

## ***Host Interface Control of the TPS65982 USB Type-C and PD Controller from a Windows-Based PC***

---

*Steve Preissig**Advanced Circuit Solutions*

### **ABSTRACT**

This application note provides instructions for installing and executing a set of python scripts used to interact with the TPS65982 USB Type-C and PD controller using the device's Host Interface and an external Windows-based computer. This interface provides a rich variety of functions for both controlling the TPS65982 and for acquiring status information.

The Host Interface is implemented via I2C. In order to communicate with a Windows system, the "Aardvark" USB to SPI/I2C adapter from Total Phase is used. Total phase provides a Windows DLL and example python scripts for interacting with a generic I2C or SPI device.

This application note provides instructions for installing all necessary software onto a Windows platform. (Aardvark has Linux driver support, although installation to a Linux system is not covered in this application note.) The application note also provides instructions for connecting the Aardvark adapter to the daughter card connector of the TPS65982-EVM kit.

#### **Hardware Required:**

- A windows-based PC with at least one USB2.0 (or later) port
- A TPS65982-EVM
- A Total Phase "Aardvark" USB to SPI/I2C adapter

## Contents

<b>1</b>	<b>Installing Python .....</b>	<b>3</b>
<b>2</b>	<b>Installing the Aardvark Driver .....</b>	<b>6</b>
<b>3</b>	<b>Connect Aardvark to TPS65982-EVM via I2C .....</b>	<b>8</b>
<b>4</b>	<b>Execute Python Scripts .....</b>	<b>11</b>
4.1	Install the Host Interface Utility Scripts .....	13
4.2	Test the read_registers.py script .....	15
4.3	Test the flash_update_image.py and flash_update_region_0.py scripts.....	16
4.4	Test the pd_trace.py script .....	18

## Figures

<b>Figure 1.</b>	<b>Total Phase Aardvark I2C/SPI Host Adapter Pin Connections and Descriptions .....</b>	<b>8</b>
<b>Figure 2.</b>	<b>Texas Instruments TPS65982-EVM J2 Pin Connections .....</b>	<b>9</b>
<b>Figure 3.</b>	<b>Aardvark to TPS65982-EVM Wiring Diagram .....</b>	<b>9</b>
<b>Figure 4.</b>	<b>A Correctly-Wired System .....</b>	<b>10</b>
<b>Figure 5.</b>	<b>Execution of read_registers.py .....</b>	<b>15</b>
<b>Figure 6.</b>	<b>Execution of flash_update_image.py .....</b>	<b>16</b>
<b>Figure 7.</b>	<b>Execution of flash_update_region_0.py.....</b>	<b>17</b>
<b>Figure 8.</b>	<b>Execution of flash_update_region_1.py.....</b>	<b>17</b>
<b>Figure 9.</b>	<b>Partial execution of pd_trace.py .....</b>	<b>18</b>
<b>Figure 10.</b>	<b>Completed Execution of pd_trace.py Script.....</b>	<b>19</b>

## 1 Installing Python

The example scripts used in this application note are based on the Python version 2.7 interpreted language.

### 1. Download the python version 2.7 installer

Total Phase, the maker of the Aardvark adapter, recommends using python version 2.7, even though this is not the latest version of python.

Download the python installer for your platform. At the time of the writing of this document, the installer is located under the “Download” section of the following page:

<https://www.python.org/download/releases/2.7>

### Download

This is a production release. Please [report any bugs](#) you encounter.

We currently support these formats for download:

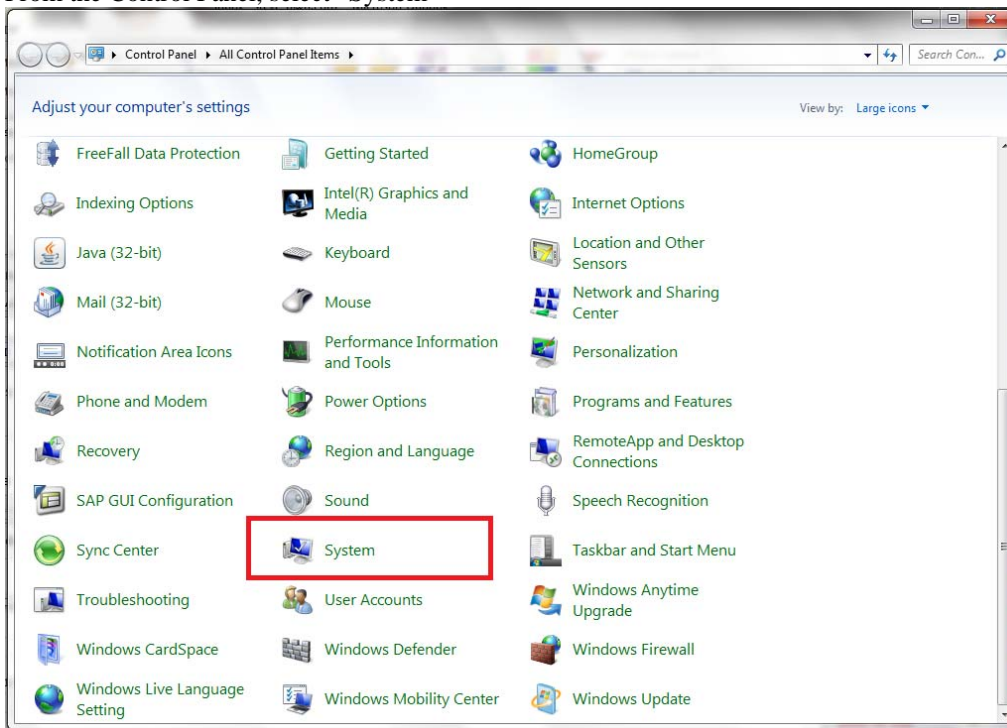
- [Gzipped source tar ball \(2.7\) \(sig\)](#)
- [Bzipped source tar ball \(2.7\) \(sig\)](#)
- [Windows x86 MSI Installer \(2.7\) \(sig\)](#)
- [Windows X86-64 MSI Installer \(2.7\) \[1\] \(sig\)](#)
- [Mac Installer disk image \(2.7\) for OS X 10.5 and later \(sig\)](#). It contains code for PPC, i386, and x86-64.
- [32-bit Mac Installer disk image \(2.7\) for OS X 10.3 and later \(sig\)](#).
- [Windows help file \(sig\)](#)

