

# DIODES AP33772 USB PD Sink Controller Raspberry Pi I2C Interface User Guide

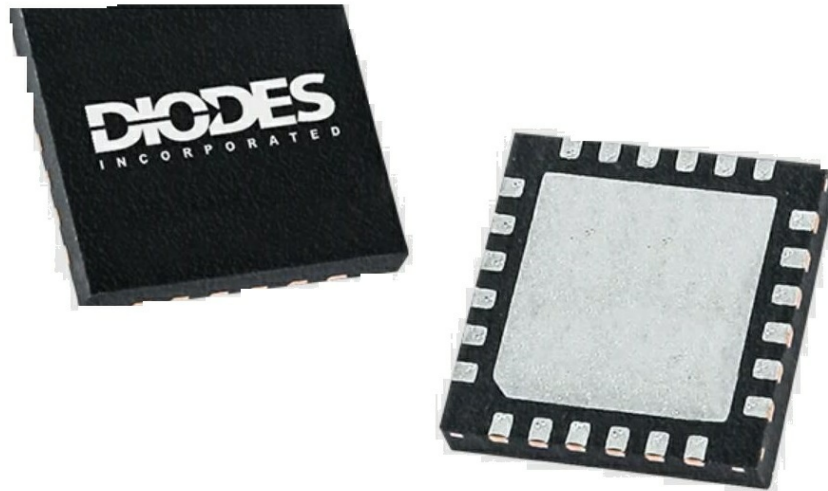
[Home](#) » [DIODES](#) » DIODES AP33772 USB PD Sink Controller Raspberry Pi I2C Interface User Guide 

## Contents

- [1 DIODES AP33772 USB PD Sink Controller Raspberry Pi I2C Interface](#)
- [2 Introduction](#)
- [3 Validation Platform Setup](#)
- [4 Raspberry Pi Software Setup](#)
- [5 Basic Command Examples](#)
- [6 Practical Examples](#)
  - [6.1 Example Code Download](#)
  - [6.2 Example Download Site](#)
- [7 References](#)
- [8 Revision History](#)
  - [8.1 IMPORTANT NOTICE](#)
- [9 Documents / Resources](#)
  - [9.1 References](#)
- [10 Related Posts](#)

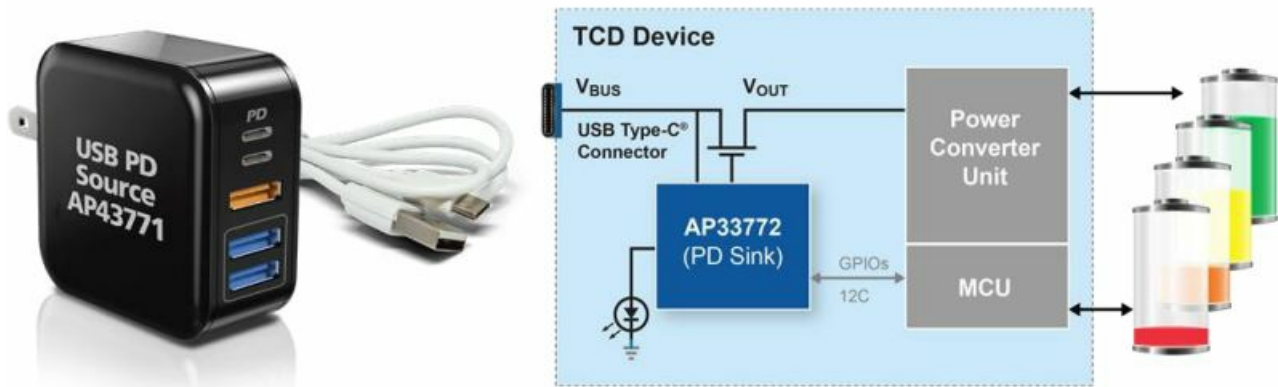


**DIODES AP33772 USB PD Sink Controller Raspberry Pi I2C Interface**



## Introduction

- AP33772 Sink Controller, working as the protocol device of USB PD3.0 Type C Connector-equipped Device (TCD, Energy Sink), is intended to request proper Power Data Object (PDO) from the USB PD3.0 Type C Connector-equipped PD3.0 compliance Charger (PDC, Energy Source).
- Figure 1 illustrates a TCD, embedded with PD3.0 Sink controller IC (AP33772), is physically connected to PDC, embedded with USB PD3.0 decoder (AP43771), through a
- Type C-to-Type C cable. Based on built-in USB PD3.0 compliant firmware, The AP33772 and AP43771 pair would go through the USB PD3.0 standard attachment procedure to establish suitable PD3.0 charging state.
- AP33772 Sink Controller EVB provides ease of use and great versatility for system designer to request PDOs from USB Power Delivery Charger by sending AP33772 built-in commands through I2C interface. Typical system design requires MCU programming which needs specific software (e.g. IDE) setup and can be a time-consuming development process.
- In contrast, Raspberry Pi (RPI), a single board computer (SBC) running on a user-friendly Linux OS and equipped with flexible GPIO pins, provides a straightforward way to validate AP33772 Sink EVB working with a PD Charger. The goal of this guide is to provide system designers an effective platform to quickly complete software validation on RPI and then port the development to any desirable MCU to meet rapid turnaround market requirements.
- As a supplemental document to the AP33772 EVB User Guide, this User Guide illustrates an easy way to control AP33772 EVB with a RPI SBC through I2C Interface.
- The role of MCU block depicted in Figure 1 to interface with AP33772 is played by an RPI. This User Guide covers a lot of register definition and usage information as examples, However, for complete and most updated information, please refer to AP33772 EVB User' Guide. (See Reference 2)

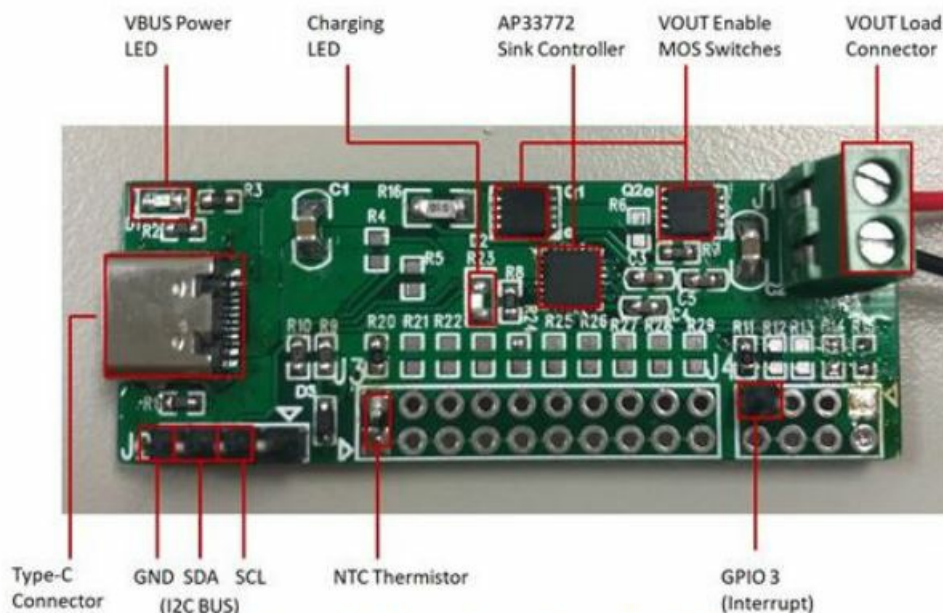


- **Figure 1** – A typical TCD uses AP33772 PD Sink Controller with I2C Interface to request power from an USB Type-C PD3.0/PPS Compliance Source Adapter

## Validation Platform Setup

### AP33772 Sink Controller EVB

**Figure 2** shows the picture of the Sink Controller EVB. It features Type-C Connector, I2C pins, GPIO3 pin for Interrupt, NTC Thermistor for OTP, LED indicators to show the charging status, and Vout connector to the load.



