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

**Description of the Modules**

Library module

ADM1266\_Lib.py

Contains all the ADM1266 related functions which are called by the user interfacing scripts. It is not recommended to modify this library.

PMBus module

ADM1266\_Lib.py

Call APIs which uses the Total Phase Aardvark dongle to communicate with the ADM1266.

If a different i2c dongle is used, the user need to include the APIs for that specific dongle.

User Interfacing Script

ADM1266 Load Firmware and Configuration.py etc.

These are user interfacing scripts which call required functions in the ADM1266\_Lib to perform a specific task for example loading firmware and configuration.

Minor modifications are required by the user to update the script with their system specific information. The required modifications are described in Modification Applicable to All User Interfacing Scripts and Modification Applicable to specific script.

**Modifications Applicable to PMBus\_I2C**

This section covers the modifications required in PMBus\_I2C.py to interface with a different I2C dongle or master. Common modifications in the section below is not applicable for this module.

PMBus\_Write\_Read(device\_address, write\_data, read\_length)

The function is provided with the ADM1266 device address, register address and number of bytes to be read. It returns array of data byte which is read back from the ADM1266.

Inside the PMBus\_Write\_Read function aardvark\_py.aa\_i2c\_write\_read API needs to be updated with a similar API of the new I2C dongle or master.

PMBus\_Write (device\_address, write\_data)

This function is provided with the ADM1266 device address and the data which need to be written to the ADM1266.

Inside the PMBUS\_Write function aardvark\_py.aa\_i2c\_write API needs to be updated with a similar API of the new I2c dongle or master.

Open\_Aardvark(Number)

This function is Total Phase Aardvark specific function can be removed, if a different I2C master is used.

Open\_Aardvark()

This function is Total Phase Aardvark specific function can be removed, if a different I2C master is used.

**Modifications Applicable to All User Interfacing Scripts**

This section covers the modifications that are applicable for all the user interfacing scripts, listed below:

*ADM1266 Load Firmware and Configuration.py*

*ADM1266 Telemetry Read.py*

*ADM1266 Blackbox Read.py*

*ADM1266 Margin - Closed Loop.py*

*ADM1266 Margin - Open Loop.py*

**Step 1:**

Open the respective python script using any text editor.

**Step 2:**

The unique ID, 1845957160, in the following line of the script requires to be updated with the unique ID of the dongle which is used to communicate with the ADM1266.

PMBus\_I2C.Open\_Aardvark(1845957160)

The unique ID of the dongle is printed on the dongle, shown in Figure 1.

If no unique ID is provided, the first un-used Aardvark detected will be used.



Figure 1. Unique ID on the Dongle

**Step 2:**

The address of the ADM1266s which are in the same system requires to be updated in the following line of the script. The addresses need be separated by commas.

ADM1266\_Lib.ADM1266\_Address = [0x40, 0x42]

The example above shows ADM1266s with PMBus address of 0x40 and 0x42.

**Modifications Applicable to Firmware and Configuration Loading Script**

This section covers the specific modifications required only for *ADM1266 Load Firmware and Configuration.py*

**Step 1:**

The firmware file which will be used to program all the ADM1266 is defined in the following line of the script. The same firmware file will be used to program all the ADM1266 in a system. The following example points to the latest firmware hex file, which is included with the script.

ADM1266\_Lib.firmware\_file\_name = "adm1266\_v1.14.2\_b0.hex"

If the firmware file is in the same directory as *ADM1266 Load Firmware and Configuration.py,* it is only required to enter the name of the firmware HEX file, otherwise the full path is required.

**Step 2:**

The configuration file which will be used to program all the ADM1266 in a system is defined in the following line of the script. Each 1266 in a multi-device system will have its own unique configuration file. Each configuration file is separated by a comma.

The following example is for programming two configuration files to two ADM1266, 2 Board [Demo-device@40.hex](mailto:Demo-device@40.hex) is programmed in ADM1266 at address 0x40 and 2 Board [Demo-device@42.hex](mailto:Demo-device@42.hex) is programmed in ADM1266 at address 0x42.

The configuration file names should be listed in the same order as PMBus addresses defined in Step 2 of Modifications Applicable to All Scripts.

ADM1266\_Lib.config\_file\_name = ["2 Board Demo-device@40.hex", "2 Board Demo-device@42.hex"]

If the configuration files are in the same directory as *ADM1266 Load Firmware and Configuration.py*, it is only required to enter the name of the configuration files, otherwise the full path is required.

