Application Report
SLVAE16A – July 2018 – Revised April 2019

TPS65987D and TPS65988 User Alternate Modes

ABSTRACT

The TPS65987D(DX) and TPS65988(DX) is a fully-integrated USB power delivery (PD) management device providing cable plug and orientation detection for USB Type-C and PD plug or receptacle. Each Type-C port that is controlled by the device is functionally identical and supports the full range of the USB Type-C and PD standards. The device supports multiple alternate modes which include DisplayPort, TBT and User alternate mode and so forth. This document describes the procedure for enabling and configuring a user alternate mode using the software tools and an optional host controller.

Contents

List of Figures

List of Tables

Trademarks

All trademarks are the property of their respective owners.

1 Introduction

The user alternate mode allows users to configure a custom SVID with up to four independently configurable mode numbers. When enabled, the device adds this custom SVID to the list of supported SVIDs, and is shared with the port partner in the acknowledgment to the ‘Discover SVIDs’ command. The DFP can then command the port partner to enter a specified mode of operation, and exchange proprietary messages after entering the custom mode.

The user alternate mode can be used either with or without an external microcontroller. Without an external microcontroller, the capabilities of the user alternate mode are limited to entering the mode, optionally sending a predefined unstructured VDM upon mode entry, and optionally reconfiguring the device registers and executing up to two host interface commands.

The ability to send a predefined unstructured VDM upon mode entry is generally used to advertise an identity. For instance, vendors can define a custom alternate mode used to communicate between supported power supplies and devices. A power supply that does not contain an external microcontroller could configure the device to automatically send an unstructured VDM, advertising information about the power supply such as the model number, revision, serial number, and other information.

The ability to reconfigure the device on mode entry allows modification of any of the configuration registers of the host interface without an external microcontroller. This ability can be used, for instance, to modify the power sourcing and sinking capabilities of the PD port when recognized and supported devices are attached. After reconfiguration of host interface registers, up to two host interface commands can be executed. These commands can be used to drive a GPIO, or force a renegotiation of the power contract, or execute a data/power role swap.

The primary limitation of the user alternate mode, when used without an external microcontroller, is its lack of decision-making capability. The user alternate mode can be configured to send an arbitrary message or to change the capabilities of the device, but this is a static configuration that is based only on mode entry and exit. With the addition of an external microcontroller, the capabilities of the user alternate mode can be greatly extended.

2 Configuration Registers

The user alternate mode is configured using the ‘User Alternate Mode (0x4A)’ configuration register. If the user is enabling the User Alternate Mode to reconfigure the behavior of the device or to issue host interface commands upon mode entry, then these capabilities are set in the ‘App Configuration Register (0x6C)’.

‘User VID Status (0x57)’ register provides status information of the user alternate mode. This can be used by an external microcontroller for decision making at runtime.

The device stores the last received attention and non-attention VDM in ‘Rx User VID Attention VDM (0x60)’ and ‘Rx User VID Other VDM (0x61)’ registers respectively. These two registers are dedicated to the user alternate mode and are not overwritten by other alternate mode messages such as DisplayPort or TBT, which may be running concurrently. These registers may be used by an external microcontroller in order to extend the capabilities of the user alternate mode.

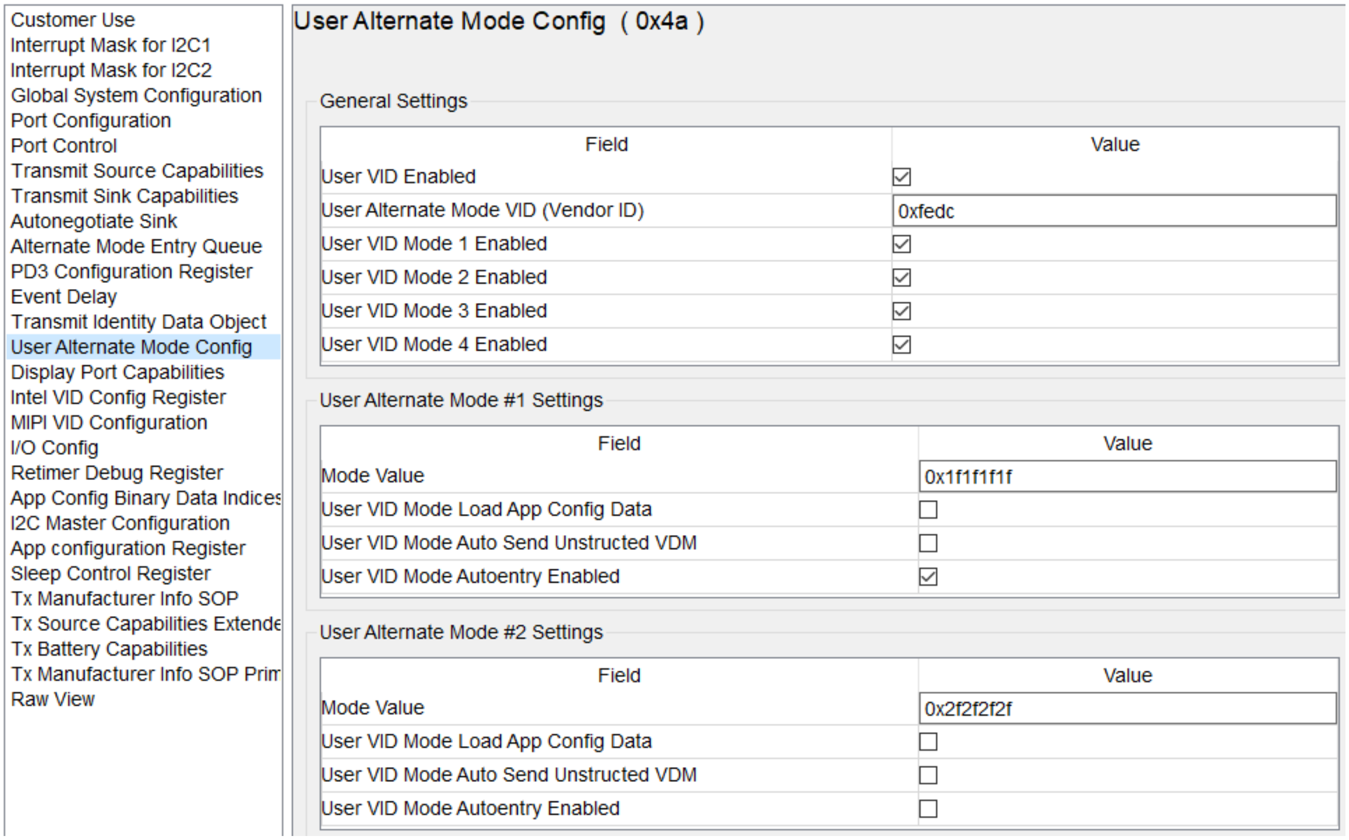
The interrupt registers (0x14 - 0x17) may be configured to generate an interrupt to the external microcontroller, whenever a new attention or non-attention VDM is received on the user SVID channel.