### 2. Install python for your system

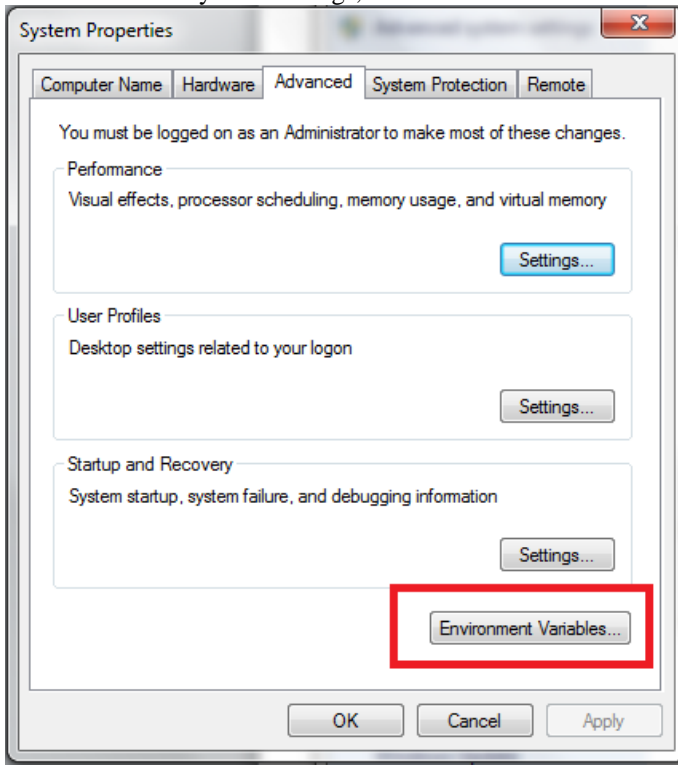
This procedure is tested for a 64-bit x86 platform. The “Windows X86-64 MSI Installer (2.7)” option was chosen to download “python-2.7.amd64.msi”

In this case the Microsoft Installer (.msi) is launched by left-clicking in the Windows explorer.

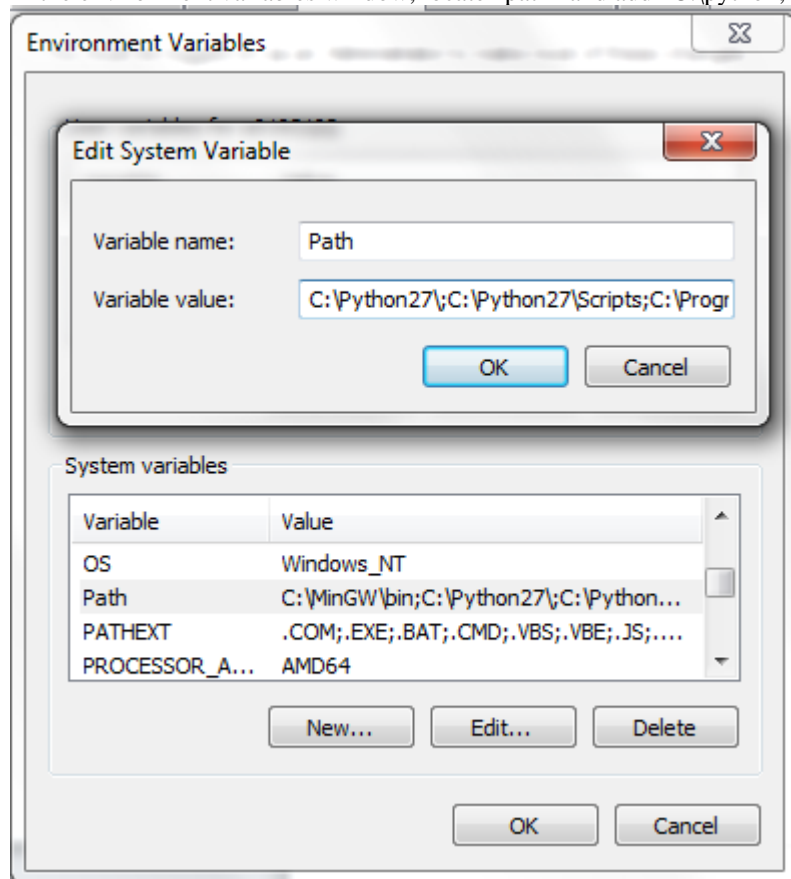
3. **Add python executable to Windows path variable**  
From the Windows start menu, select “Control Panel”  
From the Control Panel, select “System”



From System menu, select “Advanced System Settings”  
From Advanced System Settings, select “Environment Variables”



In the environment variables window, locate “path” and add “C:\python;C:\python\scripts” as below:



Press “OK” to save

## 2 Installing the Aardvark Driver

### 1. Download the Aardvark driver for your system

At the time of the writing of this document, the Windows drivers at the following page:

<http://www.totalphase.com/products/usb-drivers-windows/>



Select the “USB Drivers – Windows v2.12” link to download “totalphaseusb-v2.12\_1.zip”

### 2. Install the Aardvark drivers

Extract the zip file and execute the included file “TotalPhaseUSB-v2.12.exe”

You must execute the installer with administrative privileges. On Windows 7 you can do this by right-clicking the installer and choosing “Run as Administrator”

### 3. Download the Aardvark Software API

At the time of the writing of this document, these drivers are located at the following page:

<http://www.totalphase.com/products/aardvark-software-api/>

## Aardvark Software API

Part Number: TP200110  
Distribution: Download

Rosetta Language Bindings - 32-bit and 64-bit Software API and Shared Library for C, C#, Python, .NET, VB.NET, and VB6. Note that this software requires an update of the firmware to version 3.50. Please read the UPGRADE.txt in the package because some API calls have been changed.

Please click on the appropriate software version for your operating system. Login is required for software downloads. If you don't have an account, you will be prompted to create an account before your download commences.



