Table of Contents

Chapter 1 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)

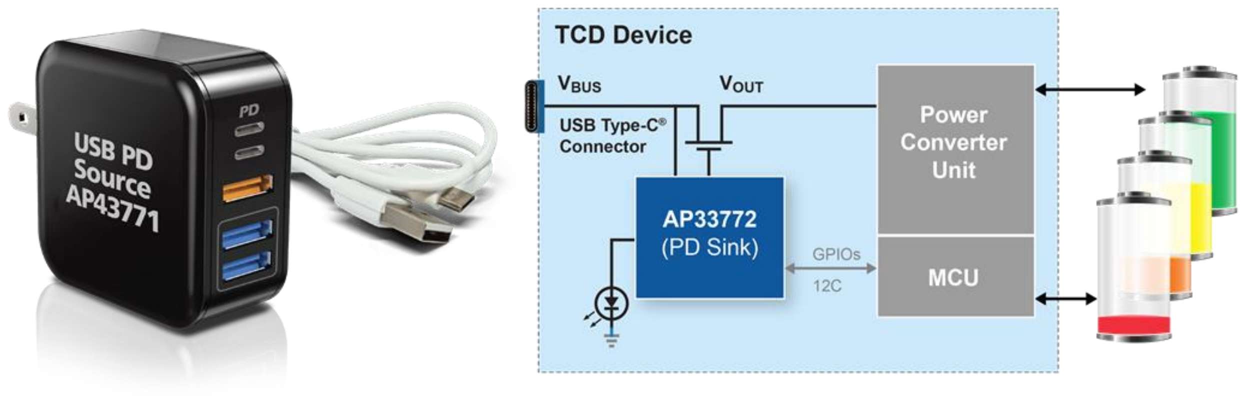

Figure 1 – A typical TCD uses AP33772 PD Sink Controller with I2C Interface to request power from an USB Type-C PD3.0/PPS Compliance Source Adapter

Chapter 2 Validation Platform Setup

2.1 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.

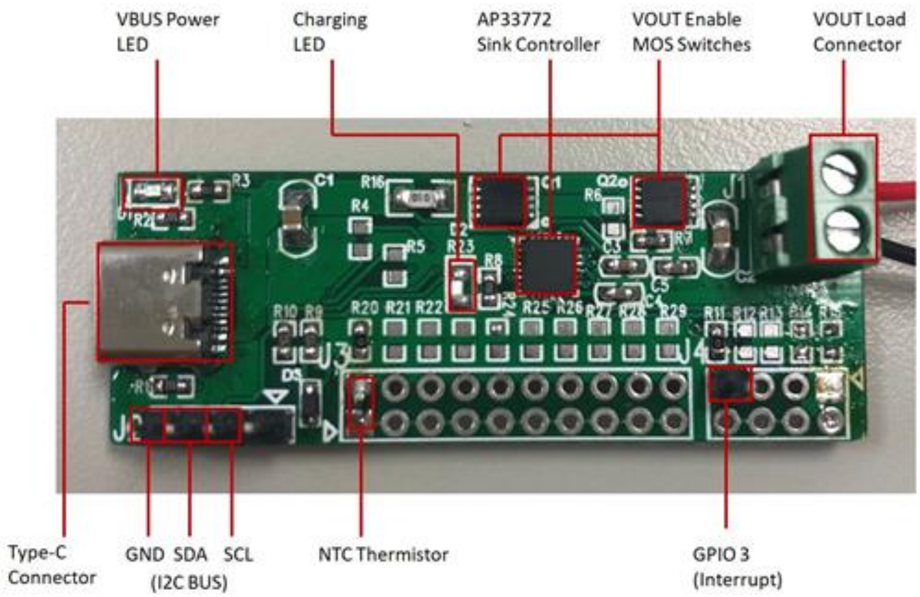

Figure 2 – AP33772 Sink Controller EVB

2.2 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 servers 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/)

