One Technology Way • P.O. Box 9106 • Norwood, MA 02062-9106, U.S.A. • Tel: 781.329.4700 • Fax: 781.461.3113 •
www.analog.com

Linux API for ADM1266 Super Sequencer with Interchip Bus

FEATURES

Linux API for following ADM1266 functions

- **Configuration Loading/Upgrade**
- **Firmware Loading/Upgrade**
- **Open Loop Margining**
- **Closed Loop Margining**
- **Blackbox Readback**
- **Monitoring and Telemetry Readback**

GENERAL DESCRIPTION

The Linux API for ADM1266 are higher level system API. They include functions that automatically scale if multiple ADM1266 are present in the system. Additionally, some functions return system information, instead of the user having to read back individual device information and stitch it all together.

Please read through the license agreement included in the package and agree to it before proceeding with the ADM1266 Library and the associated scripts.

TABLE OF CONTENTS

Linux API for ADM1266 Super Sequencer with Interchip Bus..1

Features....1

General Description....1

Revision History....2

Description of the Modules....3

- Library Module....**Error! Bookmark not defined.**
- PMBus / I2C Module....3
- User Interface Modules....3

Modifications Applicable to ADM1266_PMBUS_Interface.c....4

- I2C Write....4
- I2C Read....**Error! Bookmark not defined.**
- I2C Init....**Error! Bookmark not defined.**

Modifications Applicable to all User Interface Modules....6

Functions for Open Loop Margining....7

- Open Loop Margin....7

Functions for Closed Loop Margining....8

- Closed Loop Margin, All rails....8
- Closed Loop Margin, Single rail by pin name....8
- Closed Loop Margin, Single rail by pin index....8
- Mapping DAC to input pin....9
- Update closed loop margining thesholds, for single input rail....9
- Update closed loop margining thesholds, for all input rails..10

Functions for loading configuraion and firmware....11

- Program Firmware....11
- Program Configuration....11

Monitor/Telemetry Functions....12

- System_Read....12
- System_Parse....12
- VX, PDIO and GPIO status readback....12
- Get System Status....13
- Print System Status....13
- Print Telemetry for entire system....14
- System Lock Status....14
- Device Present....15
- Refresh Status....15
- CRC Summary....15
- Get Refresh Counter....16
- Print Refresh Counter....16
- Get CRC Error Counter....16
- Print CRC Error Counter....17
- Print MFR_ID....17
- Print MFR_MODEL....17
- Print MFR_REVISION....18
- Print MFR_LOCATION....18
- Print MFR_DATE....18
- Print MFR_SERIAL....19
- Print USER_DATA....19
- IC Device ID....19
- IC Device Rev....20
- System CRC....20
- Print CRC....20
- Part Locked Status....21
- Running Memory Status....21
- Get Current State....22
- Print Current State....22
- VX Value and Status....22
- PDIO/GPIO Status....23

Blackbox Functions....24

- Number of Records....24
- Clear Blackbox....24
- Configuration Name....24
- Get Blackbox Record....25
- Blackbox Parse....25

REVISION HISTORY

6/2019—Revision Rev0: Initial Version

DESCRIPTION OF THE MODULES

LIBRARY MODULE

adm1266.c and adm1266.h

These files contain all the ADM1266 related functions which are called by the user. It is not recommended to modify these files.

PMBUS / I2C MODULE

adm1266_pmbus_interface.c and adm1266_pmbus_interface.h

These files contain functions which call the standard Linux I2C commands for various Read and Write operations.

USER INTERFACE MODULES

Blackbox.c, closed_loop_margin.c, open_loop_margin.c, load_fw_config.c and Monitor.c

These files demo the various features which call required functions in the adm1266.c to perform a specific task for example loading firmware and configuration. Minor modifications are required by the user to update these files with their system specific information.

MODIFICATIONS APPLICABLE TO ADM1266_PMBUS_INTERFACE.C

This section covers the I2C write and read functions and the modifications required in adm1266_pmbus_interface.c if i2c master is not accessed though Linux ioctrl system call.

I2C WRITE

```
__u32 i2c_block_write(__u8 device_addr, __u8 dataout_length, __u8 *dataout);
```

Arguments

`device_addr` i2c address of the salve device, example 0x40

`dataout_length` length of the data written to the slave device

`*dataout` pointer to the data bytes written to the slave device

register address is included as part of `*dataout`

Return Value

Number of bytes written to the i2c slave.

Details

This function is used to write data to i2c slave device. The function can be used to do byte write, word write, and block write.

Modifications

If i2c master is not accessed through Linux ioctrl system call, replace the following, with the API used for I2C master.

```
i2c_smbus_block_write_big(file, device_addr, command, length, datawrite)
```

I2C READ

```
__u32 i2c_block_write_block_read(__u8 device_addr, __u8 dataout_length, __u8 *dataout, __u8
read_no_bytes, __u8 *datain);
```

Arguments

`device_addr` i2c address of the salve device, example 0x40

`dataout_length` length of the data written to the slave device

`*dataout` pointer to the data bytes written to the slave device

register address is included as part of `*dataout`

`read_no_bytes` length of the data readback from the slave device

`*datain` the data read from the slave device

Return Value

Number of bytes read from the i2c slave.