**SOFTWARE**

**TOTAL PHASE**

**Download Software Here**

- [Aardvark Software API v5.15 \(Windows 32-bit\)](#)
- [Aardvark Software API v5.15 \(Windows 64-bit\)](#)
- [Aardvark Software API v5.15 \(Linux 32-bit\)](#)
- [Aardvark Software API v5.15 \(Linux 64-bit\)](#)
- [Aardvark Software API v5.15 \(Mac 32-bit\)](#)
- [Aardvark Software API v5.15 \(Mac 64-bit\)](#)

[Reviews](#)
[Related Products](#)

Select the “Aardvark Software API v5.15 (Windows 64-bit)” link to download “aardvark-api-windows-x86\_64-v5.15.zip”

#### 4. De-archive the Aardvark Software API

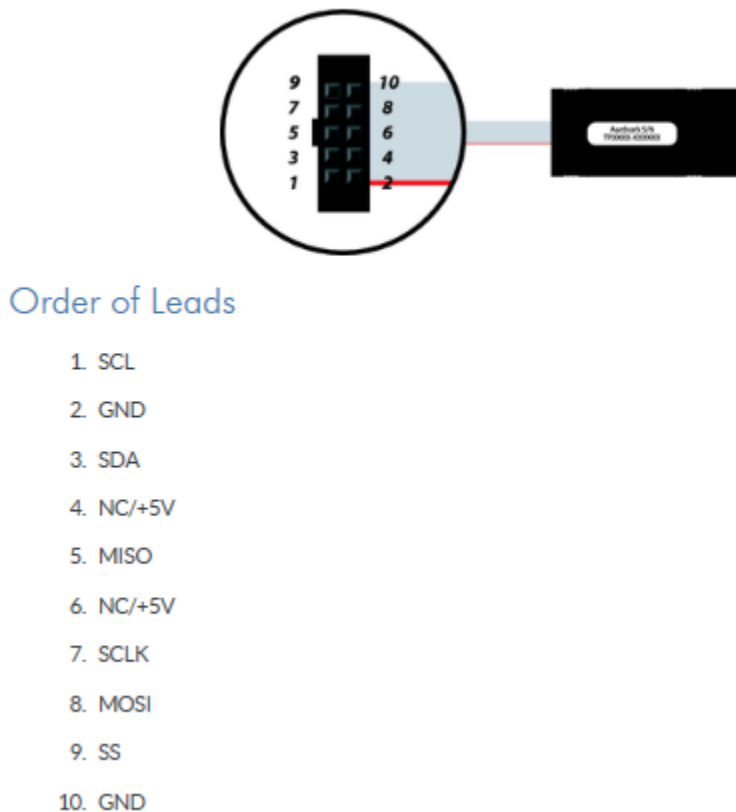
You may unzip the archive to any location on your computer that is convenient. Note that you will be installing the TI debugging scripts as an overlay on top of these files in section 4.

### 3 Connect Aardvark to TPS65982-EVM via I2C

The Aardvark connector has a single I2C line that needs to be physically connected to the I2C1 of the TPS65982

Figure 1 is adapted from the Aardvark User's Guide at <http://www.totalphase.com/support/articles/200468316> and shows the pins of the Aardvark connector. Figure 2 is adapted from the design files of the TPS65982-EVM and shows the pin-outs of the J2 connector. Finally, Figure 3 shows the wiring diagram to wire the I2C lines of the Aardvark connector into the J2 connector of the TPS65982-EVM.

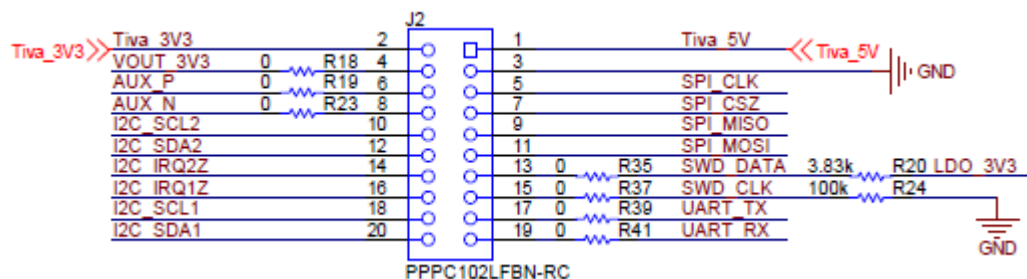
If you flip your Aardvark adapter over (figure 6) such that the text on the serial number label is in the proper upright position, the pin order is as shown in the following diagram.



**Figure 1. Total Phase Aardvark I2C/SPI Host Adapter Pin Connections and Descriptions**

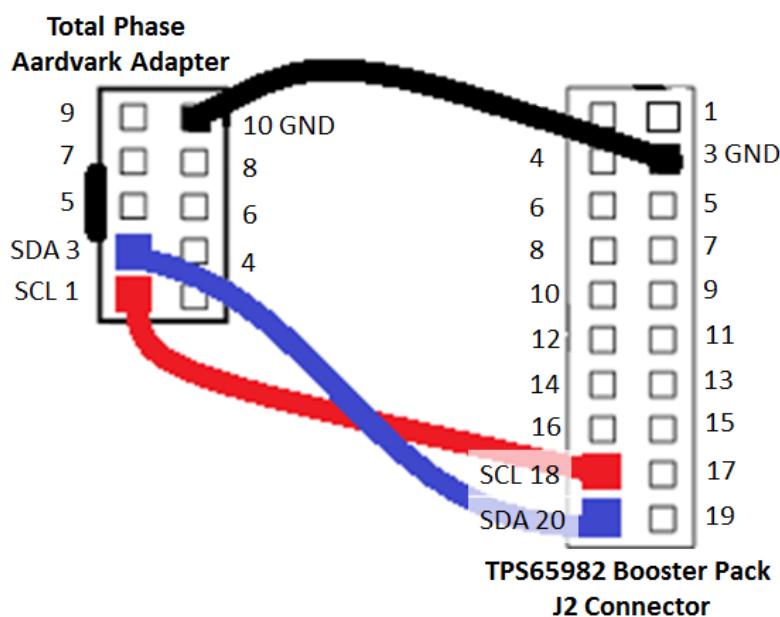
This diagram shows the position of the three I2C lines required on the Total Phase Aardvark adapter: Serial Clock, SCL (1), Serial Data (SDA) and ground, GDN (10)





**Figure 2. Texas Instruments TPS65982-EVM J2 Pin Connections**

This diagram shows the position of the three I2C lines required on the TPS65982-EVM J2 Connector, Serial Clock, I2C\_SCL1 (18), Serial Data, I2C\_SDA1 (20) and ground, GND (3). This document shows the connections for using this port, which is I2C port 1. The Host Interface will also recognize commands sent over I2C port 2. Note that the position of these pins is shown as looking down at the board from above.



**Figure 3. Aardvark to TPS65982-EVM Wiring Diagram**

This diagram shows the wire connections between the Total Phase Aardvark and the TPS65982-EVM J2 Connector



**Figure 4. A Correctly-Wired System**

An Aardvark adapter wired into the TPS65982-EVM. Note that the orientation of the TPS65982 board and the Aardvark connector match the diagram of Figure 3.

## 4 Execute Python Scripts

Texas Instruments provides a collection of python-based scripts for interacting with the TPS65982 using the Host Interface accessed via I2C. These scripts are organized into the files presented in Table 1.

