/*
*****************************************************************************
** M1x66 Register Type
*****************************************************************************/
typedef enum {
    M1x66_REG_0x00 = 0x0,
    M1x66_REG_0x01 = 0x1,
    M1x66_REG_0x02 = 0x2,
    M1x66_REG_0x03 = 0x3,
    M1x66_REG_0x04 = 0x4,
    M1x66_REG_0x05 = 0x5,
    M1x66_REG_0x06 = 0x6,
    M1x66_REG_0x07 = 0x7,
    M1x66_REG_0x08 = 0x8,
    M1x66_REG_0x09 = 0x9,
    M1x66_REG_0x0A = 0x0a,
    M1x66_REG_0x0B = 0x0b,
    M1x66_REG_0x0C = 0x0c,
    M1x66_REG_0x0D = 0x0d,
    M1x66_REG_0x0E = 0x0e,
    M1x66_REG_0x0F = 0x0f,
    M1x66_REG_0x10 = 0x10,
    M1x66_REG_0x11 = 0x11,
    M1x66_REG_0x12 = 0x12,
    M1x66_REG_0x13 = 0x13,
    M1x66_REG_0x14 = 0x14,
    M1x66_REG_0x15 = 0x15,
    M1x66_REG_0x16 = 0x16,
    M1x66_REG_0x17 = 0x17,
    M1x66_REG_0x18 = 0x18,
    M1x66_REG_0x19 = 0x19,
    M1x66_REG_0x1A = 0x1a,
    M1x66_REG_0x1B = 0x1b,
    M1x66_REG_0x1C = 0x1c,
    M1x66_REG_0x1D = 0x1d,
    M1x66_REG_0x1E = 0x1e,
    M1x66_REG_0x1F = 0x1f,
    M1x66_REG_0x20 = 0x20,
    M1x66_REG_0x21 = 0x21,
    M1x66_REG_0x22 = 0x22,
    M1x66_REG_0x23 = 0x23,
    M1x66_REG_0x24 = 0x24,
    M1x66_REG_0x25 = 0x25
}
M1x66_REG_TYPE;
```

Parameters

Name	Value	Description
M1x66_REG_0x00	Ox00	Register 0x00
M1x66_REG_0x01	0x01	Register 0x01
M1x66_REG_0x02	0x02	Register 0x02
M1x66_REG_0x03	0x03	Register 0x03
M1x66_REG_0x04	0x04	Register 0x04
M1x66_REG_0x05	0x05	Register 0x05
M1x66_REG_0x06	0x06	Register 0x06
M1x66_REG_0x07	0x07	Register 0x07
M1x66_REG_0x08	0x08	Register 0x08
M1x66_REG_0x09	0x09	Register 0x09
M1x66_REG_0x0A	0x0A	Register 0x0A
M1x66_REG_0x0B	0x0B	Register 0x0B
M1x66_REG_0x0C	0x0C	Register 0x0C
M1x66_REG_0x0D	0x0D	Register 0x0D
M1x66_REG_0x0E	0x0E	Register 0x0E
M1x66_REG_0x0F	0x0F	Register 0x0F
M1x66_REG_0x10	0x10	Register 0x10
M1x66_REG_0x11	0x11	Register 0x11
M1x66_REG_0x12	0x12	Register 0x12
M1x66_REG_0x13	0x13	Register 0x13
M1x66_REG_0x14	0x14	Register 0x14
M1x66_REG_0x15	0x15	Register 0x15
M1x66_REG_0x16	0x16	Register 0x16
M1x66_REG_0x17	0x17	Register 0x17
M1x66_REG_0x18	0x18	Register 0x18
M1x66_REG_0x19	0x19	Register 0x19
M1x66_REG_0x1A	0x1A	Register 0x1A
M1x66_REG_0x1B	0x1B	Register 0x1B
M1x66_REG_0x1C	0x1C	Register 0x1C
M1x66_REG_0x1D	0x1D	Register 0x1D
M1x66_REG_0x1E	0x1E	Register 0x1E
M1x66_REG_0x1F	0x1F	Register 0x1F
M1x66_REG_0x20	0x20	Register 0x20
M1x66_REG_0x21	0x21	Register 0x21
M1x66_REG_0x22	0x22	Register 0x22
M1x66_REG_0x23	0x23	Register 0x23
M1x66_REG_0x24	0x24	Register 0x24
M1x66_REG_0x25	0x25	Register 0x25

4.7 M1x66_REG_ACCESS_TYPE

Description

Defines 73M1x66B register access types.

Prototype