Modifications

If i2c master is not accessed through Linux ioctrl system call, replace the following, with the API used to access i2c master.

```
i2c_smbus_block_write_block_read(file, device_addr, command, length, datawrite, read_no_bytes, datain)
```

I2C INIT

```
void i2c_init()
```

Arguments

Return Value

Details

This function is used to initialize the i2c master.

Modifications

If i2c master is not accessed through Linux ioctrl system call. Replace this function with the required function.

MODIFICATIONS APPLICABLE TO ALL USER INTERFACE MODULES

Based on the number of ADM1266 and PMBus addresses some modifications are required in the high-level C code.

Modification 1:

The total number of ADM1266 in a system needs to be updated in the C code.

```
#define ADM1266_NUM 2
```

By default, it's set to 2, change this parameter based on the number of ADM1266. The total number of devices can be any number from 1 to 16.

Modification 2:

The PMBus address of ADM1266 should be updated to match the system.

```
__u8 ADM1266_Address[NO_OF_ADM1266] = {0x40, 0x42};
```

By default, the address of the two ADM1266 is set to 0x40 and 0x42, update these values accordingly. The number of address listed is based on the number of ADM1266 in a system, which can be from 1 to 16.

FUNCTIONS FOR OPEN LOOP MARGINING

OPEN LOOP MARGIN

```
void ADM1266_Margin_Open_Loop(__u8 ADM1266_Address, __u8 ADM1266_DAC_Number, float ADM1266_DAC_Output)
```

Arguments

`ADM1266_Address`	PMBus address of the device, where the margining DAC is located, example 0x40
`ADM1266_DAC_Number`	physical DAC pin number, valid input is from 1 to 9
`ADM1266_DAC_Output`	requested output voltage (V) of the DAC, valid range 0.202 – 1.565

Return Value

None

Details

This is the top-level function for open loop margining. This function calls the low-level functions to set the requested voltage on a DAC.

FUNCTIONS FOR CLOSED LOOP MARGINING

CLOSED LOOP MARGIN, ALL RAILS

```
void ADM1266_Margin_All(__u8 *ADM1266_Address, __u8 ADM1266_NUM, __u8 ADM1266_Margin_Type)
```

Arguments

*ADM1266_Address pointer to the array of all the PMBus addresses of ADM1266 in a system

ADM1266_NUM the total number of ADM1266 in a system, only valid input is the number of elements in ADM1266_Address array

ADM1266_Margin_Type type of margining for all the rails, valid inputs are 0x01 for margin high, 0x02 for margin low, 0x03 for margin Vout, 0x04 for margin disable.

Return Value

None

Details

This function margins all the rails to high, low and Vout that are configured as closed loop margining in a system.

CLOSED LOOP MARGIN, SINGLE RAIL BY PIN NAME

```
void ADM1266_Margin_Single(__u8 ADM1266_Address, char *ADM1266_Pin_Name, __u8 ADM1266_Margin_Type);
```

Arguments

ADM1266_Address PMBus address of the device where the margining rail is located, example 0x40

*ADM1266_Pin_Name the physical input pin name of the rail which is required to be margined, valid inputs are “VH1” – “VH4”, and “VP1” to “VP13”

ADM1266_Margin_Type type of margining for the selected rail, valid inputs are 0x01 for margin high, 0x02 for margin low, 0x03 for margin Vout, 0x04 for margin disable.

Return Value

None

Details

This function margins a single rail on a specific device by passing the name of the input pin, margin type and device address.

CLOSED LOOP MARGIN, SINGLE RAIL BY PIN INDEX

```
void ADM1266_Margin_Single_Input(__u8 ADM1266_Address, __u8 ADM1266_Pin_Index, __u8
ADM1266_Margin_Type);
```

Arguments

ADM1266_Address PMBus address of the device where the margining rail is located, example 0x40

ADM1266_Pin_Index pin number of the rail which is required to be margined, valid inputs are 1 for VH1, 2 for VH2, 3 for VH3, 4 for VH4, 5 for VP1 …….. 17 for VP13

ADM1266_Margin_Type type of margining for the selected rail, valid inputs are 0x01 for margin high, 0x02 for margin low, 0x03 for margin Vout, 0x04 for margin disable.

Return Value

None

Details

This function margins a single rail on a specific device by passing the pin index, margin type and device address. This function is meant to be used in conjunction with dac_mapping function.

MAPPING DAC TO INPUT PIN

```
void ADM1266_DAC_Mapping(__u8 *ADM1266_Address, __u8 ADM1266_NUM, struct ADM1266_dac_data
*ADM1266_DAC_data)
```

Arguments

*ADM1266_Address pointer to the array of all the PMBus addresses of the ADM1266 in a system

ADM1266_NUM the total number of ADM1266 in a system, only valid input is the number of elements in ADM1266_Address array

*ADM1266_DAC_data pointer to structure array which will be filled up with the device address, device index, and input channel that is margined by each DAC

Return Value

None

Details

This function populates the ADM1266_DAC_data structure with for each DAC with device address, device index, and input channel to which the DAC is mapped.