**Figure 2 – AP33772 Sink Controller EVB**

## Raspberry Pi Zero 2W

- Any latest version of RPI is capable of controlling AP33772 Sink Controller EVB through I2C pins. A Raspberry Pi Zero 2 W (RPI Z2W) is used in this User Guide for its cost effectiveness and versatility. It has the smallest formfactor among all RPIs and is integrated with WiFi and Bluetooth that makes the wireless connection without additional component. It serves the purpose as the AP33772 Sink Controller EVB Validation Platform perfectly.
- User may check the Raspberry Pi official website for additional information (<https://www.raspberrypi.com/products/raspberry-pi-zero-2-w/>)

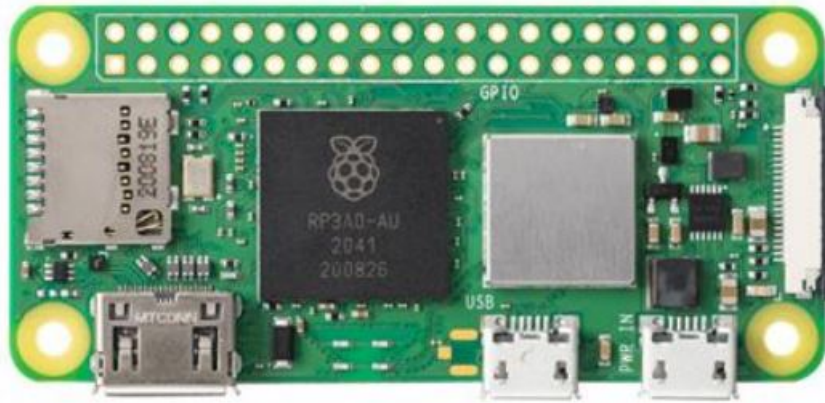


Figure 3 – Raspberry Pi Zero 2 W (RPI Z2W)

Raspberry Pi Zero (38 Header)		NAME	GPIO
1	2	3.3VDC Power	1
3	4	5.1VDC Power	2
5	6	Ground	3
7	8	GPIO 15 (TWO DART)	4
9	10	GPIO 16 (RED DART)	5
11	12	GPIO 17 (PCM_CLKIN)	6
13	14	Ground	7
15	16	GPIO 4	8
17	18	GPIO 5	9
19	20	Ground	10
21	22	GPIO 6	11
23	24	GPIO 10 (CE0 [SPI])	12
25	26	GPIO 11 (CE1 [SPI])	13
27	28	GPIO 12 (MOSI [SPI])	14
29	30	GPIO 13 (MISO [SPI])	15
31	32	GPIO 14 (SCLK [SPI])	16
33	34	Ground	17
35	36	GPIO 15 (TWO DART)	18
37	38	GPIO 16 (RED DART)	19
39	40	GPIO 17 (PCM_CLKIN)	20
41	42	Ground	21
43	44	GPIO 4	22
45	46	GPIO 5	23
47	48	Ground	24
49	50	GPIO 6	25
51	52	GPIO 10 (CE0 [SPI])	26
53	54	GPIO 11 (CE1 [SPI])	27
55	56	GPIO 12 (MOSI [SPI])	28
57	58	GPIO 13 (MISO [SPI])	29
59	60	GPIO 14 (SCLK [SPI])	30
61	62	Ground	31
63	64	GPIO 15 (TWO DART)	32
65	66	GPIO 16 (RED DART)	33
67	68	GPIO 17 (PCM_CLKIN)	34
69	70	Ground	35
71	72	GPIO 4	36
73	74	GPIO 5	37
75	76	Ground	38
77	78	GPIO 6	39
79	80	GPIO 10 (CE0 [SPI])	40
81	82	GPIO 11 (CE1 [SPI])	41
83	84	GPIO 12 (MOSI [SPI])	42
85	86	GPIO 13 (MISO [SPI])	43
87	88	GPIO 14 (SCLK [SPI])	44
89	90	Ground	45
91	92	GPIO 15 (TWO DART)	46
93	94	GPIO 16 (RED DART)	47
95	96	GPIO 17 (PCM_CLKIN)	48
97	98	Ground	49
99	100	GPIO 4	50
101	102	GPIO 5	51
103	104	Ground	52
105	106	GPIO 6	53
107	108	GPIO 10 (CE0 [SPI])	54
109	110	GPIO 11 (CE1 [SPI])	55
111	112	GPIO 12 (MOSI [SPI])	56
113	114	GPIO 13 (MISO [SPI])	57
115	116	GPIO 14 (SCLK [SPI])	58
117	118	Ground	59
119	120	GPIO 15 (TWO DART)	60
121	122	GPIO 16 (RED DART)	61
123	124	GPIO 17 (PCM_CLKIN)	62
125	126	Ground	63
127	128	GPIO 4	64
129	130	GPIO 5	65
131	132	Ground	66
133	134	GPIO 6	67
135	136	GPIO 10 (CE0 [SPI])	68
137	138	GPIO 11 (CE1 [SPI])	69
139	140	GPIO 12 (MOSI [SPI])	70
141	142	GPIO 13 (MISO [SPI])	71
143	144	GPIO 14 (SCLK [SPI])	72
145	146	Ground	73
147	148	GPIO 15 (TWO DART)	74
149	150	GPIO 16 (RED DART)	75
151	152	GPIO 17 (PCM_CLKIN)	76
153	154	Ground	77
155	156	GPIO 4	78
157	158	GPIO 5	79
159	160	Ground	80
161	162	GPIO 6	81
163	164	GPIO 10 (CE0 [SPI])	82
165	166	GPIO 11 (CE1 [SPI])	83
167	168	GPIO 12 (MOSI [SPI])	84
169	170	GPIO 13 (MISO [SPI])	85
171	172	GPIO 14 (SCLK [SPI])	86
173	174	Ground	87
175	176	GPIO 15 (TWO DART)	88
177	178	GPIO 16 (RED DART)	89
179	180	GPIO 17 (PCM_CLKIN)	90
181	182	Ground	91
183	184	GPIO 4	92
185	186	GPIO 5	93
187	188	Ground	94
189	190	GPIO 6	95
191	192	GPIO 10 (CE0 [SPI])	96
193	194	GPIO 11 (CE1 [SPI])	97
195	196	GPIO 12 (MOSI [SPI])	98
197	198	GPIO 13 (MISO [SPI])	99
199	200	GPIO 14 (SCLK [SPI])	100
201	202	Ground	101
203	204	GPIO 15 (TWO DART)	102
205	206	GPIO 16 (RED DART)	103
207	208	GPIO 17 (PCM_CLKIN)	104
209	210	Ground	105
211	212	GPIO 4	106
213	214	GPIO 5	107
215	216	Ground	108
217	218	GPIO 6	109
219	220	GPIO 10 (CE0 [SPI])	110
221	222	GPIO 11 (CE1 [SPI])	111
223	224	GPIO 12 (MOSI [SPI])	112
225	226	GPIO 13 (MISO [SPI])	113
227	228	GPIO 14 (SCLK [SPI])	114
229	230	Ground	115
231	232	GPIO 15 (TWO DART)	116
233	234	GPIO 16 (RED DART)	117
235	236	GPIO 17 (PCM_CLKIN)	118
237	238	Ground	119
239	240	GPIO 4	120
241	242	GPIO 5	121
243	244	Ground	122
245	246	GPIO 6	123
247	248	GPIO 10 (CE0 [SPI])	124
249	250	GPIO 11 (CE1 [SPI])	125
251	252	GPIO 12 (MOSI [SPI])	126
253	254	GPIO 13 (MISO [SPI])	127
255	256	GPIO 14 (SCLK [SPI])	128
257	258	Ground	129
259	260	GPIO 15 (TWO DART)	130
261	262	GPIO 16 (RED DART)	131
263	264	GPIO 17 (PCM_CLKIN)	132
265	266	Ground	133
267	268	GPIO 4	134
269	270	GPIO 5	135
271	272	Ground	136
273	274	GPIO 6	137
275	276	GPIO 10 (CE0 [SPI])	138
277	278	GPIO 11 (CE1 [SPI])	139
279	280	GPIO 12 (MOSI [SPI])	140
281	282	GPIO 13 (MISO [SPI])	141
283	284	GPIO 14 (SCLK [SPI])	142
285	286	Ground	143
287	288	GPIO 15 (TWO DART)	144
289	290	GPIO 16 (RED DART)	145
291	292	GPIO 17 (PCM_CLKIN)	146
293	294	Ground	147
295	296	GPIO 4	148
297	298	GPIO 5	149
299	300	Ground	150
301	302	GPIO 6	151
303	304	GPIO 10 (CE0 [SPI])	152
305	306	GPIO 11 (CE1 [SPI])	153
307	308	GPIO 12 (MOSI [SPI])	154
309	310	GPIO 13 (MISO [SPI])	155
311	312	GPIO 14 (SCLK [SPI])	156
313	314	Ground	157
315	316	GPIO 15 (TWO DART)	158
317	318	GPIO 16 (RED DART)	159
319	320	GPIO 17 (PCM_CLKIN)	160
321	322	Ground	161
323	324	GPIO 4	162
325	326	GPIO 5	163
327	328	Ground	164
329	330	GPIO 6	165
331	332	GPIO 10 (CE0 [SPI])	166
333	334	GPIO 11 (CE1 [SPI])	167
335	336	GPIO 12 (MOSI [SPI])	168
337	338	GPIO 13 (MISO [SPI])	169
339	340	GPIO 14 (SCLK [SPI])	170
341	342	Ground	171
343	344	GPIO 15 (TWO DART)	172
345	346	GPIO 16 (RED DART)	173
347	348	GPIO 17 (PCM_CLKIN)	174
349	350	Ground	175
351	352	GPIO 4	176
353	354	GPIO 5	177
355	356	Ground	178
357	358	GPIO 6	179
359	360	GPIO 10 (CE0 [SPI])	180
361	362	GPIO 11 (CE1 [SPI])	181
363	364	GPIO 12 (MOSI [SPI])	182
365	366	GPIO 13 (MISO [SPI])	183
367	368	GPIO 14 (SCLK [SPI])	184
369	370	Ground	185
371	372	GPIO 15 (TWO DART)	186
373	374	GPIO 16 (RED DART)	187
375	376	GPIO 17 (PCM_CLKIN)	188
377	378	Ground	189
379	380	GPIO 4	190
381	382	GPIO 5	191
383	384	Ground	192
385	386	GPIO 6	193
387	388	GPIO 10 (CE0 [SPI])	194
389	390	GPIO 11 (CE1 [SPI])	195
391	392	GPIO 12 (MOSI [SPI])	196
393	394	GPIO 13 (MISO [SPI])	197
395	396	GPIO 14 (SCLK [SPI])	198
397	398	Ground	199
399	400	GPIO 15 (TWO DART)	200