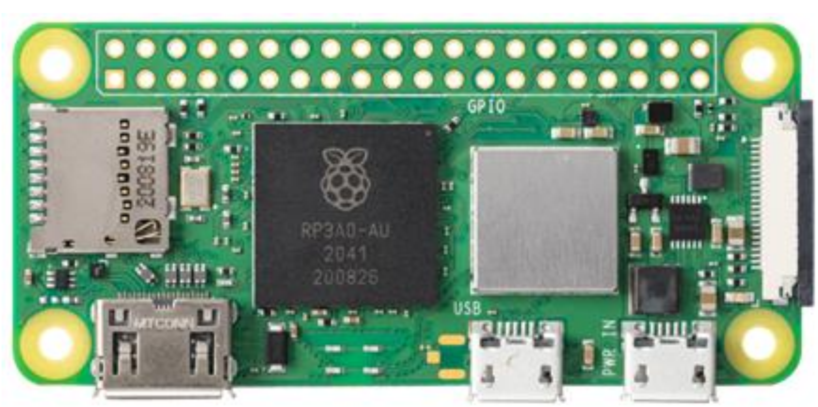

Figure 3 – Raspberry Pi Zero 2 W (RPI Z2W)

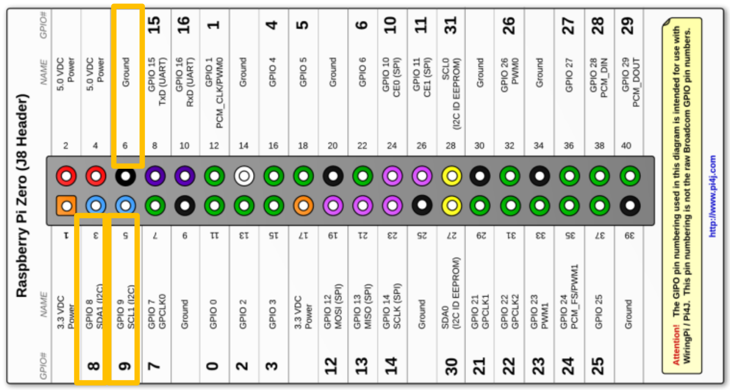

Figure 4 – Raspberry Pi Zero 2 W Pinout Diagram

2.3 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.**

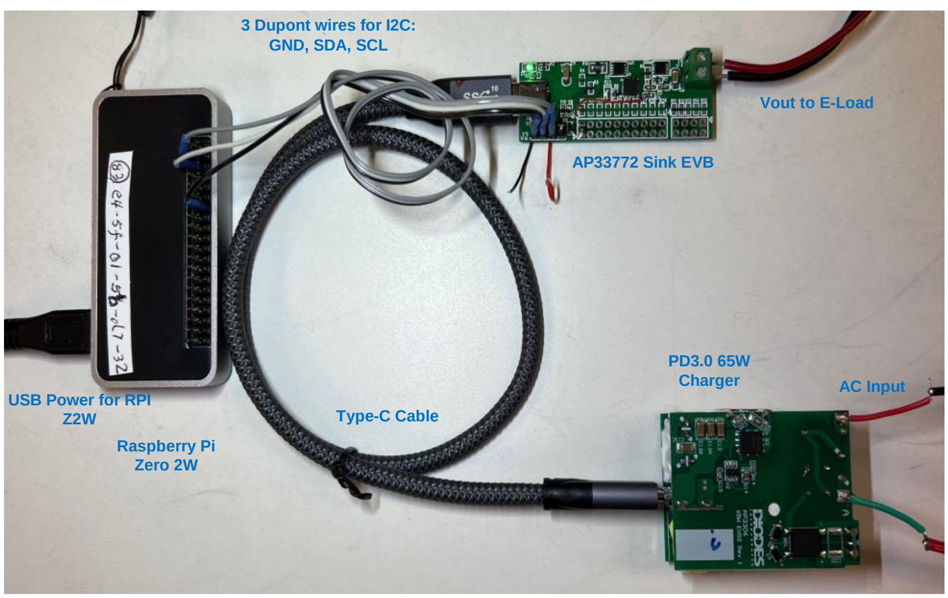

Figure 5 – Complete Setup of the Validation Platform

Chapter 3 Raspberry Pi Software Setup

3.1 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.