UPDATE CLOSED LOOP MARGINING THESHOLDS, FOR SINGLE INPUT RAIL

```
void ADM1266_Margin_Single_Percent(__u8 ADM1266_Address, __u8 ADM1266_Pin, float
ADM1266_Margin_Percent)
```

Arguments

ADM1266_Address PMBus address of the device, where the input channel is located, example 0x40

ADM1266_Pin the physical pin number of the channel for which the margining value needs to be updated, valid inputs are 1 for VH1, 2 for VH2, 3 for VH3, 4 for VH4, 5 for VP1 …….. 17 for VP13

ADM1266_Margin_Percent the percentage of nominal to which the margin high and low values will be set, valid input is 0.00 – 50

it is not required to pass negative value for margin low or % sign at the end of the value

Return Value

None

Details

This function updates the margin high and low value based on the user entered percentage for a single channel. The function automatically reads back the nominal voltage and calculates the margin high and low value based on the percentage of the nominal value.

UPDATE CLOSED LOOP MARGINING THESHOLDS, FOR ALL INPUT RAILS

```
void ADM1266_Margin_All_Percent(__u8 ADM1266_NUM, struct ADM1266_dac_data *ADM1266_DAC_data, float ADM1266_Margin_Percent)
```

Arguments

`ADM1266_NUM` the total number of ADM1266 in a system, only valid input is the number of elements in `ADM1266_Address` array

`*ADM1266_DAC_data` pointer to structure array which contains the device address, device index, and input channel that is margined by the DAC

this array requires to be filled first by calling ADM1266_DAC_Mapping function

`ADM1266_Margin_Percent` the percentage of nominal to which the margin high and low values will be set, valid input is 0.00 – 50, it is not required to pass negative value for margin low or % sign at the end of the value

Return Value

None

Details

This function updates the margin high and low value based on the user entered percentage for all channels. The function automatically reads back the nominal voltage and calculates the margin high and low value based on the percentage of the nominal value.

FUNCTIONS FOR LOADING CONFIGURAION AND FIRMWARE

PROGRAM FIRMWARE

```
void ADM1266_Program_Firmware(__u8 *ADM1266_Address, __u8 ADM1266_NUM, FILE *ADM1266_Ptr_File)
```

Arguments

`*ADM1266_Address` pointer to the array of all the PMBus addresses of the ADM1266 in a system

`ADM1266_NUM` the total number of ADM1266 in a system, only valid input is the number of elements in `ADM1266_Address` array

`*ADM1266_Ptr_File` pointer to the path of the firmware *.hex file, valid input is path to an ADM1266 firmware hex file

Return Value

None

Details

This function loads the firmware hex file, which is provided, to all the ADM1266 in a system. The file pointer should contain only one path for the firmware, since the same firmware is loaded to all the ADM1266.

PROGRAM CONFIGURATION

```
void ADM1266_Program_Config(__u8 *ADM1266_Address, __u8 ADM1266_NUM, FILE *ADM1266_Ptr_File[], __u8
ADM1266_Reset_Sequence);
```

Arguments

`*ADM1266_Address` pointer to the array of all the PMBus addresses of the ADM1266 in a system

`ADM1266_NUM` the total number of ADM1266 in a system, only valid input is the number of elements in `ADM1266_Address` array

`*ADM1266_Ptr_File` pointer to the path of the configuration *.hex files, valid input is path to an ADM1266 configuration hex files, the file name should be in the same order as the device_addr

`ADM1266_Reset_Sequence` parameter for performing a seamless update, valid input is "1" for seamless update, or "0" for reset sequence after loading configuration file

Return Value

None

Details

This function loads configuration hex files to all the ADM1266 in a system.. Each ADM1266 has its unique configuration file and should be passed as `ADM1266_Ptr_File` array in the same order as PMBus address listed in `ADM1266_Address` array.

MONITOR/TELEMETRY FUNCTIONS

SYSTEM_READ

```
void ADM1266_System_Read(__u8 ADM1266_Num, __u8 *ADM1266_Address, __u8 *ADM1266_System_Data);
```

Arguments

*`ADM1266_Address` pointer to the array of all the PMBus addresses of the ADM1266 in a system

`ADM1266_NUM` the total number of ADM1266 in a system, only valid input is the number of elements in `ADM1266_Address` array

Return Value

This function returns the raw System Data in the *`ADM1266_System_Data` array

Details

This function returns the raw System Data which includes all the Rails and Signals Mapping, Rail Names, Signal Names and State Names. The above function should be called at least once, the information readback does not change during the operation of the ADM1266. The information is fixed for a system, and only needs to be readback again, if a new configuration is loaded into the ADM1266

SYSTEM_PARSE

```
void ADM1266_System_Parse(__u8 *ADM1266_System_Data, __u16 *ADM1266_State_Name, __u16
*ADM1266_Rail_Name, __u16 *ADM1266_Signal_Name, __u8 *ADM1266_VH_Data, __u8 *ADM1266_VP_Data, __u8
*ADM1266_Signals_Data, __u8 *ADM1266_ADM1266_PDIO_GPIO_Pad, __u8 *ADM1266_ADM1266_VX_Pad);
```

Arguments

*`ADM1266_System_Data` pointer to the array of raw System Data

*`ADM1266_ADM1266_PDIO_GPIO_Pad` pointer to the array of GPIO and PDIO index mapping, this is a constant value.

*`ADM1266_ADM1266_VX_Pad` pointer to the array of VX index mapping, this is a constant value.