Figure 4 – Raspberry Pi Zero 2 W Pinout Diagram

## Validation Platform Connection and Power up

Figure 5 shows a complete connection and setup of the Validation Platform. User should follow these steps:

1. Connect SCL, SDA, and GND pins between RPI and AP33772 EVB
2. Connect 65W PD Charger and AP33772 EVB with Type-C cable
3. Power up RPI and PD Charger.

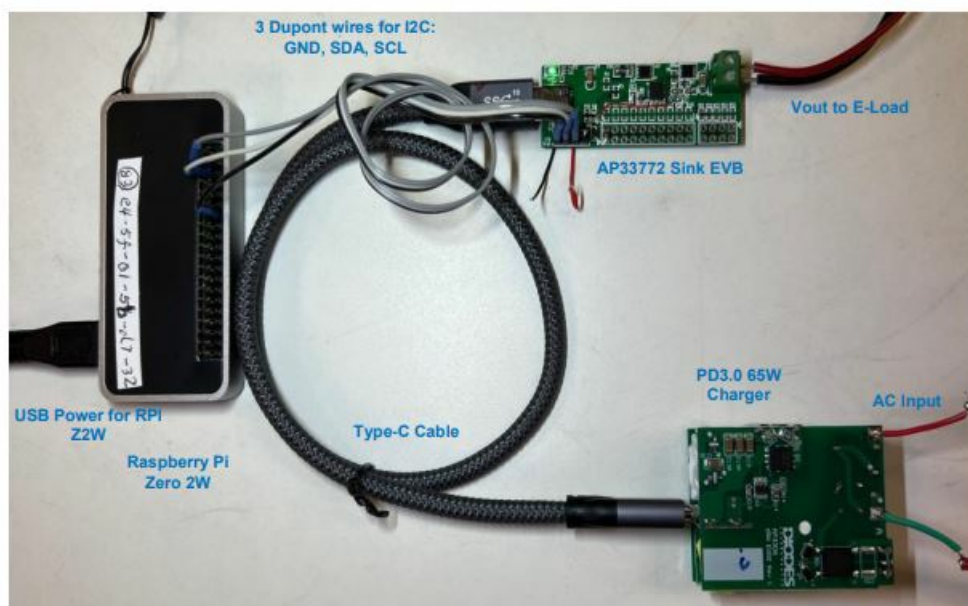


Figure 5 – Complete Setup of the Validation Platform

## Raspberry Pi Software Setup

### Raspberry Pi OS

- There are many different operating systems that support RPI. Among these, Raspberry Pi OS is chosen because it is the most used and recommended by RPI official site.

### Download OS Image and Prepare SD Card

- Download and install Raspberry Pi Imager tools on a PC (<https://www.raspberrypi.com/software/>). Follow the instruction to prepare a Micro-SD loaded with correct OS image (<https://youtu.be/ntaXWS8Lk34/>). Please note Micro-SD card of 32BG or greater is recommended.