Refer to SLVUBH2 for more details on the register definitions.

3 Basic Configuration

The basic configuration of the user alternate mode is handled in the ‘User Alternate Mode (0x4A)’ configuration register. The example in this section configures the device to support a structured Custom-VID ‘0xFEDC’ with four alternate modes. All four alternate modes are enabled and their mode values are 0x1F1F1F1F, 0x2F2F2F2F, 0x3F3F3F3F and 0x4F4F4F4F. Auto-Entry is enabled for Mode-1 and Mode-4, and Mode-4 is additionally configured to send a predefined unstructured message upon mode entry

Figure 1. Basic Configuration - User Alternate Mode Configuration


Customer Use
Interrupt Mask for I2C1
Interrupt Mask for I2C2
Global System Configuration
Port Configuration
Port Control
Transmit Source Capabilities
Transmit Sink Capabilities
Autonegotiate Sink
Alternate Mode Entry Queue
PD3 Configuration Register
Event Delay
Transmit Identity Data Object
User Alternate Mode Config
Display Port Capabilities
Intel VID Config Register
MIPI VID Configuration
I/O Config
Retimer Debug Register
App Config Binary Data Indices
I2C Master Configuration
App configuration Register
Sleep Control Register
Tx Manufacturer Info SOP
Tx Source Capabilities Extende
Tx Battery Capabilities
Tx Manufacturer Info SOP Prim
Raw View

User Alternate Mode Config (0x4a)

General Settings

Field	Value
User VID Enabled	☑
User Alternate Mode VID (Vendor ID)	0xfedc
User VID Mode 1 Enabled	☑
User VID Mode 2 Enabled	☑
User VID Mode 3 Enabled	☑
User VID Mode 4 Enabled	☑

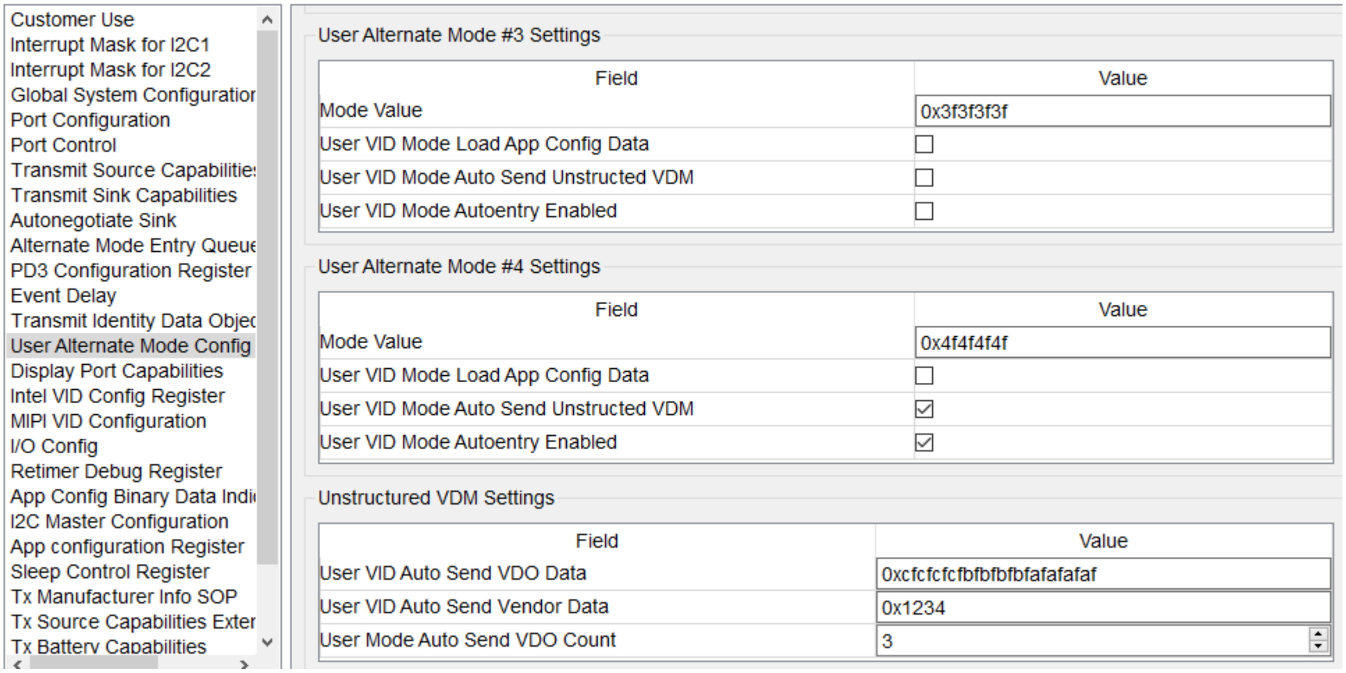
User Alternate Mode #1 Settings

Field	Value
Mode Value	0x1f1f1f1f
User VID Mode Load App Config Data	□
User VID Mode Auto Send Unstructed VDM	□
User VID Mode Autoentry Enabled	☑

User Alternate Mode #2 Settings

Field	Value
Mode Value	0x2f2f2f2f
User VID Mode Load App Config Data	□
User VID Mode Auto Send Unstructed VDM	□
User VID Mode Autoentry Enabled	□

Figure 2. Basic Configuration - User Altenate Mode Configuration


Customer Use
Interrupt Mask for I2C1
Interrupt Mask for I2C2
Global System Configuratior
Port Configuration
Port Control
Transmit Source Capabilitie
Transmit Sink Capabilities
Autonegotiate Sink
Alternate Mode Entry Queue
PD3 Configuration Register
Event Delay
Transmit Identity Data Objec
User Alternate Mode Config
Display Port Capabilities
Intel VID Config Register
MIPI VID Configuration
I/O Config
Retimer Debug Register
App Config Binary Data Indi
I2C Master Configuration
App configuration Register
Sleep Control Register
Tx Manufacturer Info SOP
Tx Source Capabilities Exter
Tx Battery Capabilities

User Alternate Mode #3 Settings

Field	Value
Mode Value	0x3f3f3f3f
User VID Mode Load App Config Data	□
User VID Mode Auto Send Unstructed VDM	□
User VID Mode Autoentry Enabled	□

User Alternate Mode #4 Settings

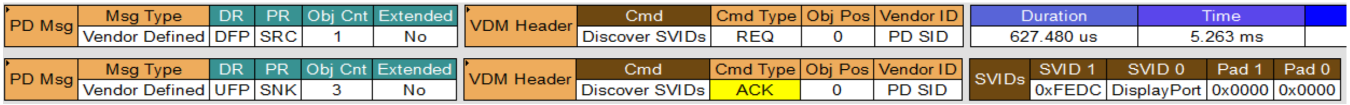
Field	Value
Mode Value	0x4f4f4f4f
User VID Mode Load App Config Data	□
User VID Mode Auto Send Unstructed VDM	☑
User VID Mode Autoentry Enabled	☑