Return Value

*`ADM1266_State_Name` array of State Names with data pointing to the System_Data

*`ADM1266_Rail_Name` array of Rail Names with data pointing to the System_Data

*`ADM1266_Signal_Name` array of Signal Names with data pointing to the System_Data

*`ADM1266_VH_Data` array of VH and PDIO pins mapping to different rails

*`ADM1266_VP_Data` array of VP and PDIO pins mapping to different rails

*`ADM1266_Signals_Data` array of GPIO and PDIO pins mapping to different Signals

Details

This function parses through the raw System Data and returns the Rails and Signals Mapping, Rail Names, Signal Names and State Names. The above function should be called at least once, the information readback does not change during the operation of the ADM1266. The information is fixed for a system, and only needs to be readback again, if a new configuration is loaded into the ADM1266

VX, PDIO AND GPIO STATUS READBACK

```
void ADM1266_Get_All_Data(__u8 ADM1266_NUM, __u8 *ADM1266_Address, __u8 *ADM1266_VH_Data, __u8
*ADM1266_VP_Data, __u8 *ADM1266_Signals_Data, __u16 *ADM1266_Voltages, __u8 *ADM1266_Status);
```

Arguments

*ADM1266_Address pointer to the array of all the PMBus addresses of the ADM1266 in a system

ADM1266_NUM the total number of ADM1266 in a system, only valid input is the number of elements in ADM1266_Address array

*ADM1266_VH_Data pointer to the array of VH and PDIO pins mapping to different rails

*ADM1266_VP_Data pointer to the array of VP and PDIO pins mapping to different rails

*ADM1266_Signals_Data pointer to the array of GPIO and PDIO pins mapping to different Signals

Return Value

*ADM1266_Voltages array of Rail voltages read back from READ_VOUT

*ADM1266_Status array of Rail status read back from STATUS_VOUT

Details

This function reads back the instantaneous status of all the VX pins, along with their ADC readings and the comparator status. It also reads back the status of all the PDIO and GPIO pins and returns back meaningful information in the above arrays.

GET SYSTEM STATUS

```
__u8 ADM1266_Get_Sys_Status(__u8 ADM1266_NUM, __u8 *ADM1266_Status);
```

Arguments

ADM1266_NUM the total number of ADM1266 in a system, only valid input is the number of elements in ADM1266_Address array

*ADM1266_Status pointer to the array of Rail status read back from STATUS_VOUT

Return Value

5 = OV Fault, 4 = UV Fault, 3 = OV Warning, 2 = UV Warning, 0 = No faults or Warnings

Details

This function goes through the entire ADM1266_Status array, checks the highest value available for the status and returns back the most critical status for the system. You need to call "ADM1266_Get_All_Data" function before calling this function

PRINT SYSTEM STATUS

```
__u8 ADM1266_Print_Sys_Status(__u8 ADM1266_NUM, __u8 *ADM1266_Status);
```

Arguments

ADM1266_NUM the total number of ADM1266 in a system, only valid input is the number of elements in ADM1266_Address array

*ADM1266_Status pointer to the array of Rail status read back from STATUS_VOUT

Return Value

Printf of system status. OV Fault, UV Fault, OV Warning, UV Warning, No faults or Warnings

Details

This function goes through the entire ADM1266_Status array, checks the highest value available for the status and prints the most critical status for the system. You need to call "`ADM1266_Get_All_Data`" function before calling this function

PRINT TELEMETRY FOR ENTIRE SYSTEM

```
void ADM1266_Print_Telemetry(__u8 ADM1266_NUM, __u8 *ADM1266_VH_Data, __u8 *ADM1266_VP_Data, __u8
*ADM1266_Signals_Data, __u16 *ADM1266_Voltages, __u8 *ADM1266_Status, __u16 *ADM1266_Rail_Name, __u16
*ADM1266_Signal_Name, __u8 *ADM1266_System_Data);
```

Arguments

`ADM1266_NUM` the total number of ADM1266 in a system, only valid input is the number of elements in `ADM1266_Address` array

`*ADM1266_VH_Data` pointer to the array of VH and PDIO pins mapping to different rails

`*ADM1266_VP_Data` pointer to the array of VP and PDIO pins mapping to different rails

`*ADM1266_Signals_Data` pointer to the array of GPIO and PDIO pins mapping to different Signals

`*ADM1266_Voltages` pointer to the array of Rail voltages read back from READ_VOUT

`*ADM1266_Status` pointer to the array of Rail status read back from STATUS_VOUT

`*ADM1266_Rail_Name` pointer to the array of Rail Names

`*ADM1266_Signal_Name` pointer to the array of Signal Names

`*ADM1266_System_Data` pointer to the array of raw System Data

Return Value

Printf of all the Rails, along with their instantaneous ADC reading and Status. Also Signals logic level status

Details

This function prints the Voltage and Status of each Rail, it also prints the status of each Signal. It sorts the rail based on following order OV Fault, UV Fault, OV Warning, UV Warning, No Fault or Warning, Disabled. You need to call "`ADM1266_Get_All_Data`" function before calling this function

SYSTEM LOCK STATUS

```
__u8 ADM1266_Get_Part_Locked_System(__u8 ADM1266_NUM, __u8 *ADM1266_Address);
```

Arguments

`ADM1266_NUM` the total number of ADM1266 in a system, only valid input is the number of elements in `ADM1266_Address` array

`*ADM1266_Address` pointer to the array of all the PMBus addresses of the ADM1266 in a system

Return Value

0 if all the ADM1266 in a system are unlocked

1 for any other conditions

Details

This function checks if all the ADM1266 in a system are unlocked and returns 0 if all the ADM1266 are unlocked.