**Table 1. TPS65982 Host Interface Python Utility Files**

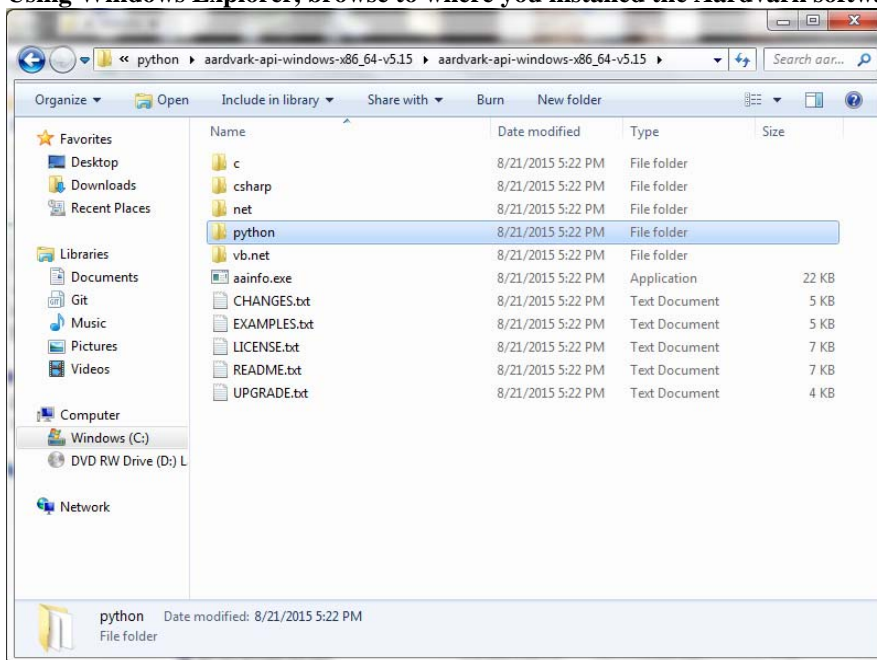
File	Description
aardvark_py.py	This file contains low-level python functions for interacting with the Total Phase Aardvark. It is not meant to be called.
aardvark_rw.py	This file builds upon aardvark_py.py to provide basic read and write commands using I2C. It is not meant to be called.
alternate_modes.py	This script will discover and then attempt to enter all modes for the following Standard and Vendor ID's (SVIDs): 0x0451 (Texas Instruments), 0x8086 (Intel), 0x8087 (Intel), 0xFF01 (Display Port)
debug_trace.py	This file builds upon the register class definitions of register_class.py to define the trace registers of the host interface. It is not meant to be called. (Use pd_trace.py for an example of using these registers.)
flash_update_image.py	This file builds upon the flash programming 4cc function definitions defined in hi_functions.py. It is used to update the SPI flash attached to TPS65982 using the host interface and a full flash binary image.
flash_update_region_0.py	This file builds upon the flash programming 4cc function definitions defined in hi_functions.py. It is used to update region 0 of the SPI flash attached to TPS65982 using the host interface and an application binary.
flash_update_region_1.py	This file builds upon the flash programming 4cc function definitions defined in hi_functions.py. It is used to update region 0 of the SPI flash attached to TPS65982 using the host interface and an application binary.
hi_functions.py	This file defines a number of python functions to call corresponding 4cc host interface functions over the TPS65982 I2C interface. It is not meant to be called.
intrusive_PD_example.py	This example demonstrates executing the Power Delivery (PD) features of the TPS65982 in intrusive mode, which disables the automatic PD negotiation of the firmware and allows the external controller (EC) to manually step through the negotiation process, providing greater control.
pd_trace.py	This file builds upon the register definitions of debug_trace.py. It is used to read and display a circular buffer of 64 PD status change messages maintained by the TPS65982 for debugging purposes.

read_registers.py	This file builds upon the register definitions of register_definitions.py. It is used to read and display the basic (non-trace) registers of the TPS65982 host interface.
register_class.py	This file defines an object class for the virtual registers of the TPS65982 host interface and various interaction methods for reading, writing and displaying the contents of these registers. It is not meant to be called.
register_definitions.py	This file builds upon the register class definitions of register_class.py to define the basic registers of the host interface. It is not meant to be called. (Use read_registers.py for an example of using these registers.)
send_commands.py	This script allows the user to send various host interface commands via I2C to a running TPS65982 controller.
set_dp_cap.py	This script demonstrates dynamic reconfiguration of the DisplayPort settings of the TPS65982.
set_sleep_level.py	This script demonstrates configuration of the TPS65982 to enter sleep mode when inactive.
set_sourcesink_cap.py	This script demonstrates dynamic reconfiguration of the source and sink capabilities of the TPS65982.

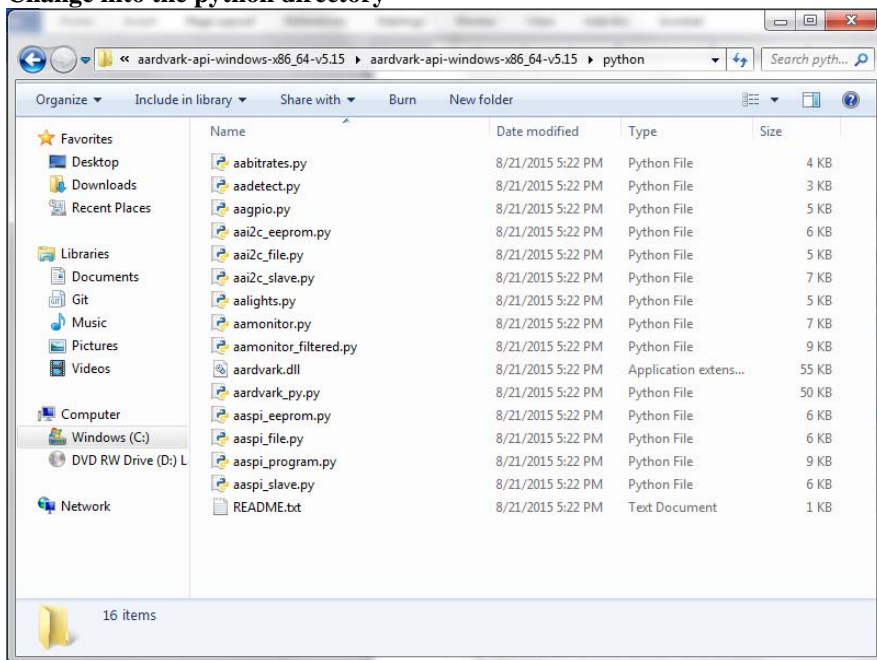
## 4.1 Install the Host Interface Utility Scripts

These scripts cannot run without the aardvark.dll provided by Total Phase and which you should have installed in section 2, step 4. The following steps outline the procedure for overlaying the Host Interface Utility scripts on top of the Aardvark software API from Total Phase.