Unstructured VDM Settings

Field	Value
User VID Auto Send VDO Data	0xcfcfcfcfbfbfbfbfafafafaf
User VID Auto Send Vendor Data	0x1234
User Mode Auto Send VDO Count	3

When this port is connected to a PD partner that supports all these modes, the port negotiate the alternate mode contract, and the PD message exchange between the ports will be as below:

1. The port-partner will share the information about all the SVIDs that it supports in the acknowledgment to the device's 'Discover SVIDs' command



PD Msg	Msg Type	DR	PR	Obj Cnt	Extended
	Vendor Defined	DFP	SRC	1	No

VDM Header	Cmd	Cmd Type	Obj Pos	Vendor ID
	Discover SVIDs	REQ	0	PD SID

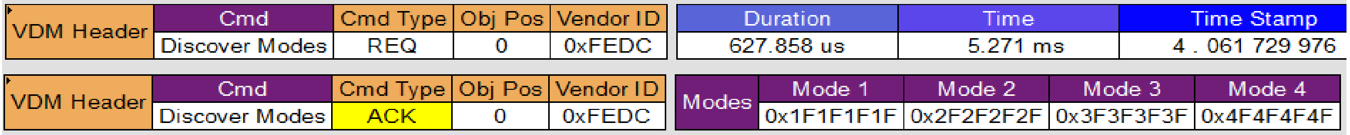
Duration	Time	
627.480 us	5.263 ms	

PD Msg	Msg Type	DR	PR	Obj Cnt	Extended
	Vendor Defined	UFP	SNK	3	No

VDM Header	Cmd	Cmd Type	Obj Pos	Vendor ID
	Discover SVIDs	ACK	0	PD SID

SVIDs	SVID 1	SVID 0	Pad 1	Pad 0
	0xFEDC	DisplayPort	0x0000	0x0000

2. The port partner will share the information about all the modes that it supports for custom SVID '0xFEDC' in the acknowledgment to the device's 'Discover Modes' command. The mode numbers returned by the UFP correspond to the 'Mode Value' field of the configuration register.



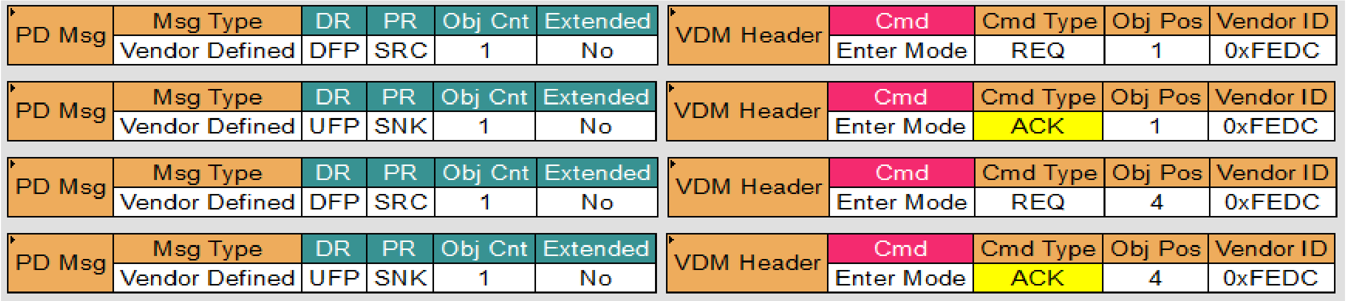
VDM Header	Cmd	Cmd Type	Obj Pos	Vendor ID
	Discover Modes	REQ	0	0xFEDC

Duration	Time	Time Stamp
627.858 us	5.271 ms	4 . 061 729 976

VDM Header	Cmd	Cmd Type	Obj Pos	Vendor ID
	Discover Modes	ACK	0	0xFEDC

Modes	Mode 1	Mode 2	Mode 3	Mode 4
	0x1F1F1F1F	0x2F2F2F2F	0x3F3F3F3F	0x4F4F4F4F

3. The device automatically enters Mode-1 and Mode-4, and sends a predefined unstructured message on entering Mode-4 as per the above configuration.



PD Msg	Msg Type	DR	PR	Obj Cnt	Extended
	Vendor Defined	DFP	SRC	1	No

VDM Header	Cmd	Cmd Type	Obj Pos	Vendor ID
	Enter Mode	REQ	1	0xFEDC

PD Msg	Msg Type	DR	PR	Obj Cnt	Extended
	Vendor Defined	UFP	SNK	1	No

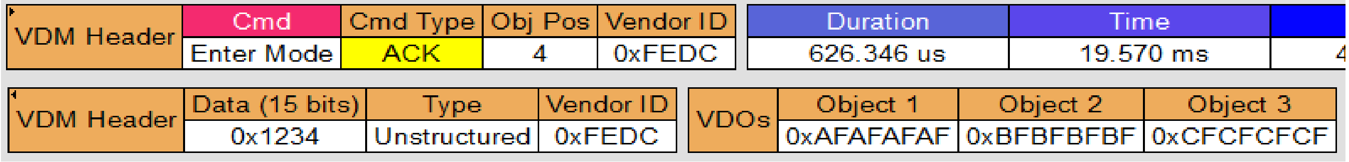
VDM Header	Cmd	Cmd Type	Obj Pos	Vendor ID
	Enter Mode	ACK	1	0xFEDC