DEVICE PRESENT

```
__u8 ADM1266_Device_Present(__u8 *ADM1266_Address, __u8 ADM1266_NUM)
```

Arguments

`*ADM1266_Address` pointer to the array of all the PMBus addresses of the ADM1266 in a system

`ADM1266_NUM` the total number of ADM1266 in a system, only valid input is the number of elements in `ADM1266_Address` array

Return Value

This function returns 1 if all the devices are present, 0 if all the devices are not present.

Details

This function checks if all the ADM1266 listed in `ADM1266_Address` array are present.

REFRESH STATUS

```
__u8 ADM1266_Refresh_Status(__u8 *ADM1266_Address, __u8 ADM1266_NUM)
```

Arguments

`*ADM1266_Address` pointer to the array of all the PMBus addresses of the ADM1266 in a system

`ADM1266_NUM` the total number of ADM1266 in a system, only valid input is the number of elements in `ADM1266_Address` array

Return Value

This function returns 1 if refresh is running in any of the devices, 0 if refresh is not running.

Details

This function checks if any of the ADM1266 listed in `ADM1266_Address` is running memory refresh.

CRC SUMMARY

```
void ADM1266_CRC_Summary(__u8 *ADM1266_Address, __u8 ADM1266_NUM);
```

Arguments

`*ADM1266_Address` pointer to the array of all the PMBus addresses of the ADM1266 in a system

`ADM1266_NUM` the total number of ADM1266 in a system, only valid input is the number of elements in `ADM1266_Address` array

Return Value

None

Details

This function prints out the CRC status of all the ADM1266 listed in `ADM1266_Address`.

GET REFRESH COUNTER

```
void ADM1266_Get_Refresh_Counter(__u8 ADM1266_NUM, __u8 *ADM1266_Address, __u16
*ADM1266_Refresh_Counter);
```

Arguments

`ADM1266_NUM` the total number of ADM1266 in a system, only valid input is the number of elements in `ADM1266_Address` array

`*ADM1266_Address` pointer to the array of all the PMBus addresses of the ADM1266 in a system

Return Value

`*ADM1266_Refresh_Counter` pointer to the array of refresh counter for each device

`ADM1266_Refresh_Counter[0] -` Refresh Counter Value for first ADM1266, `ADM1266_Refresh_Counter[1] -` Refresh Counter Value for second ADM1266

Details

This function returns the number of times Refresh Feature is run since power-up

PRINT REFRESH COUNTER

```
void ADM1266_Print_Refresh_Counter(__u8 ADM1266_NUM, __u8 *ADM1266_Address, __u16
*ADM1266_Refresh_Counter);
```

Arguments

`ADM1266_NUM` the total number of ADM1266 in a system, only valid input is the number of elements in `ADM1266_Address` array

`*ADM1266_Address` pointer to the array of all the PMBus addresses of the ADM1266 in a system

Return Value

Printf of Refresh Counter values of each device

Details

This function is like the "`ADM1266_Get_Refresh_Counter`" function, but instead of returning a value, it prints the Refresh Counters. You don't need to call the "`ADM1266_Get_Refresh_Counter`" function to run this function

GET CRC ERROR COUNTER

```
void ADM1266_Get_CRC_Error_Counter(__u8 ADM1266_NUM, __u8 *ADM1266_Address, __u16
*ADM1266_CRC_Error_Counter);
```

Arguments

`ADM1266_NUM` the total number of ADM1266 in a system, only valid input is the number of elements in `ADM1266_Address` array

`*ADM1266_Address` pointer to the array of all the PMBus addresses of the ADM1266 in a system

Return Value

`*ADM1266_CRC_Error_Counter` pointer to the array of CRC error counter of each device

`ADM1266_CRC_Error_Counter [0]` – CRC Error Counter Value for first ADM1266, `ADM1266_CRC_Error_Counter [1]` – CRC Error Counter Value for second ADM1266

Details

This function returns the number of times CRC Error has been detected since power-up

PRINT CRC ERROR COUNTER

```
void ADM1266_Print_CRC_Error_Counter(__u8 ADM1266_NUM, __u8 *ADM1266_Address, __u16
*ADM1266_CRC_Error_Counter);
```

Arguments

`*ADM1266_Address` pointer to the array of all the PMBus addresses of the ADM1266 in a system

`ADM1266_NUM` the total number of ADM1266 in a system, only valid input is the number of elements in `ADM1266_Address` array

Return Value

Printf of CRC Error Counter values of each device

Details

This function is like the "`ADM1266_Get_CRC_Error_Counter`" function, but instead of returning a value, it prints the CRC Error Counters. You don't need to call the "`ADM1266_Get_CRC_Error_Counter`" function to run this function

PRINT MFR_ID

```
void ADM1266_Print_MFR_ID(__u8 ADM1266_NUM, __u8 *ADM1266_Address);
```

Arguments

`*ADM1266_Address` pointer to the array of all the PMBus addresses of the ADM1266 in a system

`ADM1266_NUM` the total number of ADM1266 in a system, only valid input is the number of elements in `ADM1266_Address` array

Return Value

Printf of PMBus MFR_ID values of each device

Details

This function prints the value stored in the standard PMBus command called MFR_ID