**Firmware and Configuration Loading Script**

**Functions used from ADM1266\_Lib:**

* ADM1266\_Lib.program\_configration(soft\_reset)

This function loads the configuration files defined in ADM1266\_Lib.config\_file\_name

to all the devices listed in ADM1266\_Lib.ADM1266\_Address. Pass soft\_reset as False to do a seamless reset after loading the configuration, do not pass any value if soft reset/sequence reset is needed after loading the configuration.

* ADM1266\_Lib.program\_configration()

This function loads the firmware file defined in ADM1266\_Lib.firmware\_file\_name

to all the requires listed in ADM1266\_Lib.ADM1266\_Address.

No parameter needs to be passed to this function.

* ADM1266\_Lib.crc\_summary()

This function triggers a CRC recalculate and reads back the CRC status of all the devices listed in ADM1266\_Lib.ADM1266\_Address.

No parameter requires to be passed to this function.

Navigate to the directory where *ADM1266 Load Firmware and Configuration.py* is located.

In the command prompt type *python “ADM1266 Load Firmware and Configuration.py”* to run the script, shown in Figure 2.



Figure 2. Command to Run the Python Script

When the script is running it will display the loading step it is currently running.

At the end of the programming, the script will display the firmware version and the CRC status, as shown in Figure 3.

If the CRC check passed, it means the firmware and the configuration were loading successfully.

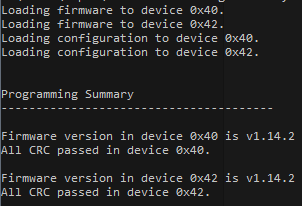


Figure 3. Programming Summary

**Closed Loop Margining Script**

**Functions used from ADM1266\_Lib:**

* ADM1266\_Lib.margin\_all(margin\_type, group\_command)

This function margins all the closed loop margin rails in devices listed in ADM1266\_Lib.ADM1266\_Address

margin\_type is passed as a string valid parameter are High, Low, Vout, Disable

group\_command is passed as a boolean. If set to True margin high will be sent as a group command to all the devices. If the parameter is left blank or False individual margining command will be set to each device

* ADM1266\_Lib.margin\_single(address, channel, margin\_type)

address is passed to the function as an integer.

channel is passed as a string which is the physical name of the input channel which needs to be margined, valid parameters are VH1-VH4, VP1-VP13.

margin\_type is passed as a string valid parameter are High, Low, Vout, Disable

* ADM1266\_Lib.margin\_single\_percent(address, pin\_number, margin\_percent)

This function is used to update the closed loop margin thresholds by percentage. The function automatically reads back the nominal voltage and sets the margin high and low limit by required percentage.

address is passed to the function as an integer

pin\_number is the index of the input pin for which the margining thresholds need to be updated, valid inputs are 0 – 16

margin\_percent is the percentage of the nominal to which the margin high and low thresholds should be set, valid inputs are any positive number

Navigate to the directory where *ADM1266 Load Firmware and Configuration.py* is located.

In the command prompt type *python “ADM1266 Margin - Closed Loop.py”* to run the script.

To margin all the rails to a certain margining status, input “a” and press enter. In the next prompt, enter one of the valid margining status: high, low vout and disable.



Figure 4: All Channel Closed Loop Margining Steps

To margin a specific rail to a certain margining status, input “s” and press enter.

Next the user will be displayed a list of all the rails that are closed loop margined, and asked to input the index of the rail that requires to be margined.

In the next prompt, enter one of the valid margining status: high, low, vout and disable.

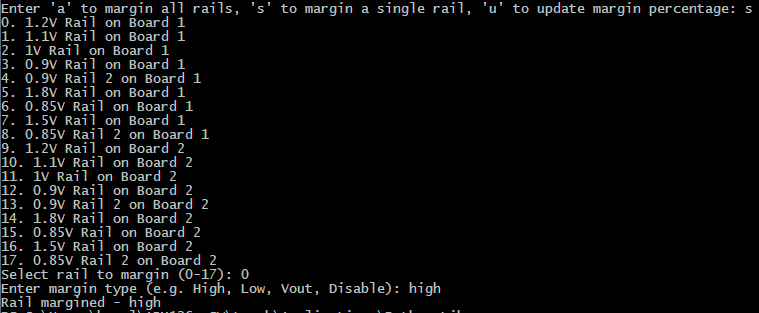


Figure 5: Single Channel Closed Loop Margining Steps

To update the margining thresholds of a specific rail to a certain value, input ‘u’ to update thresholds and then input ‘s’ to update thresholds for a single rail.