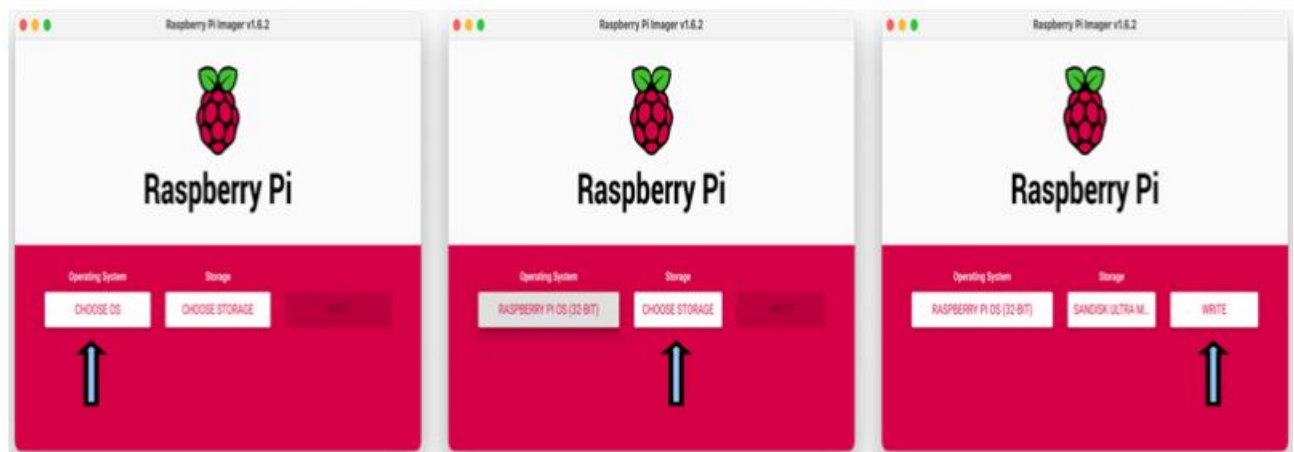


Figure 6 – Raspberry Pi Imager Tool for OS Image Download and Preparation

### Raspberry PI OS Installation

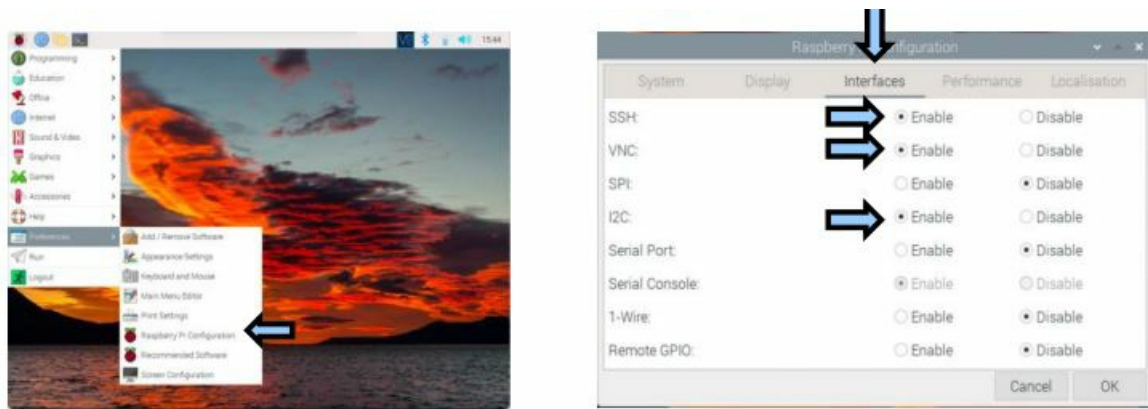
- Insert the Micro-SD card loaded with imager earlier into RPI's Micro-SD slot. Connect the power adapter, mouse/keyboard, and HDMI monitor. Power on the RPI and follow the instruction to complete OS installation and basic setup. Make sure the latest updates are included on the OS.

### Setup of Required Features

- In order to run I2C interface on RPI successfully, we must configure or install the SSH, VNC, and I2C features.

### Raspberry Pi Config – SSH, VNC, I2C

- After RPI boot-up, open “Raspberry Pi Configure” utility and turn on SSH, VNC, and I2C features.



**Figure 7 – Enable SSH, VNC, and I2C in Raspberry Pi Configuration Utility**

## I2C Baud Rate Configuration

- Replace the lines regarding dtparam and dtoverlay in /boot/config.txt file with:
- `dtoverlay=i2c-bcm2708`
- `dtparam=i2c_arm=on,i2c_arm_baudrate=640000`

## I2C-Tools Installation

- I2C-Tools is a toolset that provides simple commands running on command line under Raspberry Pi OS. Install I2C-Tools on the OS by running: `sudo apt install i2c-tools`

## SMBus2 Installation

- SMBus2 is a Python module that provides convenient functions for user to control I2C interface under Python environment. Install SMBus2 module for Python on the OS by running: `sudo pip3 install smbus2`

## Basic Command Examples

- This User Guide demonstrate two different methods to work with I2C interface on RPI. They are I2C-Tools Utility and Python SMBus2 Module. The basic commands of both methods are introduced in this section.

## I2C-Tools Command Examples

- I2C-Tools utility package provides `i2cdetect`, `i2cget`, and `i2cset` commands. Simplified usages are described in the examples under this section. For complete information about I2C-Tools utility, please refer to <https://linuxhint.com/i2c-linux-utilities/>.
- Table 1 shows the AP33772 register summary for user's convenience to digest the command usage in this section. For complete register information, please refer to AP33772 Sink Controller EVB User Guide.

Register	Command	Length	Attribute	Power-on	Description

SRCPDO	0x00	28	RO	All 00h	Power Data Object (PDO) used to expose PD Source (SRC) power capabilities.  Total length is 28 bytes
PDONUM	0x1C	1	RO	00h	Valid source PDO number
STATUS	0x1D	1	RC	00h	AP33772 status
MASK	0x1E	1	RW	01h	Interrupt enable mask
VOLTAGE	0x20	1	RO	00h	LSB 80mV
CURRENT	0x21	1	RO	00h	LSB 24mA
TEMP	0x22	1	RO	19h	Temperature, Unit: °C
OCPTHR	0x23	1	RW	00h	OCP threshold, LSB 50mA
OTPTHR	0x24	1	RW	78h	OTP threshold, Unit: °C
DRTHR	0x25	1	RW	78h	De-rating threshold, Unit: °C
TR25	0x28	2	RW	2710h	Thermal Resistance @25°C, Unit: Ω
TR50	0x2A	2	RW	1041h	Thermal Resistance @50°C, Unit: Ω
TR75	0x2C	2	RW	0788h	Thermal Resistance @75°C, Unit: Ω
TR100	0x2E	2	RW	03CEh	Thermal Resistance @100°C, Unit: Ω

RDO	0x30	4	WO	00000000h	Request Data Object (RDO) is use to request power capabilities.
VID	0x34	2	RW	0000h	Vendor ID, Reserved for future applications
PID	0x36	2	RW	0000h	Product ID, Reserved for future applications
RESERVED	0x38	4	–	–	Reserved for future applications

**Table 1 – AP33772 Register Summary**

#### **Detect all devices attached to I2C – i2cdetect**

- To display all i2c devices currently attached to I2C-1 bus, type the following under command prompt: `i2cdetect -y 1`
- If AP33772 Sink Controller EVB is attached, user should see device is attached at 0x51 address

#### **Read SRCPDO (0x00~0x1B)**

- `i2cget` command doesn't support block read longer than 2 bytes. User needs to use "for loop" to display all 28-byte long PDO data. To display all PDO data, type the following under bash command prompt for `i` in `{0..27}`; `do i2cget -y 1 0x51 $i b; done`
- 28-byte data representing 7 PDOs will be displayed

#### **Read PDONUM (0x1C)**

- To display total number of valid PDOs, type the following under command prompt: `i2cget -y 1 0x51 0x1c b`

#### **Read STATUS (0x1D)**

- This command reports the Sink Controller's status including de-rating, OTP, OCP, OVP, Request Rejected, Request Completed, and Ready. To display the status information, type the following under command prompt: `i2cget -y 1 0x51 0x1d b`
- User should use this command after each RDO request to ensure successful RDO request by reading the COMPLETE bit.
- 4.1.5 Write MASK (0x1E)
- This command enables the interrupts that signal the host through GPIO3 pin of AP33772. The interrupts include Derating, OTP, OCP, OVP, Request Rejected, Request Completed, and Ready. To enable a specific interrupt, set the corresponding bit to one. For example, to enable OCP interrupt, set bit 4 of MASK register to one by typing the following under command prompt: `i2cset -y 1 0x51 0x1e 0x10 b`

- GPIO3 pin of AP33772 will go high when the OCP protection is trigger.

### Read VOLTAGE (0x20)

- This command reports the voltage measured by the AP33772 Sink Controller. To report the voltage, type the following under command prompt: `i2cget -y 1 0x51 0x20 b`
- One unit of the reported value represents 80mV.