```
/*
**************************************************************************
** M1x66 Register Access Type
**************************************************************************/
typedef enum {
    M1x66_REG_READ     = M1966_REG_READ,
    M1x66_REG_READ_ALL = M1966_REG_READALL,
    M1x66_REG_WRITE    = M1966_REG_WRITE
}
M1x66_REG_ACCESS_TYPE;
```

Parameters

Name	Value	Description
M1x66_REG_READ	0x00	Register access – read.
M1x66_REG_READ_ALL	0x01	Register access – read all.
M1x66_REG_WRITE	0x02	Register access – write.

4.8 M1966_GPIO_NUMBER

Description

This is the list of GPIO pin definition.

Prototype

```
typedef enum
{
        M1966_GPIO_NUM_5 = 0x20,                /* GPIO-5              */
        M1966_GPIO_NUM_6 = 0x40,                /* GPIO-6              */
        M1966_GPIO_NUM_7 = 0x80                 /* GPIO-7              */
}
M1966_GPIO_NUMBER;
```

Parameters

Name	Value	Description
M1966_GPIO_NUM_5	0x20	GPIO5 – pin 25 of 73M1906B 32-pin QFN.
M1966_GPIO_NUM_6	0x40	GPIO6 – pin 32 of 73M1906B 32-pin QFN.
M1966_GPIO_NUM_7	0x80	GPIO7 – pin 1 of 73M1906B 32-pin QFN.

4.9 M1966_GPIO_CONFIG_COMMAND

Description

This is the GPIO configuration command. The GPIO can be configured using the M1966_GPIO_CONFIG_SET command. Its configuration can be read using M1966_GPIO_CONFIG_GET command. The GPIO must be enabled using the M1966_GPIO_CONTROL ioctl for the new configuration to take effect.

Prototype

```
typedef enum
{
        M1966_GPIO_CONFIG_GET = 0,              /* GPIO config GET    */
        M1966_GPIO_CONFIG_SET = 1               /* GPIO config SET    */
}
M1966_GPIO_CONFIG_COMMAND;
```

Parameters

Name	Value	Description
M1966_GPIO_CONFIG_GET	0	Read the GPIO configuration
M1966_GPIO_CONFIG_SET	1	Write the GPIO configuration

4.10 M1966_GPIO_CONTROL_TYPE

Description

GPIO control type definition.

Prototype

```
typedef enum
{
        M1966_GPIO_CONTROL_DISABLE = 0,         /* disable GPIO       */
        M1966_GPIO_CONTROL_ENABLE  = 1          /* enable GPIO        */
}
M1966_GPIO_CONTROL_TYPE;
```

Parameters

Name	Value	Description
M1966_GPIO_CONTROL_DISABLE	0	Disable GPIO.
M1966_GPIO_CONTROL_ENABLE	1	Enable GPIO.

4.11 M1966_GPIO_DATA_COMMAND

Description

GPIO data access command. If the M1966_GPIO_SIGNAL_DIRECTION is set to M1966_GPIO_DIR_INPUT, perform the M1966_GPIO_DATA_GET returns the logical value of type M1966_GPIO_DATA_TYPE of the appropriate GPIO as an input. If the M1966_GPIO_SIGNAL_DIRECTION is set to M1966_GPIO_DIR_OUTPUT, the corresponding GPIO port outputs the logical value as written.

Prototype

```
typedef enum
{
        M1966_GPIO_DATA_GET   = 0,              /* Read GPIO data     */
        M1966_GPIO_DATA_SET   = 1               /* Write GPIO data    */
}
M1966_GPIO_DATA_COMMAND;
```

Parameters

Name	Value	Description
M1966_GPIO_DATA_GET	0	Read GPIO data
M1966_GPIO_DATA_SET	1	Write GPIO data

4.12 M1966_GPIO_DATA_TYPE

Description

GPIO data types – this is the GPIO data returned from the M1966_GPIO_DATA_GET access command, or data to be written to the GPIO port using the M1966_GPIO_DATA_SET access command.

Prototype