PD Msg	Msg Type	DR	PR	Obj Cnt	Extended
	Vendor Defined	DFP	SRC	1	No

VDM Header	Cmd	Cmd Type	Obj Pos	Vendor ID
	Enter Mode	REQ	4	0xFEDC

PD Msg	Msg Type	DR	PR	Obj Cnt	Extended
	Vendor Defined	UFP	SNK	1	No

VDM Header	Cmd	Cmd Type	Obj Pos	Vendor ID
	Enter Mode	ACK	4	0xFEDC



VDM Header	Cmd	Cmd Type	Obj Pos	Vendor ID
	Enter Mode	ACK	4	0xFEDC

Duration	Time	
626.346 us	19.570 ms	4

VDM Header	Data (15 bits)	Type	Vendor ID
	0x1234	Unstructured	0xFEDC

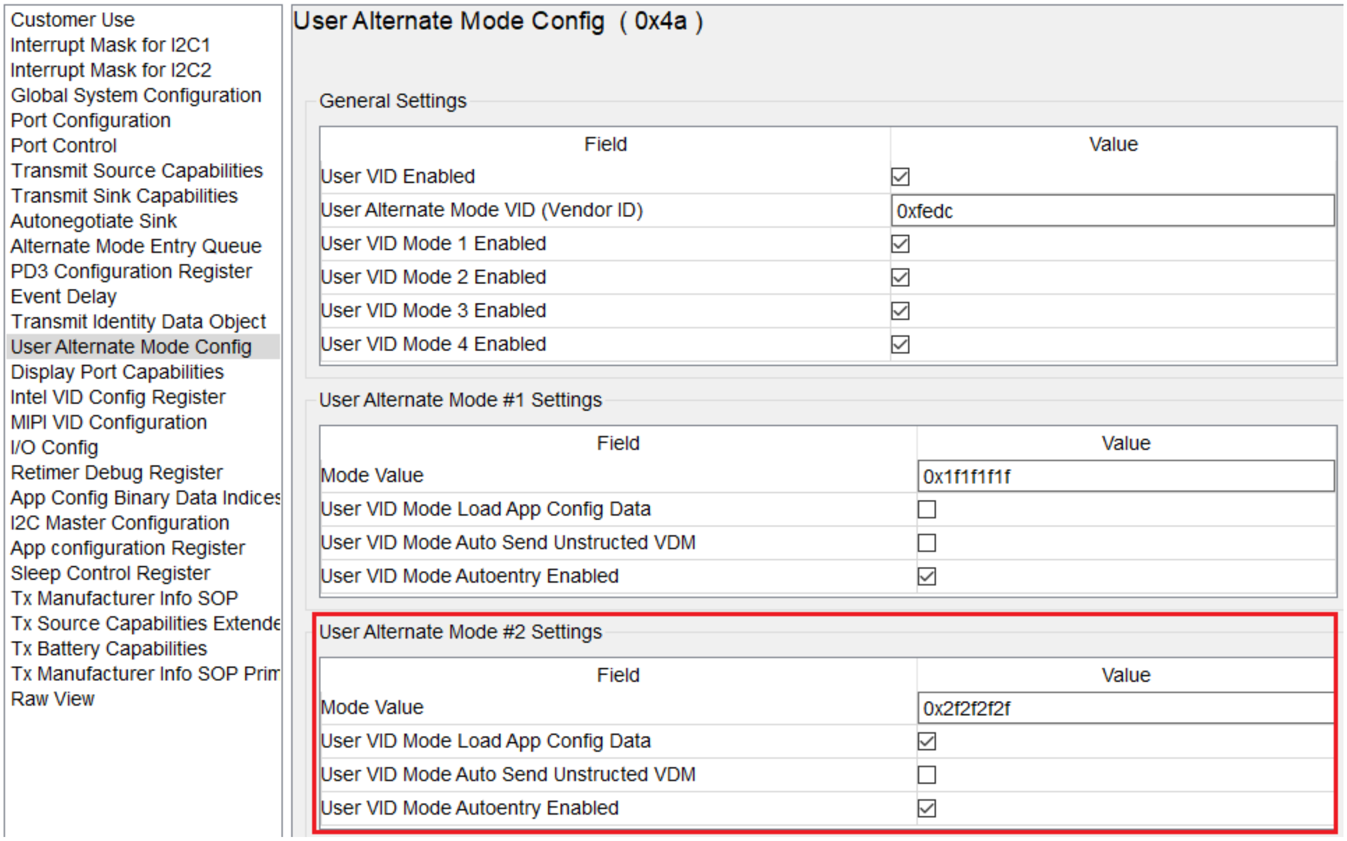
VDOs	Object 1	Object 2	Object 3
	0xAFAFAFAF	0xBFBFBFBF	0xCFCFCFCF

4. Mode-2 and Mode-3 were not marked for auto-entry. Hence the device doesn't automatically enter these modes. The host can explicitly command the device to enter these modes using 'AMEn' command

4 Advanced Configuration

In addition to advertising and automatically entering the user alternate modes, the device can be configured to load configuration sets and issue up to two host interface commands on mode entry and exit. The loading of configuration sets is enabled by 'User VID Mode Load App Config Data' flag of the corresponding 'User Alternate Mode #N Settings' in the configuration register 0x4A. In this example, the basic configuration set of the previous section is slightly modified as below to load an application configuration on Mode-2 entry:

Figure 3. Advanced Configuration - User Alternate Mode Configuration


Customer Use
Interrupt Mask for I2C1
Interrupt Mask for I2C2
Global System Configuration
Port Configuration
Port Control
Transmit Source Capabilities
Transmit Sink Capabilities
Autonegotiate Sink
Alternate Mode Entry Queue
PD3 Configuration Register
Event Delay
Transmit Identity Data Object
User Alternate Mode Config
Display Port Capabilities
Intel VID Config Register
MIPI VID Configuration
I/O Config
Retimer Debug Register
App Config Binary Data Indices
I2C Master Configuration
App configuration Register
Sleep Control Register
Tx Manufacturer Info SOP
Tx Source Capabilities Extende
Tx Battery Capabilities
Tx Manufacturer Info SOP Prim
Raw View

User Alternate Mode Config (0x4a)

General Settings

Field	Value
User VID Enabled	☑
User Alternate Mode VID (Vendor ID)	0xfedc
User VID Mode 1 Enabled	☑
User VID Mode 2 Enabled	☑
User VID Mode 3 Enabled	☑
User VID Mode 4 Enabled	☑

User Alternate Mode #1 Settings

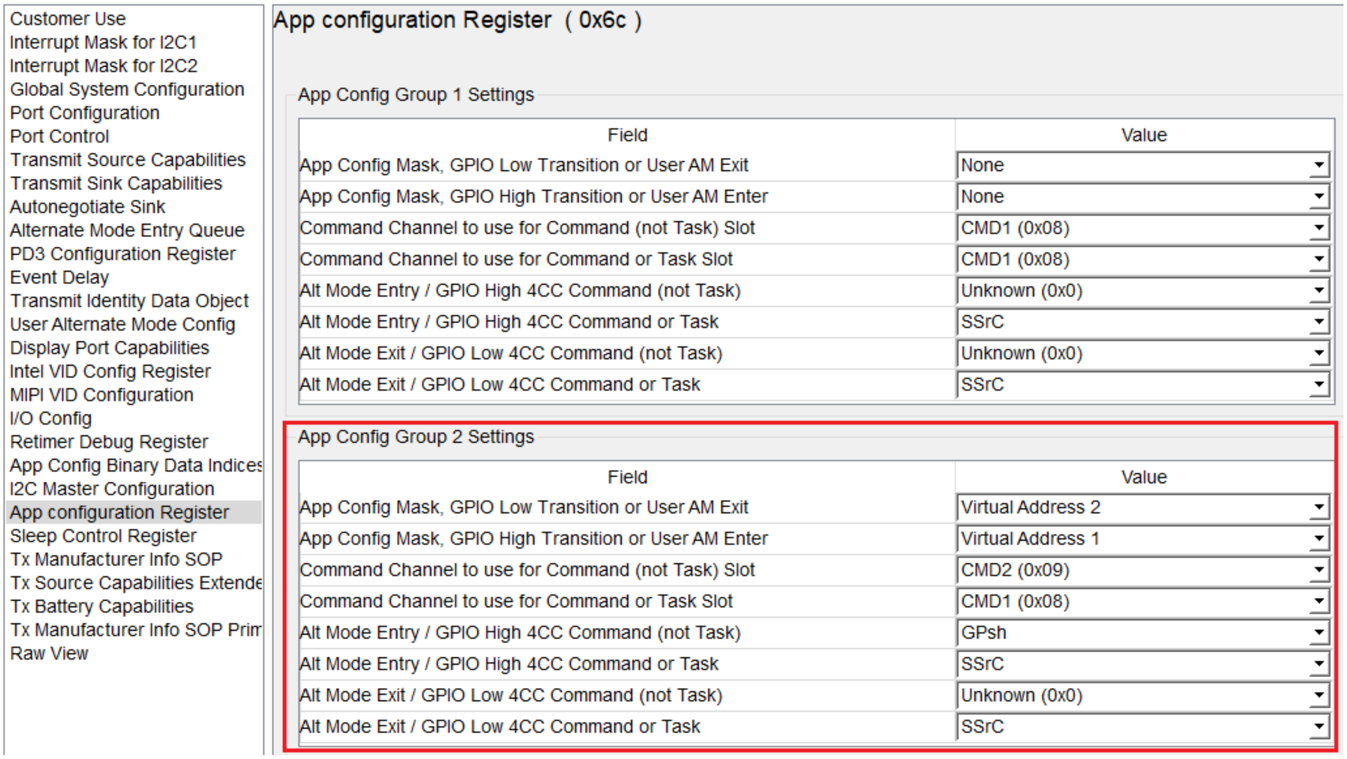
Field	Value
Mode Value	0x1f1f1f1f
User VID Mode Load App Config Data	□
User VID Mode Auto Send Unstructed VDM	□
User VID Mode Autoentry Enabled	☑