### Read CURRENT (0x21)

- This command reports the current measured by the AP33772 Sink Controller. To report the current, type the following under command prompt: `i2cget -y 1 0x51 0x21 b`
- One unit of the reported value represents 24mA.

### Read TEMP (0x22)

- This command reports the temperature measured by the AP33772 Sink Controller. To report the temperature, type the following under command prompt:  
`i2cget -y 1 0x51 0x22 b`
- One unit of the reported value represents 1°C.
- Read and Write OCPTHR (0x23), OTPTHR (0x24), and DRTHR (0x25)
- OCP, OTP, and Derating thresholds can be changed to user desirable values by writing the values to OCPTHR, OTPTHR, and DRTHR registers. As an example, to change OCP threshold to 3.1A, user should write 0x3E (=3100/50=62=0x3E) to OCPTHR by typing the following under command prompt: `i2cset -y 1 0x51 0x23 0x3e b`
- To change OTP threshold to 110°C, user should write 0x6E (=110) to OTPTHR by typing the following under command prompt:
- To read the values out of OCPTHR, OTPTHR, and DRTHR, type the following under command prompt:  
`i2cget -y 1 0x51 0x23 b i2cget -y 1 0x51 0x24 b i2cget -y 1 0x51 0x25 b`
- **Read and Write TR25 (0x28~0x29), TR50 (0x2A~0x2B), TR75 (0x2C~0x2D), and TR100 (0x2E~0x2F)**
- A Murata 10KΩ Negative Temperature Coefficient (NTC) Thermistor NCP03XH103 is populated on the AP33772 EVB. It is user's preference to change the thermistor to a different one in the final design. User should update TR25, TR50, TR75, and TR100 register values according to specifications of the thermistor used. For example,
- Murata's 6.8KΩ NCP03XH682 is used in the design. The resistance values at 25°C, 50°C, 75°C, and 100°C are 6800Ω (0x1A90), 2774Ω (0x0AD6), 1287Ω (0x0507), and 662Ω (0x0296) respectively. To write the corresponding values to these registers, type the following under command prompt:  
`i2cset -y 1 0x51 0x28 0x1a90 w i2cset -y 1 0x51 0x2a 0x0ad6 w i2cset -y 1 0x51 0x2c 0x0507 w i2cset -y 1 0x51 0x2e 0x0296 w`
- To read the values out, type the following under command prompt: `i2cget -y 1 0x51 0x28 w i2cget -y 1 0x51 0x2a w i2cget -y 1 0x51 0x2c w i2cget -y 1 0x51 0x2e w`
- The output values are 2-byte words. Since the commands handle 2-byte word directly, users don't need to worry about little endian byte order here.

## Write RDO (0x30~0x33)

- To initiate a PDO request negotiation procedure, 4-byte data is written to RDO (Request Data Object) register in little-endian byte order. As example, to request PDO3 with 15V and 3A, 0x3004B12C will be written to RDO register. Type the following under command prompt: `i2cset -y 1 0x51 0x30 0x2c 0xb1 0x04 0x30 i`
- The least significant byte (0x2C) should be written in first to fit little endian byte order notation. Please refer to Table 9 and Table 10 of AP33772 Sink Controller EVB User
- Guide for detailed RDO content information.
- User can issue a hard reset by writing RDO register with all-zero data: `i2cset -y 1 0x51 0x30 0x00 0x00 0x00 0x00 i`
- The AP33772 Sink Controller will be reset to its initial state and output will be turned off.

## Python SMBus2 Command Examples

- Python is getting more popular for its great varieties of supported modules. SMBus2 is among of those and capable of handling I2C read and write commands. SMBus2 provides `read_byte_data`, `read_word_data`, `read_i2c_block_data`, `write_byte_data`, `write_word_data`, `write_i2c_block_data` commands. Simplified usages are described in the examples under this section. For complete information about SMBus2 module, please refer to <https://smbus2.readthedocs.io/en/latest/>.

## Read SRCPDO (0x00~0x1B)

- `SMBus.read_i2c_block_data` is an effective command to support up to 32-byte block data read. To read all 28-byte PDO data, use the following under python3 environment:
- `SMBus.read_i2c_block_data(0x51, 0x00, 28)`
- 28 one-byte data representing 7 PDOs will be returned in list data structure.

## Read PDONUM (0x1C)

- To read total number of valid PDOs, use the following under python3 environment:
- `SMBus.read_byte_data(0x51, 0x1c)`
- One byte data representing valid PDO count will be returned.

## Read STATUS (0x1D)

- This command reports the Sink Controller's status including Derating , OTP, OCP, OVP, Request Rejected, Request Completed, and Ready. To read the status information, use the following under python3 environment:
- `SMBus.read_byte_data(0x51, 0x1d)`
- User may use this command after each RDO request to ensure successful RDO request by reading the COMPLETE bit.

## Write MASK (0x1E)

- This command enables the interrupts that signal the host through GPIO3 pin of AP33772. The interrupts

include Derating , OTP, OCP, OVP, Request Rejected, Request

- Completed, and Ready. To enable a specific interrupt, set the corresponding bit to one. For example, to enable OCP interrupt, set bit 4 of MASK register to one by using the following under python3 environment:
- `SMBus.write_byte_data(0x51, 0x1e, 0x10)`
- GPIO3 pin of AP33772 will go high when the OCP protection is trigger.

### **Read VOLTAGE (0x20)**

- This command reports the voltage measured by the AP33772 Sink Controller. To report the voltage, use the following under python3 environment:
- `SMBus.read_byte_data(0x51, 0x20)`
- One unit of the reported value represents 80mV.

### **Read CURRENT (0x21)**

- This command reports the current measured by the AP33772 Sink Controller. To report the current, use the following under python3 environment
- `SMBus.read_byte_data(0x51, 0x21)`
- One unit of the reported value represents 24mA.

### **Read TEMP (0x22)**

- This command reports the temperature measured by the AP33772 Sink Controller. To report the temperature, use the following under python3 environment:
- `SMBus.read_byte_data(0x51, 0x22)`
- One unit of the reported value represents 1°C.

### **Read and Write OCPTHR (0x23), OTPTHR (0x24), and DRTHR (0x25)**

- OCP, OTP, and Derating thresholds can be changed to user desirable values by writing the values to OCPTHR, OTPTHR, and DRTHR registers. As an example, to change OCP threshold to 3.1A, user should write 0x3E (=3100/50=62=0x3E) to OCPTHR by using the following under python3 environment:  
`SMBus.write_byte_data(0x51, 0x23, 0x3e)`
- To change OTP threshold to 110°C, user should write 0x6E (=110) to OTPTHR by using the following under python3 environment: `SMBus.write_byte_data(0x51, 0x24, 0x6e)`
- To change Derating threshold to 100°C, user should write 0x64 (=100) to DRTHR by using the following under python3 environment: `SMBus.write_byte_data(0x51, 0x25, 0x64)`
- **To read the values out of OCPTHR, OTPTHR, and DRTHR, use the following under python3 environment:** `SMBus.read_byte_data(0x51, 0x23)` `SMBus.read_byte_data(0x51, 0x24)` `SMBus.read_byte_data(0x51, 0x25)` .
- Read and Write TR25 (0x28~0x29), TR50 (0x2A~0x2B), TR75 (0x2C~0x2D), and TR100 (0x2E~0x2F)
- A Murata 10KΩ Negative Temperature Coefficient (NTC) Thermistor NCP03XH103 is populated on the AP33772 EVB. It is user's preference to change the thermistor to a different one in the final design. User should update TR25, TR50, TR75, and TR100 register value according to specifications of the thermistor used.