3.1.1 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.

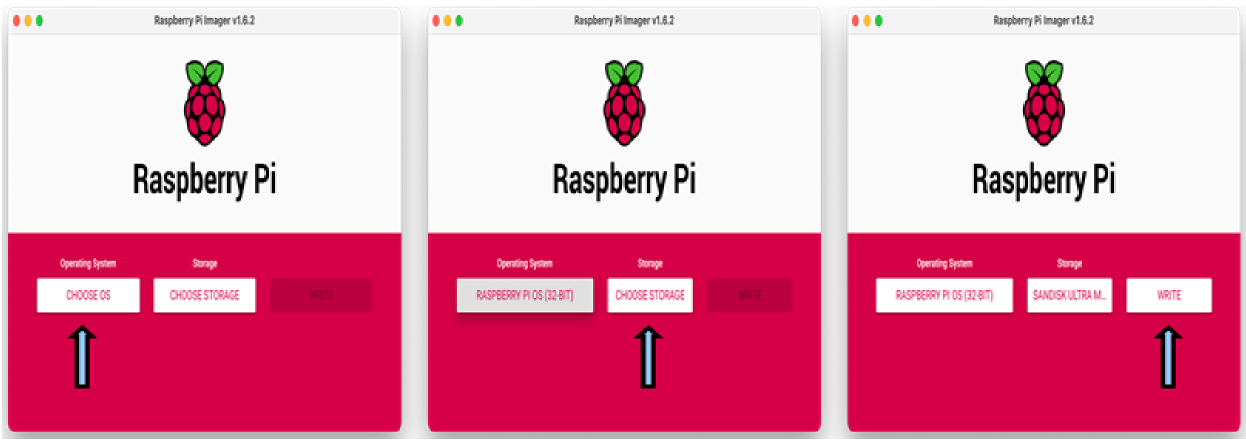

Figure 6 – Raspberry Pi Imager Tool for OS Image Download and Preparation

3.1.2 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.

3.2 Setup of Required Features

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

3.2.1 Raspberry Pi Config – SSH, VNC, I2C

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

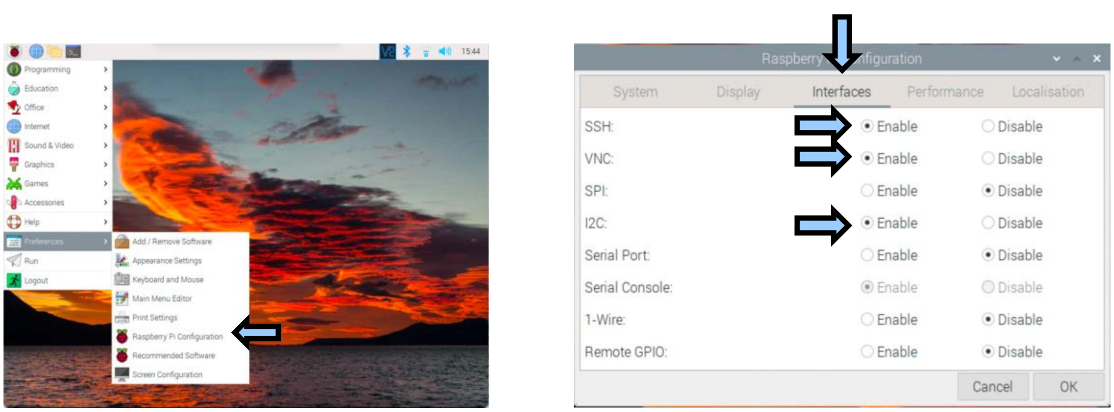

Figure 7 – Enable SSH, VNC, and I2C in Raspberry Pi Configuration Utility

3.2.2 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

3.2.3 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

3.2.4 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

Chapter 4 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.