User Alternate Mode #2 Settings

Field	Value
Mode Value	0x2f2f2f2f
User VID Mode Load App Config Data	☑
User VID Mode Auto Send Unstructed VDM	□
User VID Mode Autoentry Enabled	☑

The configuration set that is to be loaded and the optional host interface commands to be executed (after the configuration set is loaded) are specified in the App Configuration Register (0x6C). This configuration register has three sections, namely ‘App Config GPIO Group 1 Settings’, ‘App Config GPIO Group 2 Settings’ and ‘App Config GPIO Group 3 Settings’. These three sections allow the user to add configuration sets (to be loaded on the entry or exit) for the first three user alternate modes respectively. The fourth user alternate mode does not support configuration set loading. Since Mode-2 is configured to load an application configuration in this example, the settings in ‘App Config GPIO Group 2 Settings’ shall be configured. The example in this section configures the device to load ‘Virtual Device 1 (0x1)’ settings on mode entry, and ‘Virtual Device 2 (0x2)’ settings on mode exit. The device is also configured to execute ‘GPsh’ and ‘SSrC’ command after the settings are loaded. ‘GPsh’ is executed on mode entry only, and ‘SSrC’ is executed on mode entry and exit.

Figure 4. Advanced Configuration - App Configuration Register



Customer Use
Interrupt Mask for I2C1
Interrupt Mask for I2C2
Global System Configuration
Port Configuration
Port Control
Transmit Source Capabilities
Transmit Sink Capabilities
Autonegotiate Sink
Alternate Mode Entry Queue
PD3 Configuration Register
Event Delay
Transmit Identity Data Object
User Alternate Mode Config
Display Port Capabilities
Intel VID Config Register
MIPI VID Configuration
I/O Config
Retimer Debug Register
App Config Binary Data Indices
I2C Master Configuration
App configuration Register
Sleep Control Register
Tx Manufacturer Info SOP
Tx Source Capabilities Extende
Tx Battery Capabilities
Tx Manufacturer Info SOP Prim
Raw View

App configuration Register (0x6c)

App Config Group 1 Settings

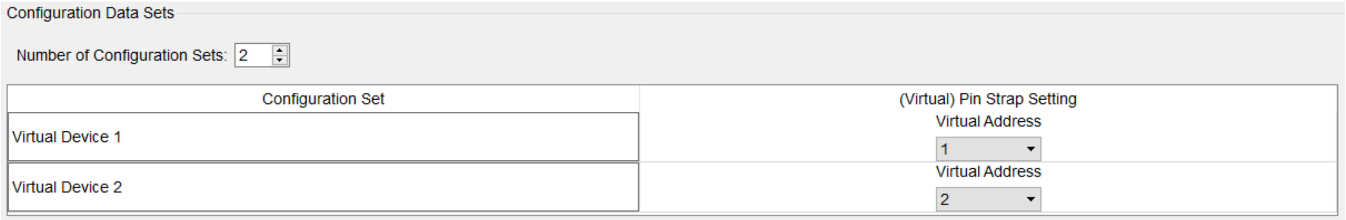
Field	Value
App Config Mask, GPIO Low Transition or User AM Exit	None
App Config Mask, GPIO High Transition or User AM Enter	None
Command Channel to use for Command (not Task) Slot	CMD1 (0x08)
Command Channel to use for Command or Task Slot	CMD1 (0x08)
Alt Mode Entry / GPIO High 4CC Command (not Task)	Unknown (0x0)
Alt Mode Entry / GPIO High 4CC Command or Task	SSrC
Alt Mode Exit / GPIO Low 4CC Command (not Task)	Unknown (0x0)
Alt Mode Exit / GPIO Low 4CC Command or Task	SSrC

App Config Group 2 Settings

Field	Value
App Config Mask, GPIO Low Transition or User AM Exit	Virtual Address 2
App Config Mask, GPIO High Transition or User AM Enter	Virtual Address 1
Command Channel to use for Command (not Task) Slot	CMD2 (0x09)
Command Channel to use for Command or Task Slot	CMD1 (0x08)
Alt Mode Entry / GPIO High 4CC Command (not Task)	GPsh
Alt Mode Entry / GPIO High 4CC Command or Task	SSrC
Alt Mode Exit / GPIO Low 4CC Command (not Task)	Unknown (0x0)
Alt Mode Exit / GPIO Low 4CC Command or Task	SSrC

In this configuration example, ‘App Config Mask, GPIO Low Transition or User AM Exit’ and ‘App Config Mask, GPIO High Transition or User AM Enter’ is mapped to virtual identifiers 0x2 and 0x1 respectively. These identifiers are determined from the ‘(Virtual) Pin Strap Setting’ field associated with each ‘Configuration Data Sets’ on the ‘General Settings’ tab.