For example, Murata's 6.8K $\Omega$  NCP03XH682 is used in the design. The resistance values at 25°C, 50°C, 75°C, and 100°C are 6800 $\Omega$  (0x1A90), 2774 $\Omega$  (0x0AD6), 1287 $\Omega$  (0x0507), and 662 $\Omega$  (0x0296) respectively. To write the corresponding values to these registers, use the following under python3 environment:

- `SMBus.write_word_data(0x51, 0x28, 0x1a90)` `SMBus.write_word_data(0x51, 0x2a, 0x0ad6)`  
`SMBus.write_word_data(0x51, 0x2c, 0x0507)` `SMBus.write_word_data(0x51, 0x2e, 0x0296)`
- To read the values out, use the following under python3 environment: `SMBus.read_word_data(0x51, 0x28)`  
`SMBus.read_word_data(0x51, 0x2a)` `SMBus.read_word_data(0x51, 0x2c)` `SMBus.read_word_data(0x51, 0x2e)`
- The return values are also 2-byte words. Since the commands handle 2-byte word directly, users don't need to worry about little endian byte order here.

### Write RDO (0x30~0x33)

- To initiate a PDO request negotiation procedure, 4-byte data is written to RDO (Request Data Object) register in little-endian byte order. As example, to request PDO3 with 15V and 3A, 0x3004B12C will be written to RDO register. Use the following under python3 environment:
- `SMBus.write_i2c_block_data(0x51, 0x30, [0x2c, 0xb1, 0x04, 0x30])`
- Please refer to Table 9 and Table 10 of AP33772 Sink Controller EVB User Guide for detailed RDO content information.
- User can issue a hard reset by writing RDO register with all-zero data:
- `SMBus.write_i2c_block_data(0x51, 0x30, [0x00, 0x00, 0x00, 0x00])`
- The AP33772 Sink Controller will be reset to its initial state and output will be turned off.

### Practical Examples

#### **Example 1:** Bash I2C-Tools Example: `ap33772_querypdo.bash`

This example checks all valid PDOs and lists the voltage and current capability information out.

#### **Code Details**

```
#!/bin/bash

# This program reports all PDO information

RPI_I2CBUS=1 # Using Raspberry Pi I2C_1
I2C_ADDR=0x51 # I2C address 0x51
PDO_ADDR=0x00 # PDO address range 0x00 ~ 0x1b, Starting at 0x00, max is 7 PDOs
ValidPDOCnt=0 # reset Valid PDO count

for i in {0..6}
do
    # Read PDO info 4-byte or 32-bit long each
    PDO=$(( `i2cget -y $RPI_I2CBUS $I2C_ADDR $((PDO_ADDR+4*$i+3))` << 24)
    | `i2cget -y $RPI_I2CBUS $I2C_ADDR $((PDO_ADDR+4*$i+2))` << 16)
    | `i2cget -y $RPI_I2CBUS $I2C_ADDR $((PDO_ADDR+4*$i+1))` << 8)
    | `i2cget -y $RPI_I2CBUS $I2C_ADDR $((PDO_ADDR+4*$i))` ))

    # If PDO reads all zero data, it's not a valid PDO. Only processing valide PDO
    IS_VALID_PDO=$(( $PDO != 0x00000000 ))
    if [ $IS_VALID_PDO == 1 ]
    then
        # Check if this is regular PDO or APDO, Bit31..30==11 is APDO, Bit31..30==00 is PDO
        IS_APDO=$(( ($PDO & 0xc0000000) == 0xc0000000 ))
        if [ $IS_APDO != 1 ]
        then
            # Print out Fixed PDO information
            printf "PDO ID:%d\nPDO=0x%.8x is a %s\n" $((i+1)) $PDO "Fixed PDO"

            # Find out profiles for Fixed PDO, refer to Table 5 of AP33772 Sink Controller EVB User Guide Section 5.3
            MaxCurr=$(( (($PDO & (0x3ff<<0))>>0) * 10 )) # bit 9..0, 1LSB is 10mA
            Volt=$(( (($PDO & (0x3ff<<10))>>10) * 50 )) # bit 19..10, 1LSB is 50mV
            echo Voltage=$($Volt)mV
            echo Max Current=$($MaxCurr)mA
            echo
        else
            # Print out APDO information
            printf "PDO ID:%d\nPDO=0x%.8x is a %s\n" $((i+1)) $PDO "APDO"

            # Find out profiles for APDO, refer to Table 6 of AP33772 Sink Controller EVB User Guide Section 5.3
            MaxCurr=$(( (($PDO & (0x3ff<<0))>>0) * 50 )) # bit 6..0, 1LSB is 50mA
            MinVolt=$(( (($PDO & (0xff<<8))>>8) * 100 )) # bit 15..8, 1LSB is 100mV
            MaxVolt=$(( (($PDO & (0xff<<17))>>17) * 100 )) # bit 24..17, 1LSB is 100mV
            echo Min Voltage=$($MinVolt)mV
            echo Max Voltage=$($MaxVolt)mV
            echo Max Current=$($MaxCurr)mA
            echo
        fi
        ValidPDOCnt=$((ValidPDOCnt + 1))
    fi
done

echo Total $ValidPDOCnt valid PDOs are detected!
```

## Code Execution and Outputs

```
pi@raspberrypi:~/Project/ap33772/bash $ ./ap33772_querypdo.bash
PDO ID:1
PDO=0x0a01912c is a Fixed PDO
Voltage=5000mV
Max Current=3000mA

PDO ID:2
PDO=0x0002d12c is a Fixed PDO
Voltage=9000mV
Max Current=3000mA

PDO ID:3
PDO=0x0004b12c is a Fixed PDO
Voltage=15000mV
Max Current=3000mA

PDO ID:4
PDO=0x00064145 is a Fixed PDO
Voltage=20000mV
Max Current=3250mA

PDO ID:5
PDO=0xc1a4213c is a APDO
Min Voltage=3300mV
Max Voltage=21000mV
Max Current=3000mA

Total 5 valid PDOs are detected!
```

### Example 2: Python SMBus2 Example: ap33772\_allpdo.py3

This example checks all valid PDOs and requests them one by one in up and down order.

### Code Details

```
#!/usr/bin/env python3
```

```
# This program reports all PDO information and walks through all PDOs in up and down manner
```

```
from smbus2 import SMBus
from time import sleep
```

Import SMBus

```
RPI_I2CBUS=1 # Using Raspberry Pi I2C_1
I2C_ADDR=0x51 # I2C address 0x51
```

AP33772 has I2C address 0x51

```
class Pdo:
```

```
    def __init__(self, word=0x00000000, pdotype="FPDO", id=0):
        self.word=word # PDO's word contents
        self.pdtype=pdotype # "FPDO" or "APDO"
        self.id=id # PDO id/position
        if self.pdtype != "APDO": # FPDO
            self.Volt=5000
            self.MaxCurr=3000
        else: # APDO
            self.MaxVolt=21000
            self.MinVolt=3300
            self.MaxCurr=3000

    def display(self):
        print("PDO%d: 0x%.8x %s" % (self.id, self.word, self.pdtype))
```

Class definition for PDO:

Common attributes: word, pdotype, id  
Fixed PDO attributes: Voltage and Max Current  
APDO attributes: Max/Min Voltage and Max Current  
Common function: display()

```
class Rdo:
```

```
    def __init__(self, word=0x00000000, pdotype="FPDO", id=0):
        self.word=word # RDO's word contents
        self.pdtype=pdotype # "FPDO" or "APDO"
        self.id=id # RDO id/position
        if self.pdtype != "APDO": # FPDO
            self.RpoOpCurr=3000
            self.RpoMaxOpCurr=3000
        else: # APDO
            self.RpoOpVolt=5000
            self.RpoOpCurr=3000

    def display(self):
        print("RDO%d: 0x%.8x %s" % (self.id, self.word, self.pdtype))
```