```
typedef enum
{
        M1966_GPIO_DATA_LOW   = 0,              /* GPIO data - low    */
        M1966_GPIO_DATA_HIGH  = 1               /* GPIO data - high   */
}
M1966_GPIO_DATA_TYPE;
```

Parameters

Name	Value	Description
M1966_GPIO_DATA_LOW	0	GPIO data - low
M1966_GPIO_DATA_HIGH	1	GPIO data - high

4.13 M1966_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 {
    M1966_GPIO_DIR_INPUT  = 0,      /* GPIO pin signal direction – INPUT  */
    M1966_GPIO_DIR_OUTPUT = 1       /* GPIO pin signal direction - OUTPUT */
}
M1966_GPIO_SIGNAL_DIRECTION;
```

Parameters

Name	Value	Description
M1966_GPIO_DIR_INPUT	0	GPIO pin signal direction – INPUT
M1966_GPIO_DIR_OUTPUT	1	GPIO pin signal direction - OUTPUT

4.14 M1966_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 M1966_GPIO_POL_RISING a rising edge will trigger an interrupt from the corresponding GPIO pin. If configured as M1966_GPIO_POL_FALLING a falling edge will trigger an interrupt from the corresponding GPIO pin.

Prototype

```
typedef enum {
    M1966_GPIO_POL_RISING  = 0,  /* Sig transition edge polarity - RISING  */
    M1966_GPIO_POL_FALLING = 1   /* Sig transition edge polarity - FALLING */
}
M1966_GPIO_INTR_POLARITY;
```

Parameters

Name	Value	Description
M1966_GPIO_POL_RISING	0	Interrupt edge selection - RISING
M1966_GPIO_POL_FALLING	1	Interrupt edge selection - FALLING

4.15 M1x66_DEBUG_TRACE_MASK

Description

Defines all debug trace masks.

Prototype

```
/*
*****************************************************************************
** Debug Trace Mask
*****************************************************************************/
typedef enum {
    M1x66_TRACE_MASK_EVENT      = M1966_DEBUG_EVENT,
    M1x66_TRACE_MASK_INIT       = M1966_DEBUG_INIT,
    M1x66_TRACE_MASK_PATH       = M1966_DEBUG_RING_PATH,
    M1x66_TRACE_MASK_TRACE      = M1966_DEBUG_TRACE,
    M1x66_TRACE_MASK_CNTRY_CODE = M1966_DEBUG_COUNTRY_CODE,
    M1x66_TRACE_MASK_UNUSED_A   = M1966_DEBUG_UNUSED_A,
    M1x66_TRACE_MASK_LINE_STATE = M1966_DEBUG_LINE_STATE,
    M1x66_TRACE_MASK_IOCTL      = M1966_DEBUG_IOCTL,
    M1x66_TRACE_MASK_PCM        = M1966_DEBUG_PCM,
    M1x66_TRACE_MASK_BARRIER    = M1966_DEBUG_BARRIER,
    M1x66_TRACE_MASK_INT        = M1966_DEBUG_INT,
    M1x66_TRACE_MASK_PHU        = M1966_DEBUG_PHU,
    M1x66_TRACE_MASK_TAPI       = M1966_DEBUG_TAPI,
    M1x66_TRACE_MASK_KPROC      = M1966_DEBUG_KPROC,
    M1x66_TRACE_MASK_SPI        = M1966_DEBUG_SPI,
    M1x66_TRACE_MASK_ERROR      = M1966_DEBUG_ERROR
}
M1x66_DEBUG_TRACE_MASK;
```