Next the user will be displayed a list of all the rails that are closed loop margined and asked to input the index of the rail that requires its threshold to be updated.

In the next prompt, enter the percentage of the nominal voltage to which the high and low margining thresholds should be set to.

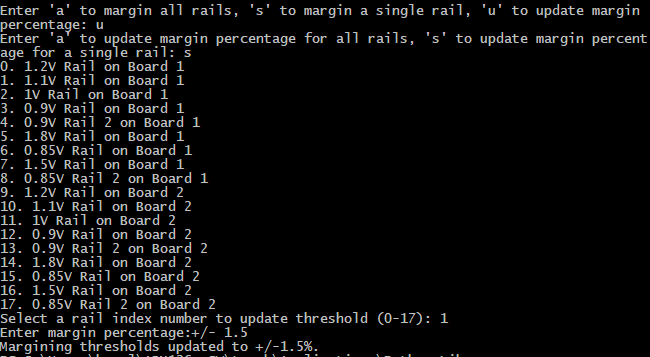


Figure 6. Update Closed Loop Margining Thresholds for a Single Channel

**Open Loop Margining Script**

**Functions used from ADM1266\_Lib:**

* ADM1266\_Lib.dac\_config(address, dac)

This function configures the DAC as open loop.

address is passed to the function as an integer.

dac is the physical DAC name passed as a string, example DAC1

* ADM1266\_Lib.margin\_open\_loop(address, dac, dac\_output)

This function sets the output voltage of a specific DAC

address is passed to the function as an integer.

dac is the physical DAC name passed as a string. Valid range is DAC1 – DAC 9.

ouput is the output voltage the DAC should output and is passed to the function a float point number. Valid range is 0.202 – 1.565.

Navigate to the directory where *ADM1266 Load Firmware and Configuration.py* is located.

In the command prompt type *python “ADM1266 Margin - Open Loop.py”* to run the script.

The user will be prompted to enter the PMBus device address of the DAC which should be configured. The address should be entered in hex format, example: 0x42.

After inputting the device address, the user will be prompted for the DAC name, the valid inputs are DAC1 to DAC9.

After that user will be asked to input expected DAC output voltage. Valid voltage is from 0.202V to 1.565V.

(Optional) If the selected DAC is configured as closed loop, the script will prompt the user if they want to configure the DAC as open loop. User can either select to configure the DAC as open loop by entering “y” or press any key if they do not want to configure the DAC as open loop.

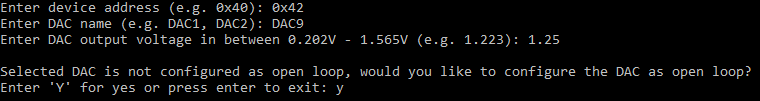


Figure 7: Open Loop Margining Steps

**ADM1266 Blackbox Read.py**

The ADM1266 Blackbox Read.py is an example of reading back blackbox information from a system consisting of a single or multiple ADM1266.

* ADM1266\_Lib.Init\_Lists()
  + A system might have a single or multiple ADM1266, and this function will dynamically initialize nested lists to store the system and blackbox data. It is important to call this function after listing the number of ADM1266 and before proceeding with any other functions
* ADM1266\_Lib.System\_Parse()
  + This function is used to readback the system information from all the ADM1266 in the system and store it in the above dynamically initialized lists. Examples of system information is, Rail names and mapping, Signal Names and State Names
* ADM1266\_Lib.Number\_Of\_Records()
  + This function is used to readback the number of blackbox records present in the ADM1266. The function reads back the data and stores it in the “ADM1266\_Lib.Num\_Records” variable for the user to access.
* ADM1266\_Lib.Get\_Raw\_Data(record\_number)
  + This function is used to readback raw blackbox data from the ADM1266. The ADM1266 can store upto 32 records at any given time. Please pass the “record\_number” to the function to specify which of the 32 records you want to readback.
* ADM1266\_Lib.BB\_Parse()
  + This function is used to parse the above raw blackbox data and fill it in the following lists
  + ADM1266\_Lib.Summary\_Data
    - This list contains the summary or the blackbox information. For eg: Record Number, Power up counter, State in which blackbox was triggered, etc.
  + ADM1266\_Lib.OV\_Rails
    - List of all the rails that have seen an OV fault in this blackbox record
  + ADM1266\_Lib.UV\_Rails
    - List of all the rails that have seen an UV fault in this blackbox record
  + ADM1266\_Lib.Normal\_Rails
    - List of all the rails that are at their nominal voltage in this blackbox record
  + ADM1266\_Lib.Disabled\_Rails
    - List of all the rails that are disabled in this blackbox record
  + ADM1266\_Lib. Signals\_Status
    - List of all the Signals along with their output and input values in this blackbox record
* ADM1266\_Lib.Blackbox\_Clear()
  + This function is used to clear the blackbox data from all the ADM1266 connected in the system
* BB\_Print()
  + This function is used to print out the blackbox data from the above lists