Figure 5. Advanced Configuration - Configuration Data Sets



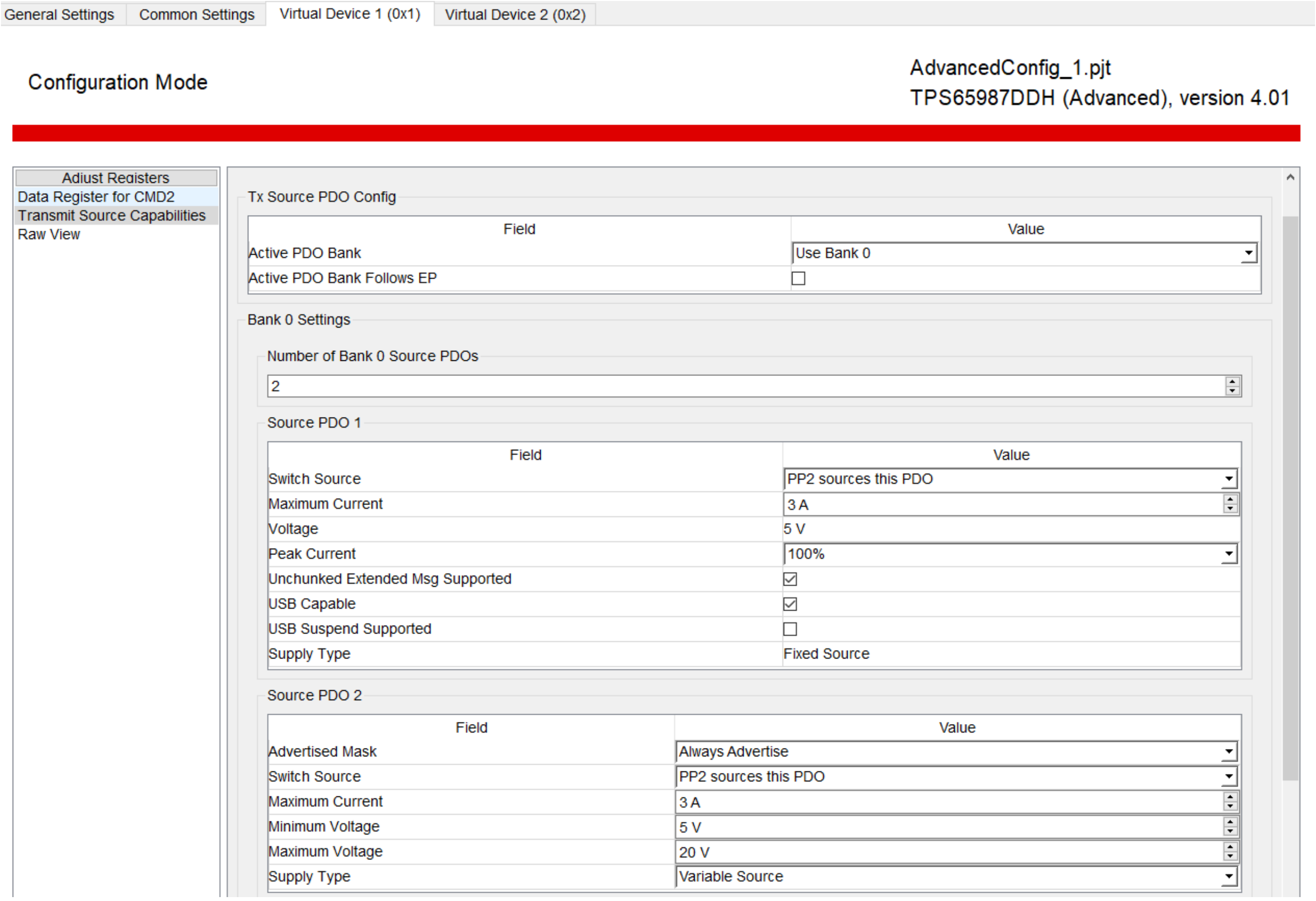
Configuration Data Sets

Number of Configuration Sets: 2

Configuration Set	(Virtual) Pin Strap Setting
Virtual Device 1	Virtual Address 1
Virtual Device 2	Virtual Address 2

Configuration settings tab for ‘Virtual Device 1 (0x1)’ and ‘Virtual Device 2 (0x2)’ shows that they specify settings for the CMD2 Data Register and Transmit Source Capabilities Register (0x32). Registers may be added to or removed from this set by selecting the ‘Adjust Registers’ button that appears above the register list in the left pane.