Parameters

Name	Value	Description
M1x66_TRACE_MASK_EVENT	0x00000001	Event trace mask.
M1x66_TRACE_MASK_INIT	0x00000002	Initialization trace mask.
M1x66_TRACE_MASK_RING_PATH	0x00000004	Ring path analysis trace mask.
M1x66_TRACE_MASK_TRACE	0x00000008	Function level trace mask.
M1x66_TRACE_MASK_CNTRY_CODE	0x00000010	Country code processing trace mask.
M1x66_TRACE_MASK_LINE_STATE	0x00000040	Line state analysis path trace mask.
M1x66_TRACE_MASK_IOCTL	0x00000080	IOCTL trace mask.
M1x66_TRACE_MASK_PCM	0x00000100	PCM trace mask.
M1x66_TRACE_MASK_BARRIER	0x00000200	Barrier related function trace mask.
M1x66_TRACE_MASK_INT	0x00000400	Interrupt trace mask.
M1x66_TRACE_MASK_PHU	0x00000800	Phone hung up in off-hook trace mask.
M1x66_TRACE_MASK_TAPI	0x00001000	TAPI related trace mask.
M1x66_TRACE_MASK_KPROC	0x00002000	Kernel processing trace mask.
M1x66_TRACE_MASK_SPI	0x00004000	SPI related trace mask.
M1x66_TRACE_MASK_ERROR	0x80000000	Error trace mask.

5 Sample Application

This section illustrates how an FXO application can be implemented using the FXOAPIs. The sample program is a typical telephony application that automatically answers the call when a ring burst is detected. This is accomplished by simply going off-hook.