**ADM1266 Telemetry Read.py**

The ADM1266 Telemetry Read.py is an example of reading back Telemetry information from a system consisting of a single or multiple ADM1266.

* ADM1266\_Lib.Init\_Lists()
  + A system might have a single or multiple ADM1266, and this function will dynamically initialize nested lists to store the system and blackbox data. It is important to call this function after listening the number of ADM1266 and before proceeding with any other functions
* ADM1266\_Lib.System\_Parse()
  + This function is used to readback the system information from all the ADM1266 in the system and store it in the above dynamically initialized lists. Examples of system information is, Rail names and mapping, Signal Names and State Names
* ADM1266\_Lib.Get\_Current\_Data()
  + This function is used to readback raw data for all voltages, their status and all signals status
* (value, status, name) = ADM1266\_Lib.Get\_Rail\_Current\_Data(address, page)
  + This function is used to readback raw data for voltage and status of the ADM1266 device whose “address” is passed to the function and pin number which is passed to the function as “page”
  + The “address” passed to the function is the index of the ADM1266 address specified in the “ADM1266\_Lib.ADM1266\_Address” list
  + The “page” passed to the function is the value corresponding to the 0x00—Page command, as specified in the datasheet.
    - 0 = VH1, 1 = VH2, 2 = VH3, 3 = VH4, 4 = VP1, 5 = VP2, 6 = VP3, 7 = VP4, 8 = VP5, 9 = VP6, 10 = VP7, 11 = VP8, 12 = VP9, 13 = VP10, 14 = VP11, 15 = VP12, 16 = VP13
  + This function returns the voltage at the specified pin, it’s fault and warning status and the name of the rail assigned to the pin
* (status, name) = ADM1266\_Lib.Get\_Signal\_Current\_Data(address, index)
  + This function is used to readback raw data for status of the ADM1266 device whose “address” is passed to the function and pin number which is passed to the function as “index”
  + The “address” passed to the function is the index of the ADM1266 address specified in the “ADM1266\_Lib.ADM1266\_Address” list
  + The “index” passed to the function is the value corresponding to the list below
    - 0 = PDIO1, 1 = PDIO 2, 2 = PDIO 3, 3 = PDIO 4, 4 = PDIO5, 5 = PDIO6, 6 = PDIO7, 7 = PDIO8, 8 = PDIO9, 9 = PDIO10, 10 = PDIO11, 11 = PDIO12, 12 = PDIO13, 13 = PDIO14, 14 = PDIO15, 15 = PDIO16, 16 = GPIO1, 17 = GPIO2, 18 = GPIO3, 19 = GPIO4, 20 = GPIO5, 21 = GPIO6, 22 = GPIO7, 23 = GPIO8, 24 = GPIO9
  + This function returns, the logic level status and the name of the signal assigned to the pin
* ADM1266\_Lib.Rails\_I\_Status()
  + This function is used to parse the raw voltage telemetry data and fill it in the following lists
  + ADM1266\_Lib.OV\_I\_Rails
    - List of all the rails that currently have an OV fault
  + ADM1266\_Lib.UV\_I\_Rails
    - List of all the rails that currently have an UV fault
  + ADM1266\_Lib.OVW\_I\_Rails
    - List of all the rails that currently have an OV warning
  + ADM1266\_Lib.UVW\_I\_Rails
    - List of all the rails that currently have an UV warning
  + ADM1266\_Lib.Normal\_I\_Rails
    - List of all the rails that are at their nominal voltage
  + ADM1266\_Lib.Disabled\_I\_Rails
    - List of all the rails that are disabled
* ADM1266\_Lib.Signals\_I\_Status\_Fill()
  + This function is used to parse the raw voltage telemetry data and fill it in the following lists
  + ADM1266\_Lib.Signals\_I\_Status
    - List of all the Signals along with their output and input values