PRINT MFR_MODEL

```
void ADM1266_Print_MFR_MODEL(__u8 ADM1266_NUM, __u8 *ADM1266_Address);
```

Arguments

`*ADM1266_Address` pointer to the array of all the PMBus addresses of the ADM1266 in a system

`ADM1266_NUM` the total number of ADM1266 in a system, only valid input is the number of elements in `ADM1266_Address` array

Return Value

Printf of PMBus MFR_MODEL values of each device

Details

This function prints the value stored in the standard PMBus command called MFR_MODEL

PRINT MFR_REVISION

```
void ADM1266_Print_MFR_REVISION(__u8 ADM1266_NUM, __u8 *ADM1266_Address);
```

Arguments

*`ADM1266_Address` pointer to the array of all the PMBus addresses of the ADM1266 in a system

`ADM1266_NUM` the total number of ADM1266 in a system, only valid input is the number of elements in `ADM1266_Address` array

Return Value

Printf of PMBus MFR_REVISION values of each device

Details

This function prints the value stored in the standard PMBus command called MFR_REVISION

PRINT MFR_LOCATION

```
void ADM1266_Print_MFR_LOCATION(__u8 ADM1266_NUM, __u8 *ADM1266_Address);
```

Arguments

*`ADM1266_Address` pointer to the array of all the PMBus addresses of the ADM1266 in a system

`ADM1266_NUM` the total number of ADM1266 in a system, only valid input is the number of elements in `ADM1266_Address` array

Return Value

Printf of PMBus MFR_LOCATION values of each device

Details

This function prints the value stored in the standard PMBus command called MFR_LOCATION

PRINT MFR_DATE

```
void ADM1266_Print_MFR_DATE(__u8 ADM1266_NUM, __u8 *ADM1266_Address);
```

Arguments

*`ADM1266_Address` pointer to the array of all the PMBus addresses of the ADM1266 in a system

`ADM1266_NUM` the total number of ADM1266 in a system, only valid input is the number of elements in `ADM1266_Address` array

Return Value

Printf of PMBus MFR_DATE values of each device

Details

This function prints the value stored in the standard PMBus command called MFR_DATE

PRINT MFR_SERIAL

```
void ADM1266_Print_MFR_MODEL(__u8 ADM1266_NUM, __u8 *ADM1266_Address);
```

Arguments

*ADM1266_Address pointer to the array of all the PMBus addresses of the ADM1266 in a system

ADM1266_NUM the total number of ADM1266 in a system, only valid input is the number of elements in ADM1266_Address array

Return Value

Printf of PMBus MFR_SERIAL values of each device

Details

This function prints the value stored in the standard PMBus command called MFR_SERIAL

PRINT USER_DATA

```
void ADM1266_Print_User_Data(__u8 ADM1266_NUM, __u8 *ADM1266_Address);
```

Arguments

*ADM1266_Address pointer to the array of all the PMBus addresses of the ADM1266 in a system

ADM1266_NUM the total number of ADM1266 in a system, only valid input is the number of elements in ADM1266_Address array

Return Value

Printf of values stored in the 0xE3 USER_DATA command values of each device

Details

This function prints the value stored in the 0xE3 USER_DATA command

IC DEVICE ID

```
void ADM1266_Get_IC_Device_ID(__u8 ADM1266_NUM, __u8 *ADM1266_Address, __u8 *ADM1266_IC_Device_ID);
```

Arguments

*ADM1266_Address pointer to the array of all the PMBus addresses of the ADM1266 in a system

ADM1266_NUM the total number of ADM1266 in a system, only valid input is the number of elements in ADM1266_Address array

Return Value

* ADM1266_IC_Device_ID pointer to the array of IC_DEVICE_ID of each device

ADM1266_IC_Device_ID [0][n] – IC_DEVICE_ID for first ADM1266, ADM1266_IC_Device_ID [1][n] – IC_DEVICE_ID for second ADM1266

Right Values - ADM1266_IC_Device_ID[0][0] = 0x42, ADM1266_IC_Device_ID[0][1] = 0x12, ADM1266_IC_Device_ID[0][2] = 0x66

Details

This function returns the IC_DEVICE_ID for each ADM1266. It should read back 0x42, 0x12, 0x66. This can be used to identify if the device is ADM1266 for a particular PMBus address.

IC DEVICE REV

```
void ADM1266_Get_IC_Device_Rev(__u8 ADM1266_NUM, __u8 *ADM1266_Address, __u8 *ADM1266_Firmware_Rev,
__u8 *ADM1266_Bootloader_Rev);
```

Arguments

`*ADM1266_Address` pointer to the array of all the PMBus addresses of the ADM1266 in a system

`ADM1266_NUM` the total number of ADM1266 in a system, only valid input is the number of elements in `ADM1266_Address` array

Return Value

`*ADM1266_Firmware_Rev` pointer to the array of Firmware_Rev of each device

`ADM1266_Firmware_Rev [0][n] –` Firmware_Rev for first ADM1266, `ADM1266_Firmware_Rev [1][n] –` Firmware_Rev for second ADM1266

ADM1266_Firmware_Rev[0][0].ADM1266_Firmware_Rev[0][1].ADM1266_Firmware_Rev[0][2] = 1.14.3

`*ADM1266_Bootloader_Rev` pointer to the array of Bootloader_Rev of each device

`ADM1266_ Bootloader _Rev [0][n] –` Bootloader _Rev for first ADM1266, `ADM1266_ Bootloader _Rev [1][n] –` Bootloader _Rev for second ADM1266