Note: this sample program will definitely not compile and is not a complete application. It is intended only for demonstration purpose to show the capability and to illustrate the interaction between the application program and the FXOAPI.

```
/*************************************************************
**
**   FXO sample application
**
**   Perform the following:
**   - Initialize the API subsystem – M1x66_ApiInit()
**   - Open the FXO device          - M1x66_OpenDevice()
**   - Open each channel            - M1x66_OpenChannel()
**   - Config PCM interface         - M1x66_PcmIfConfig()
**   - Initialize channel           - M1x66_InitChannel()
**
*************************************************************/
int main (int argc, char **argv)
{
      M1x66_PCM_IF_CFG  pcmIfConfig;
      M1x66_CHAN_INIT   chanInit;
      M1x66_HANDLE      pFxoDev;

      /* Initialize the API subsystem */
      if (M1x66_FxoApiInit() != M1x66_SUCCESS)
            return (-1);

      /* Open device descriptor with event callback function */
      pFxoDev = M1x66_OpenDevice("/dev/ter", &fxoEventCallback);
      if (pFxoDev == NULL)
            return (-1);

      /* For each channel descriptor */
      for (i=0; i<MAX_DAA_INSTALLED; i++)
      {
            /* Open the channel descriptor */
            channles[i].pHandle = M1x66_OpenChannel("/dev/ter", pFxoDev, i);
            if (channels[i].pHandle == NULL)
                  return (-1);

            /* configure PCM interface */
            pcmIfConfig.pcmOperMode       = 0;
            pcmIfConfig.pcmFrequency      = M1x66_2048KHZ;
            pcmIfConfig.pcmRxEdgePolarity = 0;
            pcmIfConfig.pcmTxEdgePolarity = 1;
            pcmIfConfig.pcmRxClockSlot    = 0;
            pcmIfConfig.pcmTxClockSlot    = 0;

            /* Now configure the PCM interface */
            ret = M1x66_PcmIfConfig (channels[i].pHandle, &pcmIfConfig);
            if (ret != M1x66_SUCCESS)
                  return (-1);
```

```
            /* Initialize the channel for operation in US */
            if (M1x66_InitChannel (channels[i].pHandle, M1x66_CNTRY_CODE_US)
            != M1x66_SUCCESS)
                  return (-1);
      }

      /* program main loop */

      While (1)
      {
            /* this is the main program loop…
            /* . . . */
      }

      /* Release the API subsystem before exiting */
      M1x66_RET M1x66_FxoApiRelease (void);

      Return (0);
}

/*
*************************************************************************
**********************
* FUNCTION NAME:
*   fxoEventCallback
*
* DESCRIPTION:
*   This is the FXO event callback function. It is invoked by the API
*   subsystem when an FOX event occurs. Currently, this callback function
*   only handles the RING start event. It takes the FXO line off-hook when
*   the ring burst is detected.
*
* PARAMETERS:
*   event_id     - Event ID
*   channelId    - FXO channel where event is occurring
*   data1        - Additional data (1)
*   data2        - Additional data (2)
*   data3        - Additional data (3)
*   data4        - Additional data (4)
*
* RETURNS:
*   N/A
*************************************************************************
**********************
*/
void fxoEventCallback (M1x66_EVENT_ID event_id,
                       unsigned int channelId,
                       unsigned int data1,
                       unsigned int data2,
                       unsigned int data3,
                       unsigned int data4)
{
      int ret;
      M1x66_HANDLE  pHandle;
      M1x66_PCM_CFG pcmConfig;

      pHandle = channels[channelId].pHandle;
```

```
      switch (event_id)
      {
        case M1x66_EVENT_RING_START:
                  printf("\nRING(%d)",channelId);
                  printf("\n     Frequency: %d(hz)",data1);
                  printf("...auto answer...\n");

                  /* configure PCM timeslot */
                  pcmConfig.pcmResolution = 0;  /* use A-law */
                  pcmConfig.pcmTimeslotRX = 7;
                  pcmConfig.pcmTimeslotTX = 6;

                  ret = M1x66_PcmConfig (pHandle, &pcmConfig);
                  if (ret != M1x66_SUCCESS)
                        break;

                  ret = M1x66_PcmActivation (pHandle);
                  if (ret != M1x66_SUCCESS)
                        break;

                  ret = M1x66_HookSwitch (pHandle, M1x66_OFF_HOOK);
                  if (ret == M1x66_FAILED)
                        break;
                  else
                        printf("\nSuccessful");
                  break;

        case M1x66_EVENT_RING_END:
             printf("\nRING_END(%d)\n\r",channelId);
             printf("\n     Frequency: %d(hz), Duration:
%d(ms)\n",data1,data2);
             break;

        case M1x66_EVENT_POLARITY_REV:
             printf("\nPOLARITY_CHANGE(%d)\n\r",channelId);
             break;

        case M1x66_EVENT_BATT_FEED:
             printf("\nBATTERY_FEEDED(%d)\n\r",channelId);
             break;

        case M1x66_EVENT_BATT_DROP:
             printf("\nBATTERY_DROPPED(%d)\n\r",channelId);
             break;

        case M1x66_EVENT_APOH:
             printf("\nAPOH(%d)\n\r",channelId);
             break;

        case M1x66_EVENT_NOPOH:
             printf("\nNOPOH(%d)\n\r",channelId);
             break;

        case M1x66_EVENT_LINE_STATE:
             printf("\n\r   LINE_STATE(%d)",channelId);
             printf("\n     HookState: %s, Entity: %s, Row: %d, Event: %d\n",

M1x66_GetHookStateStr(data1),M1x66_GetEntityStr(data2),data3,data4);
break;
```

```
        case M1x66_EVENT_DIAL_COMPLETE:
             printf("\n\r   DIAL_COMPLETE(%d)\n\r",channelId);
             break;

        case M1x66_EVENT_DIAL_ABORTED:
             printf("\n\r   DIAL_ABORTED(%d)\n\r",channelId);
             break;

        case M1x66_EVENT_SYNC_LOST:
             printf("\n\r   BARRIER_SYNC_LOST(%d)\n\r",channelId);
             break;

        case M1x66_EVENT_SYNC_RECOVERED:
             printf("\n\r   BARRIER_SYNC_RECOVERED(%d)\n\r",channelId);
             break;

        case M1x66_EVENT_GPIO_INTERRUPT:
             printf("\n\r   %s interrupt:(%d)\n\r",GPIO_NUM_TO_STR(data1),channelId);
             break;

        default:
             printf("\nUnkown event\n\r");
             break;
    }
}
```

6 Related Documentation

The following 73M1x66B documents are available from Teridian Semiconductor Corporation:

73M1866B/73M1966B Data Sheet
73M1866B/73M1966B Demo Board User Manual
73M1866B/73M1966B GUI User Guide
73M1866B/73M1966B Reference Driver User Guide
73M1866B/73M1966B Layout Guidelines
73M1x66B Worldwide Design Guide
73M1866B/73M1966B FXOCTL Application User Guide

7 Contact Information

For more information about Teridian Semiconductor products or to check the availability of the 73M1866B and 73M1966B, contact us at:

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

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

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

———

Revision History

Revision	Date	Description
1.0	6/12/2009	First publication.
2.0	10/2/2009	Added Section 2.4.3 through Section 2.8.2. Added Section 2.9.6. Added Section 3.5 and Section 3.6. Added Section 4.8 through Section 4.14.
2.1	11/2/2009	Added support for PCM sample rate selection.