1. Using Windows Explorer, browse to where you installed the Aardvark software API in section 2, step 4




































2. Change into the python directory



### 3. De-archive the files of “SLVA701\_files\_Host\_Interface\_Utility\_<date>.zip” into this directory

Note that only the files “aardvark.dll” and “aardvark\_py.py” are required from the aardvark API.

Name	Type	Compressed size	Password ...	Size	Ratio	Date modified
 aadetct.py	Python File	2 KB	No	3 KB	60%	2/28/2014 11:00 AM
 aagpio.py	Python File	2 KB	No	5 KB	63%	2/28/2014 11:00 AM
 aai2c_eeprom.py	Python File	3 KB	No	6 KB	65%	2/28/2014 11:00 AM
 aai2c_file.py	Python File	2 KB	No	5 KB	64%	2/28/2014 11:00 AM
 aai2c_slave.py	Python File	3 KB	No	7 KB	65%	2/28/2014 11:00 AM
 aalights.py	Python File	2 KB	No	5 KB	66%	2/28/2014 11:00 AM
 aamonitor.py	Python File	3 KB	No	7 KB	68%	2/28/2014 11:00 AM
 aamonitor_filtered.py	Python File	3 KB	No	9 KB	70%	2/28/2014 11:00 AM
 aardvark.dll	Application extension	23 KB	No	55 KB	58%	2/28/2014 11:00 AM
 aardvark_py.py	Python File	10 KB	No	50 KB	82%	4/15/2015 6:06 PM
 aardvark_rw.py	Python File	3 KB	No	7 KB	65%	7/13/2015 5:10 PM
 aaspi_eeprom.py	Python File	3 KB	No	6 KB	65%	2/28/2014 11:00 AM
 aaspi_file.py	Python File	2 KB	No	6 KB	65%	2/28/2014 11:00 AM
 aaspi_program.py	Python File	3 KB	No	9 KB	67%	2/28/2014 11:00 AM
 aaspi_slave.py	Python File	2 KB	No	6 KB	64%	2/28/2014 11:00 AM
 alternate_modes.py	Python File	2 KB	No	5 KB	70%	8/10/2015 6:57 PM
 debug_trace.py	Python File	6 KB	No	35 KB	85%	8/21/2015 2:23 PM
 flash_update_image.py	Python File	2 KB	No	4 KB	62%	7/20/2015 1:37 PM
 flash_update_region_0.py	Python File	3 KB	No	10 KB	69%	8/21/2015 4:19 PM
 flash_update_region_1.py	Python File	3 KB	No	10 KB	69%	8/21/2015 4:27 PM
 hi_functions.py	Python File	3 KB	No	17 KB	84%	8/17/2015 10:52 AM
 intrusive_PD_example.py	Python File	3 KB	No	7 KB	69%	8/17/2015 12:42 PM
 pd_trace.py	Python File	2 KB	No	3 KB	60%	6/22/2015 5:11 PM
 read_registers.py	Python File	2 KB	No	4 KB	60%	7/13/2015 4:19 PM
 README.txt	Text Document	1 KB	No	1 KB	45%	2/28/2014 11:00 AM
 register_class.py	Python File	3 KB	No	9 KB	71%	8/4/2015 1:50 PM
 register_definitions.py	Python File	15 KB	No	144 KB	90%	8/21/2015 3:53 PM
 send_commands.py	Python File	3 KB	No	10 KB	79%	8/20/2015 4:49 PM
 set_dp_cap.py	Python File	2 KB	No	7 KB	74%	8/6/2015 6:15 PM
 set_sleep_level.py	Python File	2 KB	No	4 KB	63%	8/21/2015 3:53 PM
 set_sourcetrace_cap.py	Python File	2 KB	No	7 KB	75%	8/2/2015 11:06 AM
 SLVA701 TPS65982 Host Interface ...	Adobe Acrobat Document	900 KB	No	1,016 KB	12%	8/21/2015 4:54 PM
 TPS65982 HI Util TI Only_1.0_manif...	HTML Document	6 KB	No	21 KB	75%	8/21/2015 4:00 PM

## 4.2 Test the read\_registers.py script

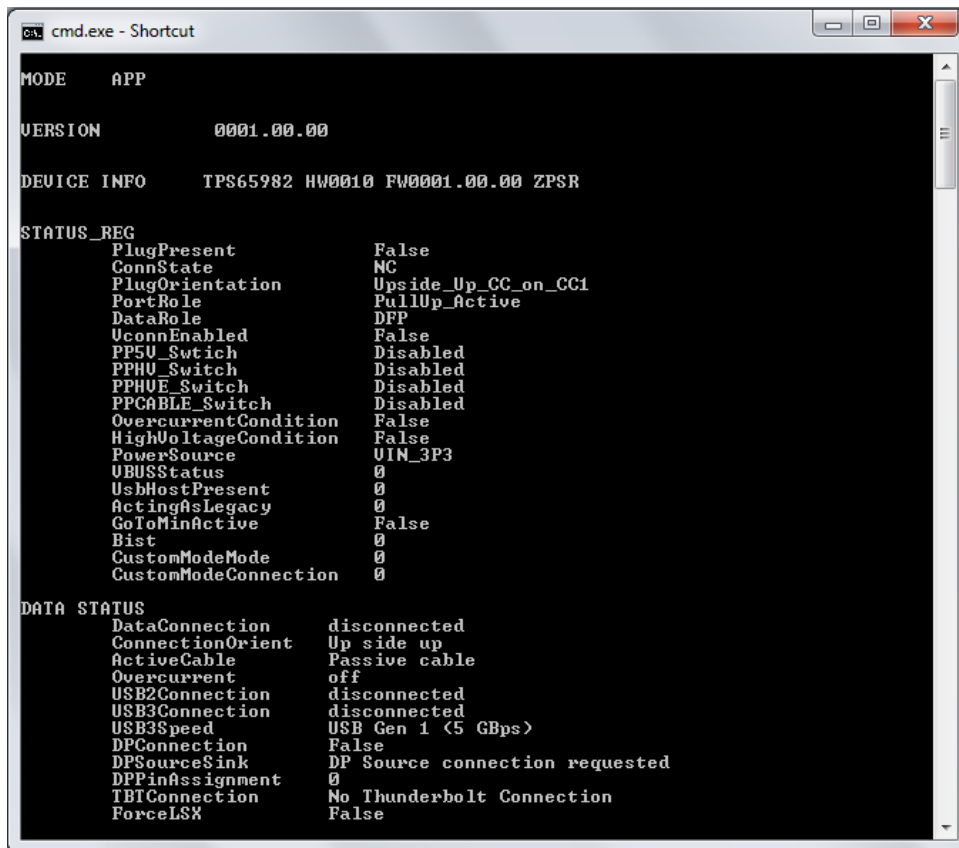
1. Unzip the SLVA701\_files\_Host\_Interface\_Utility.zip folder to the location of your choice
2. Attach the Total Phase Aardvark I2C adapter to a TPS65982-EVM
3. Make sure that the TPS65982-EVM is powered and has a valid firmware image
4. Launch a windows command window  
run the application cmd.exe, which is usually located at  
C:\Windows\System32\cmd.exe
5. Within the command window browse to the directory (using cd command) where you have installed the python script files in step 1.
6. Execute the read\_registers.py script