Class definition for RDO:

Common attributes: word, pdotype, id  
Fixed PDO attributes: Op Current and Max Current  
APDO attributes: Op Voltage and Op Current  
Common function: display()

```
try:
```

```
    # Create i2c object
    i2c=SMBus(RPI_I2CBUS)
    # Dummy write command to flush out unfinished I2C traffic
    #i2c.write_i2c_block_data(I2C_ADDR, 0x30, [0x2c, 0xb1, 0x04, 0x10])
    # Read all 28-Byte PDO information
    pdo28b=i2c.read_i2c_block_data(I2C_ADDR, 0x00, 28)
```

Create SMBus object called i2c

Read all 28-Byte 7 PDOs info at one time  
0x00~0x1b

```
    # Build PDO objects based on the first 28-byte data
    pdolist=list()
    ValidPDONum=0 # reset Valid PDO count
    for i in range(0, len(pdo28b), 4):
```

```
        p = Pdo()
        pdolist.append(p)
        p.id = int(i/4) + 1
        p.word = 0
        for j in range(4):
            p.word=(p.word+(pdo28b[i+j]<<(8*j)))
```

Analyze 28-Byte data and Create up to 7 PDO  
objects. Each PDO type, word contents, Voltage,  
and Current capability

```
    # Process only valid PDO which has contents other than 0x00000000
    IS_VALID_PDO=(p.word != 0x00000000)
    #IS_VALID_PDO=
```

```

# Process only valid PDO which has contents other than 0x00000000
IS_VALID_PDO=(p.word != 0x00000000)
if IS_VALID_PDO:
    #print("PDO ID:%d 0x%.8x" %(p.id, p))
    ValidPDOCnt+=1

    IS_APDO=((p.word & 0xc0000000)==0xc0000000)    # APDO bit 31..30 is 0b11
    if IS_APDO:
        print("PDO ID:%d\nPDO=0x%.8x is a APDO" %(p.id, p.word))
        p.pdtype="APDO"
        p.MaxCurr=((p.word&(0x3f<<0))>>0)*50        # bit 6..0, 1LSB is 50mA
        p.MinVolt=((p.word&(0xff<<8))>>8)*100        # bit 15..8, 1LSB is 100mV
        p.MaxVolt=((p.word&(0xff<<17))>>17)*100      # bit 24..17, 1LSB is 100mV
        print("MinVoltage=%dmV\nMaxVoltage=%dmV\nMax Current=%dmA\n" %(p.MinVolt, p.MaxVolt,
p.MaxCurr))
    else:
        print("PDO ID:%d\nPDO=0x%.8x is a Fixed PDO" %(p.id, p.word))
        p.pdtype="FPDO"
        p.MaxCurr=((p.word&(0x3ff<<0))>>0)*10        # bit 9..0, 1LSB is 10mA
        p.Volt=((p.word&(0x3ff<<10))>>10)*50         # bit 19..10, 1LSB is 50mV
        print("Voltage=%dmV\nMax Current=%dmA\n" %(p.Volt, p.MaxCurr))

print("Total %d valid PDOs are detected!" %(ValidPDOCnt))
sleep(1.0)

# Delete unused PDOs
for i in range(ValidPDOCnt, 7):
    pdolist.pop(-1)

# Print all PDO out
print("PDO List:")
for p in pdolist:
    p.display()

sleep(1.0)

# Preparing RDO for later request
rdolist=list()
for p in pdolist:
    r=Rdo()
    rdolist.append(r)
    r.id=p.id
    r.pdtype=p.pdtype
    if p.pdtype == "FPDO": # This is Fixed PDO
        r.RpoOpCurr=p.MaxCurr
        r.RpoMaxOpCurr=p.MaxCurr
        # Set position value bit30..28
        # Set Operating Current in 10mA units, bit19..10
        # Set Max Operating Current in 10mA units, bit9..0
        r.word = ((r.id & 0x7) << 28) | (int(r.RpoOpCurr/10)<<10) | (int(r.RpoMaxOpCurr/10)<<0)
    else: # This is APDO
        r.RpoOpVolt=p.MaxVolt
        r.RpoOpCurr=p.MaxCurr
        # Set position value bit30..28
        # Set Output Voltage in 20mV units, bit19..9
        # Set Operating Current in 50mA units, bit6..0
        r.word = ((r.id & 0x7) << 28) | (int(r.RpoOpVolt/20)<<9) | (int(r.RpoMaxOpCurr/50)<<0)

```

**APDO bit 31..30 is 0b11**  
Fixed PDO bit 31..30 is

**APDO:**  
Max Current  
Min Voltage  
Max Voltage

**Fixed PDO:**  
Max Current  
Voltage

**Print out valid PDO**

**Clean up invalid PDOs**  
and keep only valid PDOs

**Print all PDO information out**

**Analyze all valid PDO objects and create RDO**  
objects corresponding to each PDO  
with id, PDO type, word contents, Voltage,  
and Current to request later

**Fixed PDO set:**  
Op Current  
Max Op Current

**APDO set:**  
Op Current  
Out Voltage  
Will be changed later.

```

# Print all RDO out
print("RDO List:")
for p in rdolist:
    p.display()

print("#####")
print("Start requesting PDOs")
print("")

# RDO submission
cmdcnt=0
rejcnt=0
while True:
    for i in list(range(ValidPDOCnt-1))+list(reversed(range(1, ValidPDOCnt))):
        cmdcnt=cmdcnt+1
        # Request PDO by writing to 0x30~0x33
        i2c.write_i2c_block_data(I2C_ADDR, 0x30, [(rdolist[i].word>>0)&0xff, (rdolist[i].word>>8)&0xff,
(rdolist[i].word>>16)&0xff, (rdolist[i].word>>24)&0xff])
        sleep(0.5)
        status = i2c.read_byte_data(I2C_ADDR, 0x1d)
        if (status & 0x02) != 0x02:
            rejcnt=rejcnt+1
            voltage = i2c.read_byte_data(I2C_ADDR, 0x20) * 80      # 80mV per LSB
            current = i2c.read_byte_data(I2C_ADDR, 0x21) * 24     # 24mA per LSB
            temperature = i2c.read_byte_data(I2C_ADDR, 0x22)
            print("PDO%d:Total:%-3d\tstatus:0x%.2x\tRejects=%d\tV=%dmV\tI=%dmA\tT=%dC" %((i+1), cmdcnt, status, rejcnt,
voltage, current, temperature))

            #sleep(0.3)

# The following command will never be reached due to "while True" command! Actually object closure is done in except condition.
i2c.close()

except KeyboardInterrupt:
    # If there is a KeyboardInterrupt (when you press ctrl+c), exit the program and cleanup
    print("Break detected!")
    if i2c:
        i2c.close()