ADM1266_Bootloader_Rev[0][0].ADM1266_Bootloader_Rev[0][1].ADM1266_Bootloader_Rev[0][2] = 0.0.9

Details

This function returns the IC_DEVICE_REV for each ADM1266. It reads back the Firmware and Bootloader versions for each ADM1266

SYSTEM CRC

```
__u8 ADM1266_Get_Sys_CRC(__u8 ADM1266_NUM, __u8 *ADM1266_Address);
```

Arguments

`*ADM1266_Address` pointer to the array of all the PMBus addresses of the ADM1266 in a system

`ADM1266_NUM` the total number of ADM1266 in a system, only valid input is the number of elements in `ADM1266_Address` array

Return Value

1 = CRC Fault, 0 = No CRC fault

Details

This function returns 1 if there is any CRC Fault in any of the ADM1266 devices in the system

PRINT CRC

```
void ADM1266_Print_CRC(__u8 ADM1266_NUM, __u8 *ADM1266_Address);
```

Arguments

*ADM1266_Address pointer to the array of all the PMBus addresses of the ADM1266 in a system

ADM1266_NUM the total number of ADM1266 in a system, only valid input is the number of elements in ADM1266_Address array

Return Value

Printf of all the CRC faults present across all the ADM1266 in the system

Details

This function prints all the CRC faults present across all the ADM1266 in the system

PART LOCKED STATUS

```
__u8 ADM1266_Get_Part_Locked(__u8 ADM1266_NUM, __u8 *ADM1266_Address, __u8 *ADM1266_Part_Locked);
```

Arguments

*ADM1266_Address pointer to the array of all the PMBus addresses of the ADM1266 in a system

ADM1266_NUM the total number of ADM1266 in a system, only valid input is the number of elements in ADM1266_Address array

Return Value

The function also returns a value global value, 1 = At least one part is locked. 0 = All parts are locked

*ADM1266_Part_Locked pointer to the array of Part Locked Status of each device

ADM1266_Part_Locked[0][n] – Part Locked Status for first ADM1266, ADM1266_Part_Locked[1][n] – Part Locked Status for second ADM1266

1 = Locked, 0 = Unlocked

Details

This function is used to read back the part locked status for all the ADM1266 in the system

RUNNING MEMORY STATUS

```
void ADM1266_Get_Main_Backup(__u8 ADM1266_NUM, __u8 *ADM1266_Address, __u8 *ADM1266_Main_Backup);
```

Arguments

*ADM1266_Address pointer to the array of all the PMBus addresses of the ADM1266 in a system

ADM1266_NUM the total number of ADM1266 in a system, only valid input is the number of elements in ADM1266_Address array

Return Value

*ADM1266_Main_Backup pointer to the array of running memory status of each device

ADM1266_Main_Backup[0][n] – running memory status for first ADM1266, ADM1266_Main_Backup[1][n] – running memory status for second ADM1266

1 = Backup, 0 = Main

Details

This function is used to read back if the ADM1266 is running Main or Backup memory for User Configuration data

GET CURRENT STATE

```
void ADM1266_Get_Current_State(__u8 ADM1266_NUM, __u8 *ADM1266_Address, __u8 *ADM1266_Current_State);
```

Arguments

*ADM1266_Address pointer to the array of all the PMBus addresses of the ADM1266 in a system

ADM1266_NUM the total number of ADM1266 in a system, only valid input is the number of elements in ADM1266_Address array

Return Value

*ADM1266_Current_State pointer to the array of current state of each device

ADM1266_Current_State[0][n] – current state of first ADM1266, ADM1266_Current_State[1][n] – current state of second ADM1266

Details

This function is used to read back the current state for each ADM1266 in the system

PRINT CURRENT STATE

```
void ADM1266_Print_Current_State(__u8 ADM1266_NUM, __u8 *ADM1266_Address, __u8 *ADM1266_System_Data,
__u16 *ADM1266_State_Name);
```

Arguments

*ADM1266_Address pointer to the array of all the PMBus addresses of the ADM1266 in a system

ADM1266_NUM the total number of ADM1266 in a system, only valid input is the number of elements in ADM1266_Address array

*ADM1266_State_Name pointer to the array of Signal Names

*ADM1266_System_Data pointer to the array of raw System Data

Return Value

Printf for the current state name for each ADM1266 in the system

Details

This function is used to print the current state for each ADM1266 in the system

VX VALUE AND STATUS

```
void ADM1266_VX_Telemetry(__u8 ADM1266_Dev, __u8 ADM1266_Pin, __u8 *ADM1266_VX_Status, float
*ADM1266_VX_Value, __u8 *ADM1266_VH_Data, __u8 *ADM1266_VP_Data, __u16 *ADM1266_Voltages, __u8
*ADM1266_Status);
```

Arguments

*ADM1266_VH_Data pointer to the array of VH and PDIO pins mapping to different rails

*ADM1266_VP_Data pointer to the array of VP and PDIO pins mapping to different rails

*ADM1266_Voltages pointer to the array of Rail voltages read back from READ_VOUT

*ADM1266_Status pointer to the array of Rail status read back from STATUS_VOUT

ADM1266_Dev ADM1266_Dev = 0 means first ADM1266, 1 means second ADM1266, etc.