At the windows command window prompt, type the name of the script:

C:\path> read\_registers.py

This script assumes an I2C address of 0x38 for the target board. If the I2C address of the target board is different, you can overwrite this default as an optional argument. To run the script for an I2C address of 0x3C,

C:\path> read\_registers.py 0x3C



```

cmd.exe - Shortcut

MODE    APP

VERSION    0001.00.00

DEVICE INFO    TPS65982 HW0010 FW0001.00.00 ZPSR

STATUS_REG
PlugPresent      False
ConnState        NC
PlugOrientation   Upside_Up_CC_on_CC1
PortRole         PullUp_Active
DataRole         DFP
UconnEnabled     False
PP5V_Switch      Disabled
PPH0_Switch      Disabled
PPH0E_Switch     Disabled
PPCABLE_Switch   Disabled
OvercurrentCondition  False
HighVoltageCondition  False
PowerSource      UIN_3P3
VBUSStatus       0
UsbHostPresent   0
ActingAsLegacy   0
GoToMinActive    False
Bist             0
CustomModeMode   0
CustomModeConnection  0

DATA STATUS
DataConnection   disconnected
ConnectionOrient Up side up
ActiveCable       Passive cable
Overcurrent       off
USB2Connection   disconnected
USB3Connection   disconnected
USB3Speed        USB Gen 1 (5 Gbps)
DPConnection     False
DPSourceSink     DP Source connection requested
DPPinAssignment  0
TBTConnection    No Thunderbolt Connection
ForceLSX         False
  
```

Figure 5. Execution of read\_registers.py



### 4.3 Test the flash\_update\_image.py and flash\_update\_region\_0.py scripts

The TPS65982 Utility scripts include three flash programming examples. These examples program the SPI Flash attached to the TPS65982 using the TPS65982 Host Interface.

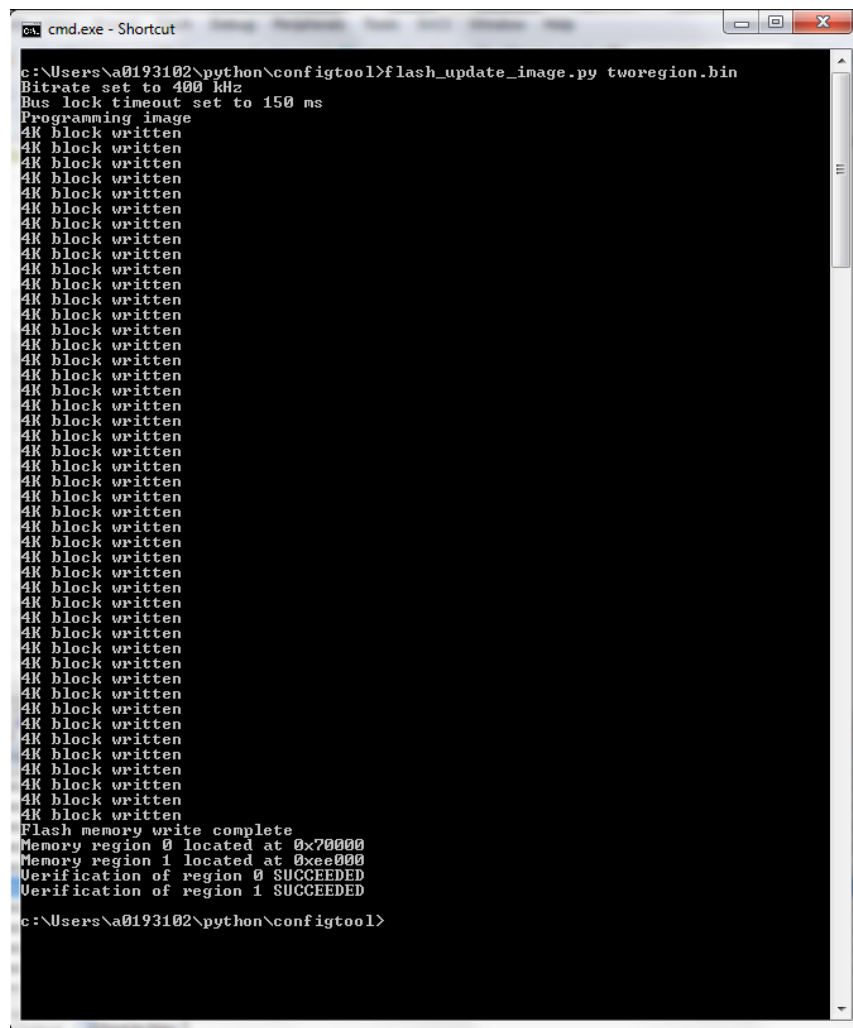
The TPS65982 bootloader specifies two regions in which an application image may reside. These regions provide redundancy in case the Flash is corrupted via field update or another mechanism. The device will attempt to boot from region 0, and if this boot fails, will then attempt to boot from region 1.