Figure 6. Advanced Configuration - Virtual Device Settings



Configuration Mode

AdvancedConfig_1.pjt
TPS65987DDH (Advanced), version 4.01

Tx Source PDO Config

Field	Value
Active PDO Bank	Use Bank 0
Active PDO Bank Follows EP	□

Bank 0 Settings

Number of Bank 0 Source PDOs: 2

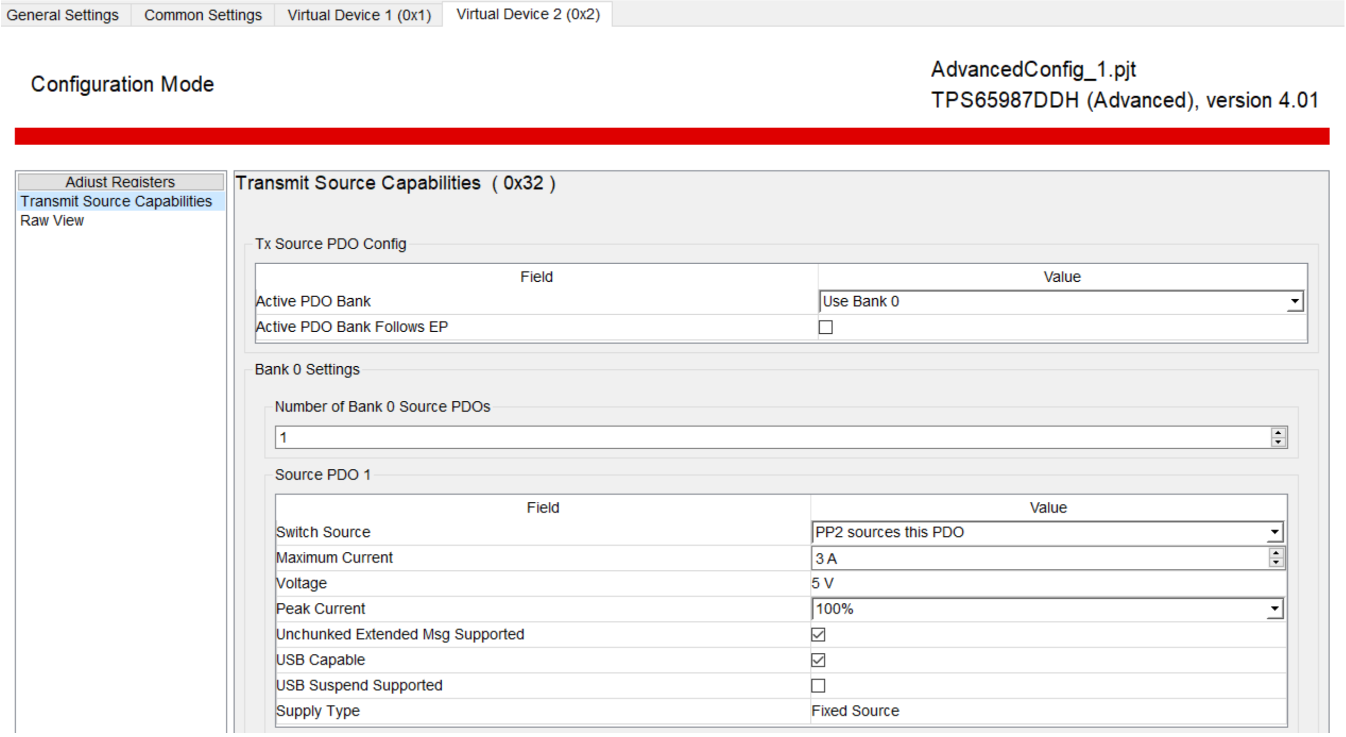
Source PDO 1

Field	Value
Switch Source	PP2 sources this PDO
Maximum Current	3 A
Voltage	5 V
Peak Current	100%
Unchunked Extended Msg Supported	☑
USB Capable	☑
USB Suspend Supported	□
Supply Type	Fixed Source

Source PDO 2

Field	Value
Advertised Mask	Always Advertise
Switch Source	PP2 sources this PDO
Maximum Current	3 A
Minimum Voltage	5 V
Maximum Voltage	20 V
Supply Type	Variable Source

Figure 7. Advanced Configuration - Virtual Device Settings



General Settings | Common Settings | Virtual Device 1 (0x1) | Virtual Device 2 (0x2)

Configuration Mode

AdvancedConfig_1.pjt
TPS65987DDH (Advanced), version 4.01

Adjust Registers
Transmit Source Capabilities
Raw View

Transmit Source Capabilities (0x32)

Tx Source PDO Config

Field	Value
Active PDO Bank	Use Bank 0
Active PDO Bank Follows EP	□

Bank 0 Settings

Number of Bank 0 Source PDOs: 1

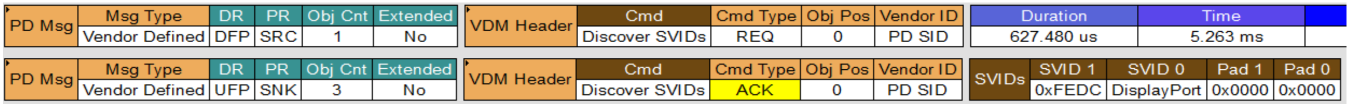
Source PDO 1

Field	Value
Switch Source	PP2 sources this PDO
Maximum Current	3 A
Voltage	5 V
Peak Current	100%
Unchunked Extended Msg Supported	☑
USB Capable	☑
USB Suspend Supported	□
Supply Type	Fixed Source

Comparison of the Transmit Source Capabilities registers as specified in 'Virtual Device 1' and 'Virtual Device 2' shows that upon entry into the user alternate Mode-2, the Transmit Source Capabilities register will be populated with two source PDOs (fixed 5 V and variable 5 V — 20), and upon exit, the Transmit Source Capabilities register will be populated with one source PDO (fixed 5 V). It can also be verified that the initialization parameters for this register, specified in the 'Common Settings' tab match those of 'Virtual Device 2' since the system is always initialized in a state where no alternate modes have been entered.

When this port is connected to a PD partner that supports all these modes, the ports negotiate the alternate mode contract, and the PD message exchange between the ports will be as below:

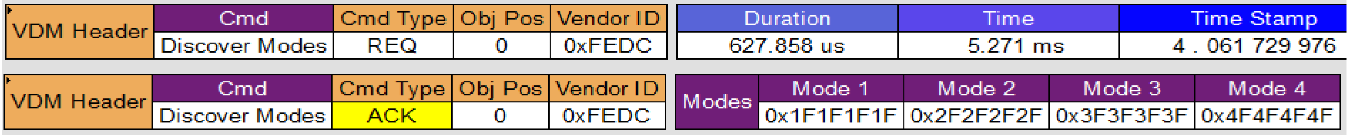
1. The port-partner will share the information about all the SVIDs that it supports in the acknowledgment to the device's 'Discover SVIDs' command



	Msg Type	DR	PR	Obj Cnt	Extended		Cmd	Cmd Type	Obj Pos	Vendor ID	Duration	Time
PD Msg	Vendor Defined	DFP	SRC	1	No	VDM Header	Discover SVIDs	REQ	0	PD SID	627.480 us	5.263 ms

	Msg Type	DR	PR	Obj Cnt	Extended		Cmd	Cmd Type	Obj Pos	Vendor ID		SVID 1	SVID 0	Pad 1	Pad 0
PD Msg	Vendor Defined	UFP	SNK	3	No	VDM Header	Discover SVIDs	ACK	0	PD SID	SVIDs	0xFEDC	DisplayPort	0x0000	0x0000

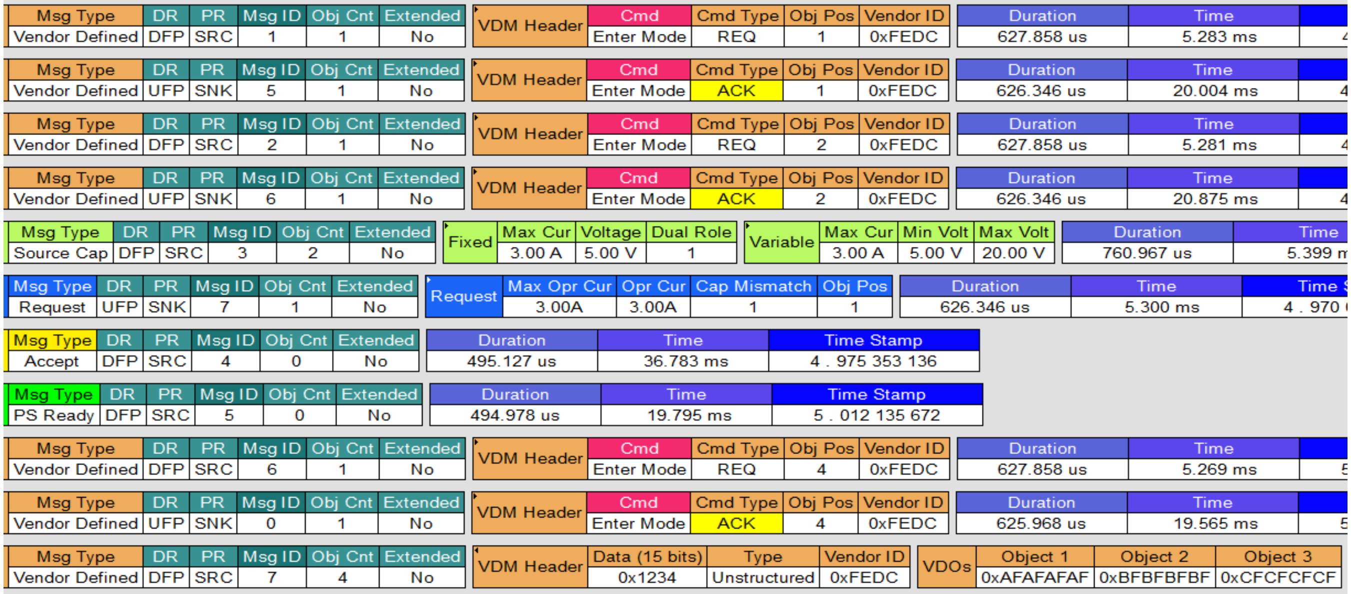
2. The port partner will share the information about all the modes that it supports for this custom SVID '0xFEDC' in the acknowledgment to the device's 'Discover Modes' command. The mode numbers returned by the UFP correspond to the 'Mode Value' field of the configuration register.