`ADM1266_Pin` ADM1266_Pin = 1:4 for VH1:VH4, 5:17 for VP1:VP13

Return Value

`ADM1266_VX_Status` ADM1266_VX_Status = 5 = OV Fault, 4 = UV Fault, 3 = OV Warning, 2 = UV Warning, 1 = Rail Disabled, 0 = No faults or Warnings

`ADM1266_VX_Value` Rail Voltage

Details

This function is used to readback the voltage and status for a specific VX pin for a specific ADM1266 in a system

PDIO/GPIO STATUS

```
__u8 ADM1266_PDIOGPIO_Telemetry(__u8 ADM1266_Dev, __u8 ADM1266_Pin, __u8 *ADM1266_Signals_Data);
```

Arguments

`*ADM1266_Signals_Data` pointer to the array of GPIO and PDIO pins mapping to different signals

`ADM1266_Dev` ADM1266_Dev = 0 means first ADM1266, 1 means second ADM1266, etc.

`ADM1266_Pin` ADM1266_Pin = ADM1266_Pin = 1:16 for PDIO1:PDIO16, 17:25 for GPIO1:GPIO9

Return Value

The function returns this value = 1 High, 0 Low

Details

This function is used to readback the status for a specific PDIO/GPIO pin for a specific ADM1266 in a system

BLACKBOX FUNCTIONS

NUMBER OF RECORDS

```
void ADM1266_Get_Num_Records(__u8 *ADM1266_Address, __u16 *ADM1266_Record_Index, __u16
*ADM1266_Num_Records);
```

Arguments

*ADM1266_Address pointer to the array of all the PMBus addresses of the ADM1266 in a system

Return Value

*ADM1266_Record_Index the record index of the last blackbox value

*ADM1266_Num_Records the number of blackbox records currently present in the ADM1266

Details

This function takes the PMBus address, and returns the Number of Blackbox Records and the Last Blackbox Record Index

CLEAR BLACKBOX

```
void ADM1266_BB_Clear(__u8 ADM1266_Num, __u8 *ADM1266_Address);
```

Arguments

*ADM1266_Address pointer to the array of all the PMBus addresses of the ADM1266 in a system

ADM1266_NUM the total number of ADM1266 in a system, only valid input is the number of elements in ADM1266_Address array

Return Value

None

Details

This function erases and clears the Blackbox records of all the ADM1266 listed in ADM1266_Address.

It is important to call the "ADM1266_System_Read" and "ADM1266_System_Parse" functions before calling the following functions

CONFIGURATION NAME

```
void ADM1266_Configuration_Name(__u8 *ADM1266_System_Data);
```

Arguments

*ADM1266_System_Data pointer to the array of raw System Data

Return Value

Printf of the user configuration name saved in the ADM1266

Details

This function takes in the raw 'ADM1266_System_Data' and prints the name of the User Configuration present in ADM1266

GET BLACKBOX RECORD

```
void ADM1266_Get_BB_Raw_Data(__u8 ADM1266_Num, __u8 *ADM1266_Address, __u8 index, __u16
ADM1266_Record_Index, __u16 ADM1266_Num_Records, __u8 *ADM1266_BB_Data);
```

Arguments

*ADM1266_Address pointer to the array of all the PMBus addresses of the ADM1266 in a system

ADM1266_NUM the total number of ADM1266 in a system, only valid input is the number of elements in ADM1266_Address array

*index the blackbox record number you would like to readback. index = 1 to 32

*ADM1266_Record_Index the record index of the last blackbox value

*ADM1266_Num_Records the number of blackbox records currently present in the ADM1266

Return Value

*ADM1266_BB_Data pointer to the array of raw blackbox data for one blackbox record

Details

This function takes in the Number of ADM1266 and their PMBus address, it also takes in the index of the Blackbox record to readback, along with the Record_Index and Num_Records, that were read back using the function "ADM1266_Get_Num_Records". It reads back the Blackbox Record Raw Data and saves it in the 'ADM1266_BB_Data' array

BLACKBOX PARSE

```
void ADM1266_BB_Parse(__u8 ADM1266_Num, __u8 *ADM1266_BB_Data, __u8 *ADM1266_System_Data, __u16
*ADM1266_State_Name, __u8 *ADM1266_VH_Data, __u8 *ADM1266_VP_Data, __u8 *ADM1266_Signals_Data, __u16
*ADM1266_Rail_Name, __u16 *ADM1266_Signal_Name);
```

Arguments

ADM1266_NUM the total number of ADM1266 in a system, only valid input is the number of elements in ADM1266_Address array

*ADM1266_BB_Data pointer to the array of raw blackbox data for one blackbox record

*ADM1266_System_Data pointer to the array of raw System Data

*ADM1266_State_Name pointer to the array of State Names

*ADM1266_VH_Data pointer to the array of VH and PDIO pins mapping to different rails

*ADM1266_VP_Data pointer to the array of VP and PDIO pins mapping to different rails

*ADM1266_Signals_Data pointer to the array of GPIO and PDIO pins mapping to different Signals

*ADM1266_Rail_Name pointer to the array of Rail Names

*ADM1266_Signal_Name pointer to the array of Signal Names

Return Value

Printf of the blackbox record

Details

This function takes in the raw blackbox data 'ADM1266_BB_Data' Array, and parses it to print the blackbox Record