The “flash\_update\_image.py” script reprograms the entire flash using a full flash image. This image is written into flash offset 0. In order to use this utility, the flash image programmed must contain region records as well as boot headers for the images in each region. The script takes 1 required argument and one optional argument:

```
flash_update_image.py <filename.bin> [I2C address]
```

for instance,

```
flash_update_image.py towregion.bin 0x3C
```



**Figure 6. Execution of flash\_update\_image.py**

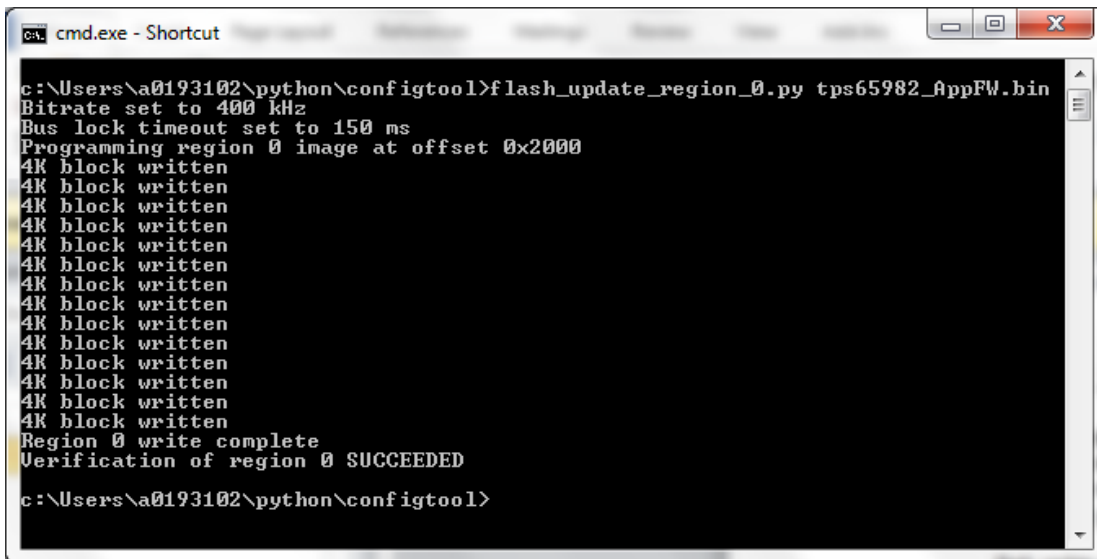


The “flash\_update\_region\_0.py” and “flash\_update\_region\_1.py” scripts update a single region, 0 or 1, respectively. These scripts expect an application binary which does not contain region records or a boot header. These scripts likewise take the single required filename argument and the optional I2C address argument.

flash\_update\_region\_0.py <filename.bin> [I2C address]

for instance,

flash\_update\_region\_0.py apponly.bin 0x3C

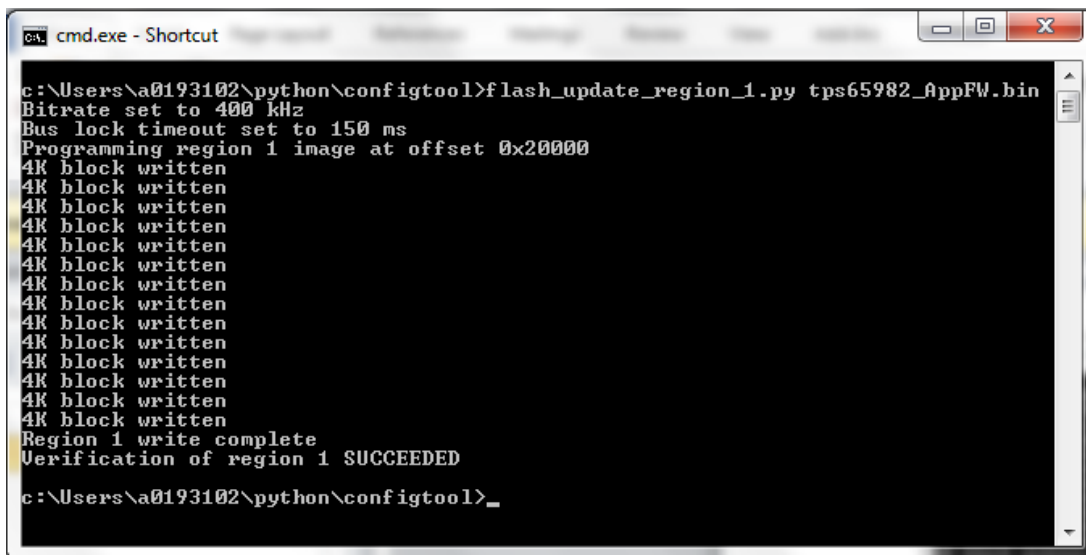


```

c:\Users\ao193102\python\configtool>flash_update_region_0.py tps65982_AppFW.bin
Bitrate set to 400 kHz
Bus lock timeout set to 150 ms
Programming region 0 image at offset 0x2000
4K block written
4K block written
4K block written
4K block written
4K block written
4K block written
4K block written
4K block written
4K block written
4K block written
4K block written
4K block written
Region 0 write complete
Verification of region 0 SUCCEEDED
c:\Users\ao193102\python\configtool>

```

**Figure 7. Execution of flash\_update\_region\_0.py**



```

c:\Users\ao193102\python\configtool>flash_update_region_1.py tps65982_AppFW.bin
Bitrate set to 400 kHz
Bus lock timeout set to 150 ms
Programming region 1 image at offset 0x2000
4K block written
4K block written
4K block written
4K block written
4K block written
4K block written
4K block written
4K block written
4K block written
4K block written
4K block written
4K block written
Region 1 write complete
Verification of region 1 SUCCEEDED
c:\Users\ao193102\python\configtool>

```

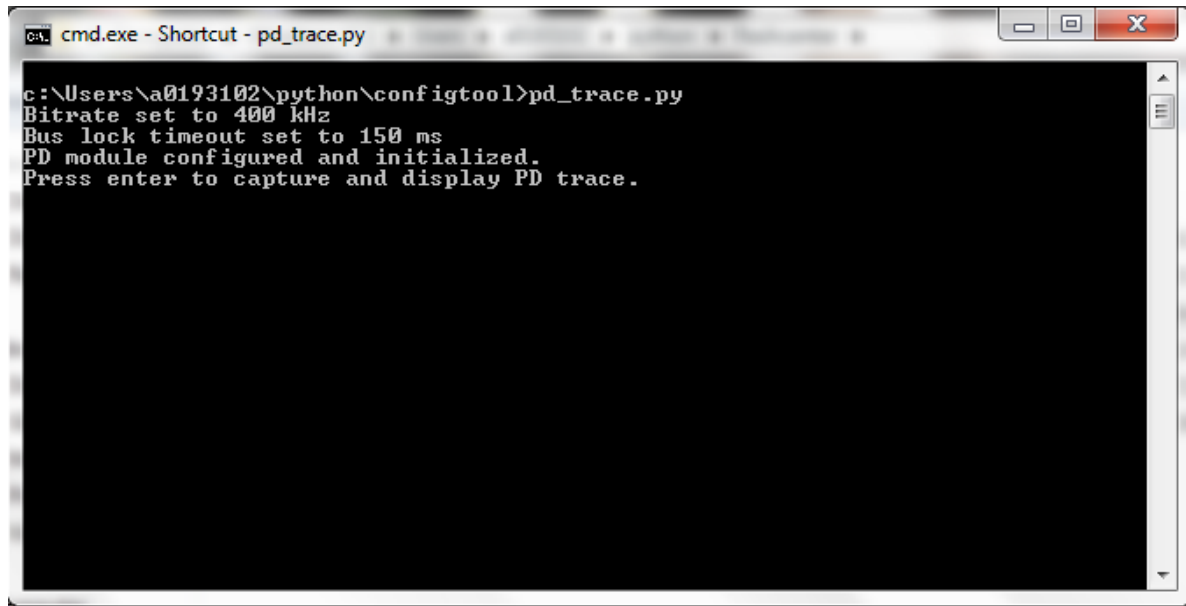
**Figure 8. Execution of flash\_update\_region\_1.py**