4.1 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

4.1.1 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

4.1.2 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

4.1.3 Read PDONUM (0x1C)

To display total number of valid PDOs, type the following under command prompt:

i2cget -y 1 0x51 0x1c b

4.1.4 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.

4.1.6 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.

4.1.7 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.

4.1.8 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.

4.1.9 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:

i2cset -y 1 0x51 0x24 0x6e b

To change De-rating threshold to 100°C, user should write 0x64 (=100) to DRTHR by typing the following under command prompt:

i2cset -y 1 0x51 0x25 0x64 b

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

4.1.10 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.

4.1.11 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.

4.2 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/.

4.2.1 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.

4.2.2 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.

4.2.3 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.

4.2.4 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.

4.2.5 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.

4.2.6 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.

4.2.7 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.

4.2.8 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)

4.2.9 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 6.8KΩ 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, 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.

4.2.10 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.

Chapter 5 Practical Examples

5.1 Example 1: Bash I2C-Tools Example: ap33772_querypdo.bash

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

5.1.1 Code Details

```
#!/bin/bash

# This program reports all PDO information

RPI_I2CBUS=1    # Using Raspberry Pi I2C_1            AP33772 has I2C address 0x51
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 times with 1 byte each
        # 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 ))        Process valid PDOs and ignore  invalid
        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 ]                  Process Fixed PDOs. Report Voltage and Max Current
                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                                  Process APDOs. Report Max/Min Voltage and Max
                        # 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 & (0x3f<<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!      Finally report total valid PDO count.
```

5.1.2 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!
```

All PDOs' Capabilities

5.2 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.

5.2.1 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.pdotype=pdotype      # "FPDO" or "APDO"
                self.id=id                       # PDO id/position
                if self.pdotype != "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.pdotype))
```

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.pdotype=pdotype      # "FPDO" or "APDO"
                self.id=id                       # RDO id/position
                if self.pdotype != "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.pdotype))
```

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()
        ValidPDOCnt=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))

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

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

```
        # 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.pdotype="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.pdotype="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))
```

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

Print out valid PDO

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

Clean up invalid PDOs
and keep only valid PDOs

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

    sleep(1.0)
```

Print all PDO information out

```
    # Preparing RDO for later request
    rdolist=list()
    for p in pdolist:
        r=Rdo()
        rdolist.append(r)
        r.id=p.id
        r.pdotype=p.pdotype
        if p.pdotype == "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)
```

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:\tTotal:%-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

5.2.2 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

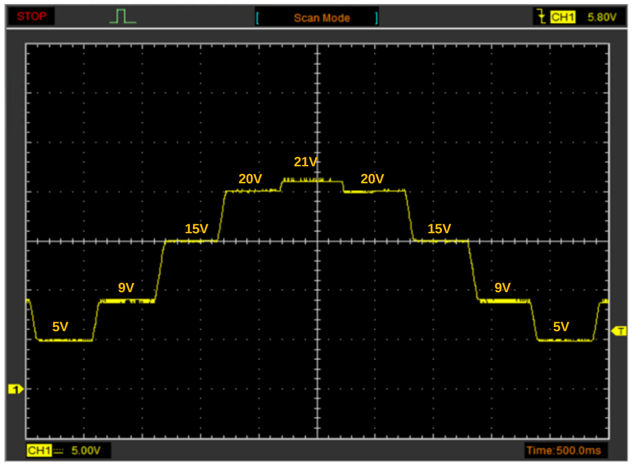

Figure 8 – Example ap33772_allpdo.py Output Waveform

5.3 Example Code Download

5.3.1 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

5.3.2 Example Download Site

Example Codes can be downloaded from Github. Issue the following command to download:

git clone https://github.com/diodinciot/ap33772.git ap33772

Chapter 6 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/

Chapter 7 Revision History

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