```

Print all PDO information out

Index runs up and down all PDO until

Send out the chosen RDO and check if status bit is set for successful accept

Read and report voltage, current, and temperature

Exception handling for graceful program closure

## Code Execution and Outputs

```

pi@raspberrypi:~/Project/ap33772/py3 $ ./ap33772_allpdo.py3
PDO ID:1
PDO=0x0a01912c is a Fixed PDO
Voltage=5000mV
Max Current=3000mA

PDO ID:2
PDO=0x0002d12c is a Fixed PDO
Voltage=9000mV
Max Current=3000mA

PDO ID:3
PDO=0x0004b12c is a Fixed PDO
Voltage=15000mV
Max Current=3000mA

PDO ID:4
PDO=0x00064145 is a Fixed PDO
Voltage=20000mV
Max Current=3250mA

PDO ID:5
PDO=0xc1a4213c is a APDO
MinVoltage=3300mV
MaxVoltage=21000mV
Max Current=3000mA

Total 5 valid PDOs are detected!
PDO List:
PDO1: 0x0a01912c FPDO
PDO2: 0x0002d12c FPDO
PDO3: 0x0004b12c FPDO
PDO4: 0x00064145 FPDO
PDO5: 0xc1a4213c APDO
RDO List:
RDO1: 0x1004b12c FPDO
RDO2: 0x2004b12c FPDO
RDO3: 0x3004b12c FPDO
RDO4: 0x40051545 FPDO
RDO5: 0x5008343c APDO
#####
Start requesting PDOs

PDO1: Total:1      status:0x03  Rejects=0  V=5040mV  I=48mA  T=25C
PDO2: Total:2      status:0x02  Rejects=0  V=9040mV  I=72mA  T=25C
PDO3: Total:3      status:0x02  Rejects=0  V=15200mV I=96mA  T=25C
PDO4: Total:4      status:0x02  Rejects=0  V=20160mV I=120mA T=25C
PDO5: Total:5      status:0x02  Rejects=0  V=20400mV I=120mA T=25C
PDO4: Total:6      status:0x02  Rejects=0  V=20160mV I=120mA T=25C
PDO3: Total:7      status:0x02  Rejects=0  V=15200mV I=96mA  T=25C
PDO2: Total:8      status:0x02  Rejects=0  V=9040mV  I=72mA  T=25C
PDO1: Total:9      status:0x02  Rejects=0  V=5040mV  I=48mA  T=25C
PDO2: Total:10     status:0x02  Rejects=0  V=9040mV  I=72mA  T=25C
PDO3: Total:11     status:0x02  Rejects=0  V=15200mV I=96mA  T=25C
PDO4: Total:12     status:0x02  Rejects=0  V=20400mV I=120mA T=25C
PDO5: Total:13     status:0x02  Rejects=0  V=20400mV I=120mA T=25C
PDO4: Total:14     status:0x02  Rejects=0  V=20160mV I=120mA T=25C
PDO3: Total:15     status:0x02  Rejects=0  V=15200mV I=96mA  T=25C
^CBreak detected!

```

All PDOs' Capabilities

All PDOs and RDOs  
32-bit Contents

Send RDO and Report  
Status, Voltage, Current, Temperature

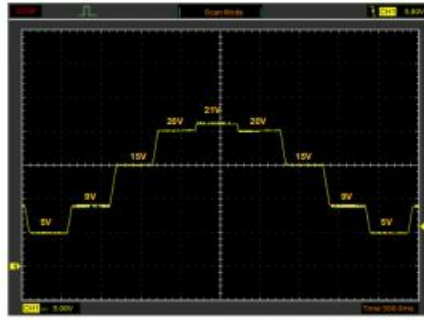


Figure 8 – Example ap33772\_allpdo.py Output Waveform

## Example Code Download

### List of Example Codes

- Example Codes have Bash Script and Python Versions
1. ap33772\_querypdo: queries all PDO information
  2. ap33772\_reqpdo: reports all PDO information and sends out PDO request specified by user
  3. ap33772\_allpdo: reports all PDO information and walks through all PDO requests in up and down manner
  4. ap33772\_pps: reports all PDO information, and ramps up and down the entire PPS voltage range in 50mV step size
  5. ap33772\_vit: reports voltage, current, and temperature information

### Example Download Site

**Example** Codes can be downloaded from Github. Issue the following command to download: git clone <https://github.com/diodinciot/ap33772.git>

### References

1. AP33772 Datasheet (USB PD3.0 PPS Sink Controller): <https://www.diodes.com/products/power-management/ac-dc-converters/usb-pd-sink-controllers/>
2. AP33772 I2C Sink Controller EVB User Guide: <https://www.diodes.com/applications/ac-dc-chargers-and-adapters/usb-pd-sink-controller/>
3. Raspberry Pi Zero 2 W: <https://www.raspberrypi.com/products/raspberry-pi-zero-2-w/>
4. Raspberry Pi OS: <https://www.raspberrypi.com/software/>
5. I2C-Tools utility: <https://linuxhint.com/i2c-linux-utilities/>
6. SMBus2 Module: <https://smbus2.readthedocs.io/en/latest/>

### Revision History

Revision	Issue Date	Comment	Author
1.0	4/15/2022	Initial Release	Edward Zhao

## IMPORTANT NOTICE

- DIODES INCORPORATED MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION).
- Diodes Incorporated and its subsidiaries reserve the right to make modifications, enhancements, improvements, corrections, or other changes without further notice to this document and any product described herein. Diodes Incorporated does not assume any liability arising out of the application or use of this document or any product described herein; neither does Diodes Incorporated convey any license under its patent or trademark rights, nor the rights of others. Any Customer or user of this document or products described herein in such applications shall assume all risks of such use and will agree to hold Diodes Incorporated and all the companies whose products are represented on the Diodes Incorporated website, harmless against all damages.
- Diodes Incorporated does not warrant or accept any liability whatsoever in respect of any products purchased through unauthorized sales channels.  
Should Customers purchase or use Diodes Incorporated products for any unintended or unauthorized application, Customers shall indemnify and hold Diodes Incorporated and its representatives harmless against all claims, damages, expenses, and attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized application.
- Products described herein may be covered by one or more United States, international or foreign patents pending. Product names and markings noted herein may also be covered by one or more United States, international or foreign trademarks.
- This document is written in English but may be translated into multiple languages for reference. Only the English version of this document is the final and determinative format released by Diodes Incorporated.

## LIFE SUPPORT


- Diodes Incorporated products are specifically not authorized for use as critical components in life support devices or systems without the express written approval of the Chief Executive Officer of Diodes Incorporated. As used herein:
- **A.** Life support devices or systems are devices or systems which:
  1. are intended to implant into the body, or
  2. support or sustain life and whose failure to perform when properly used in accordance with instructions for use provided in the labeling can be reasonably expected to result in significant injury to the user.
- **B.** A critical component is any component in a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or to affect its safety or effectiveness.
- Customers represent that they have all necessary expertise in the safety and regulatory ramifications of their life support devices or systems, and acknowledge and agree that they are solely responsible for all legal, regulatory, and safety-related requirements concerning their products and any use of Diodes Incorporated products in such safety-critical, life support devices or systems, notwithstanding any

devices- or systems-related information or support that may be provided by Diodes Incorporated.

- Further, Customers must fully indemnify Diodes Incorporated and its representatives against any damages arising out of the use of Diodes Incorporated products in such safety-critical, life-support devices or systems.

- Copyright © 2017, Diodes Incorporated
- [www.diodes.com](http://www.diodes.com)

## Documents / Resources

	<p><b><a href="#">DIODES AP33772 USB PD Sink Controller Raspberry Pi I2C Interface</a></b> [pdf] User Guide AP33772 USB PD Sink Controller Raspberry Pi I2C Interface, AP33772, USB PD Sink Controller Raspberry Pi I2C Interface, Raspberry Pi I2C Interface, Pi I2C Interface</p>
---	---

## References

-  [Self.ID](#)
-  [Diodes Incorporated - Analog, Discrete, Logic, Mixed-Signal](#)
-  [Diodes Incorporated - Analog, Discrete, Logic, Mixed-Signal](#)
-  [GitHub - diodinciot/ap33772: Using RPI to control ap33772 sink controller through I2C](#)
-  [I2C Utilities in Linux](#)
-  [smbus2 — smbus2 0.4.2 documentation](#)
-  [USB PD Sink Controller](#)
-  [USB PD Sink Controller](#)
-  [Buy a Raspberry Pi Zero 2 W – Raspberry Pi](#)
-  [Raspberry Pi OS – Raspberry Pi](#)