#### 4.4 Test the pd\_trace.py script

The final host interface utilities example program is `pd_trace.py`

This script uses the trace registers defined in `debug_trace.py` to read a circular buffer of PD state transitions that are captured on the TPS65982.

Before the script is able to read this data, it needs to configure the trace module and initialize the circular buffer to 0xFF, the circular buffer marker. The script will complete these tasks then prompt the user to plug in the type-C cable to initiate PD negotiation as shown in Figure 9.



```
cmd.exe - Shortcut - pd_trace.py
c:\Users\A0193102\python\configtool>pd_trace.py
Btrate set to 400 kHz
Bus lock timeout set to 150 ms
PD module configured and initialized.
Press enter to capture and display PD trace.
```

**Figure 9. Partial execution of `pd_trace.py`**

At this point the user attaches the type-C cable and waits for the PD negotiation to complete. This is very quick, a waiting period of 1-2 seconds is sufficient. Afterwards, press the enter key to read the circular buffer as shown in Figure 10.

```

c:\Users\A0193102\python\configtool>pd_trace.py
Bitrate set to 400 kHz
Bus lock timeout set to 150 ms
PD module configured and initialized.
Press enter to capture and display PD trace.
Firmware State Focus
Ring Buffer 0      0xFF <Circular Buffer Entry Point>
Ring Buffer 1      PESTate_LaunchPolicyEngine
Ring Buffer 2      PESTate_Enable_UCONN
Ring Buffer 3      PESTate_Source_Startup
Ring Buffer 4      PESTate_Source_SendCapabilities
Ring Buffer 5      PESTate_Source_Discovery
Ring Buffer 6      PESTate_Source_SendCapabilities
Ring Buffer 7      PESTate_Source_Discovery
Ring Buffer 8      PESTate_Source_SendCapabilities
Ring Buffer 9      PESTate_Source_NegotiateCapability
Ring Buffer 10     PESTate_Source_TransitionSupply_Accept
Ring Buffer 11     PESTate_Source_TransitionSupply
Ring Buffer 12     PESTate_Source_TransitionSupply_SetAlarmsNew
Ring Buffer 13     PESTate_Source_TransitionSupply_PS_RDY
Ring Buffer 14     PESTate_Source_Ready
Ring Buffer 15     PESTate_DRS_Evaluate_DR_Swap
Ring Buffer 16     PESTate_DRS_Accept_DR_Swap
Ring Buffer 17     PESTate_DRS_DFP_UFP_Change_to_UFP
Ring Buffer 18     PESTate_Source_Ready
Ring Buffer 19     PESTate_UFP_UDM_Send_Identity
Ring Buffer 20     PESTate_Source_Ready
Ring Buffer 21     PESTate_UFP_UDM_Get_SUIDs
Ring Buffer 22     PESTate_Source_Ready
Ring Buffer 23     PESTate_UFP_UDM_Get_Modes
Ring Buffer 24     PESTate_Source_Ready
Ring Buffer 25     PESTate_UFP_UDM_Send_Modes
Ring Buffer 26     PESTate_UFP_UDM_Evaluate_Mode_Entry
Ring Buffer 27     PESTate_Source_Ready
Ring Buffer 28     PESTate_UFP_UDM_Status_Request
Ring Buffer 29     PESTate_Source_Ready
Ring Buffer 30     PESTate_UFP_UDM_Config_Request
Ring Buffer 31     PESTate_UFP_UDM_Config_ACK
Ring Buffer 32     PESTate_Source_Ready
Ring Buffer 33     0xFF <Circular Buffer Entry Point>
Ring Buffer 34     0xFF <Circular Buffer Entry Point>
Ring Buffer 35     0xFF <Circular Buffer Entry Point>
Ring Buffer 36     0xFF <Circular Buffer Entry Point>
Ring Buffer 37     0xFF <Circular Buffer Entry Point>
Ring Buffer 38     0xFF <Circular Buffer Entry Point>
Ring Buffer 39     0xFF <Circular Buffer Entry Point>
Ring Buffer 40     0xFF <Circular Buffer Entry Point>
Ring Buffer 41     0xFF <Circular Buffer Entry Point>
Ring Buffer 42     0xFF <Circular Buffer Entry Point>
Ring Buffer 43     0xFF <Circular Buffer Entry Point>
Ring Buffer 44     0xFF <Circular Buffer Entry Point>
Ring Buffer 45     0xFF <Circular Buffer Entry Point>
Ring Buffer 46     0xFF <Circular Buffer Entry Point>
Ring Buffer 47     0xFF <Circular Buffer Entry Point>
Ring Buffer 48     0xFF <Circular Buffer Entry Point>
Ring Buffer 49     0xFF <Circular Buffer Entry Point>
Ring Buffer 50     0xFF <Circular Buffer Entry Point>
Ring Buffer 51     0xFF <Circular Buffer Entry Point>
Ring Buffer 52     0xFF <Circular Buffer Entry Point>
Ring Buffer 53     0xFF <Circular Buffer Entry Point>
Ring Buffer 54     0xFF <Circular Buffer Entry Point>
Ring Buffer 55     0xFF <Circular Buffer Entry Point>
Ring Buffer 56     0xFF <Circular Buffer Entry Point>
Ring Buffer 57     0xFF <Circular Buffer Entry Point>
Ring Buffer 58     0xFF <Circular Buffer Entry Point>
Ring Buffer 59     0xFF <Circular Buffer Entry Point>
Ring Buffer 60     0xFF <Circular Buffer Entry Point>
Ring Buffer 61     0xFF <Circular Buffer Entry Point>
  
```

Figure 10. Completed Execution of pd\_trace.py Script