	Cmd	Cmd Type	Obj Pos	Vendor ID	Duration	Time	Time Stamp
VDM Header	Discover Modes	REQ	0	0xFEDC	627.858 us	5.271 ms	4 . 061 729 976

	Cmd	Cmd Type	Obj Pos	Vendor ID		Mode 1	Mode 2	Mode 3	Mode 4
VDM Header	Discover Modes	ACK	0	0xFEDC	Modes	0x1F1F1F1F	0x2F2F2F2F	0x3F3F3F3F	0x4F4F4F4F

3. The device automatically enters Mode-1, Mode-2 and Mode-4. The device sends the 'Source Capabilities' on entering Mode-2 (thereby renegotiate the PD contract) and sends a predefined unstructured message on entering Mode-4 as per the above configuration.



Msg Type	DR	PR	Msg ID	Obj Cnt	Extended	VDM Header	Cmd	Cmd Type	Obj Pos	Vendor ID	Duration	Time	
Vendor Defined	DFP	SRC	1	1	No		Enter Mode	REQ	1	0xFEDC	627.858 us	5.283 ms	4

Msg Type	DR	PR	Msg ID	Obj Cnt	Extended	VDM Header	Cmd	Cmd Type	Obj Pos	Vendor ID	Duration	Time	
Vendor Defined	UFP	SNK	5	1	No		Enter Mode	ACK	1	0xFEDC	626.346 us	20.004 ms	4

Msg Type	DR	PR	Msg ID	Obj Cnt	Extended	VDM Header	Cmd	Cmd Type	Obj Pos	Vendor ID	Duration	Time	
Vendor Defined	DFP	SRC	2	1	No		Enter Mode	REQ	2	0xFEDC	627.858 us	5.281 ms	4

Msg Type	DR	PR	Msg ID	Obj Cnt	Extended	VDM Header	Cmd	Cmd Type	Obj Pos	Vendor ID	Duration	Time	
Vendor Defined	UFP	SNK	6	1	No		Enter Mode	ACK	2	0xFEDC	626.346 us	20.875 ms	4

Msg Type	DR	PR	Msg ID	Obj Cnt	Extended	Fixed	Max Cur	Voltage	Dual Role	Variable	Max Cur	Min Volt	Max Volt	Duration	Time
Source Cap	DFP	SRC	3	2	No		3.00 A	5.00 V	1		3.00 A	5.00 V	20.00 V	760.967 us	5.399 m

Msg Type	DR	PR	Msg ID	Obj Cnt	Extended	Request	Max Opr Cur	Opr Cur	Cap Mismatch	Obj Pos	Duration	Time	Time S
Request	UFP	SNK	7	1	No		3.00A	3.00A	1	1	626.346 us	5.300 ms	4 . 970

Msg Type	DR	PR	Msg ID	Obj Cnt	Extended	Duration	Time	Time Stamp
Accept	DFP	SRC	4	0	No	495.127 us	36.783 ms	4 . 975 353 136

Msg Type	DR	PR	Msg ID	Obj Cnt	Extended	Duration	Time	Time Stamp
PS Ready	DFP	SRC	5	0	No	494.978 us	19.795 ms	5 . 012 135 672

Msg Type	DR	PR	Msg ID	Obj Cnt	Extended	VDM Header	Cmd	Cmd Type	Obj Pos	Vendor ID	Duration	Time	
Vendor Defined	DFP	SRC	6	1	No		Enter Mode	REQ	4	0xFEDC	627.858 us	5.269 ms	5

Msg Type	DR	PR	Msg ID	Obj Cnt	Extended	VDM Header	Cmd	Cmd Type	Obj Pos	Vendor ID	Duration	Time	
Vendor Defined	UFP	SNK	0	1	No		Enter Mode	ACK	4	0xFEDC	625.968 us	19.565 ms	5

Msg Type	DR	PR	Msg ID	Obj Cnt	Extended	VDM Header	Data (15 bits)	Type	Vendor ID	VDOs	Object 1	Object 2	Object 3
Vendor Defined	DFP	SRC	7	4	No		0x1234	Unstructured	0xFEDC		0xAFAFAFAF	0xBFBFBFBF	0xCFCFCFCF

4. The device does not automatically enter Mode-3 as this mode was not marked for auto-entry. The host application can explicitly command the device to enter this mode using ‘AMEn’ command.

The preceding section explains the method for reconfiguring the host interface register settings automatically upon entry into or exit from user alternate modes. In the example presented, the Transmit Source Capabilities register was modified upon entry into and exit from user alternate Mode-2. Overwriting the Transmit Source Capabilities register does not, however, force a retransmission of source capabilities. This is accomplished by issuing the host interface command ‘SSrC’. As many as one Host Interface Command and one Host Interface Task may be executed upon user alternate mode entry and exit. These may be individually specified for entry and exit. For instance, a mode could issue the ‘SWSr’ (SWap to Source) task upon entering a given mode but issue the ‘SWSk’ (SWap to Sink) task upon exiting the same mode.

5 Advanced Configurations with EC

The user alternate mode capabilities and example configurations presented in the previous sections of this document are static configurations based on mode entry and/or exit. The capabilities of the user alternate mode can be greatly expanded with the addition of an external microcontroller, and the subsequent sections present few simple use-cases that can be implemented using the user alternate modes.

5.1 Example 1

This example defines a Custom-VID ‘0x0055’ which supports two alternate modes with their mode values as 0x1 and 0x2 respectively. The port partner is assumed to support this custom VID and its modes.

1. Using Mode-1, Port-A commands Port-B to drive a GPIO(s) on receiving an external trigger. The example uses unstructured VDM for exchanging messages between the port partners, and the message construct can be entirely defined by the vendors. This simple use-case is particularly applicable to applications such as laptop docking stations, where a push button event can be used to send status information from one device to another.
2. Using Mode-2, Port-A queries the status information of Port-B. This simple example demonstrates the ability of the user alternate modes to exchange proprietary information and build complex use-cases around it, for instance, to modify the power sinking capabilities of a laptop depending on its battery charging properties when connected to a recognized and supported PD adapter.

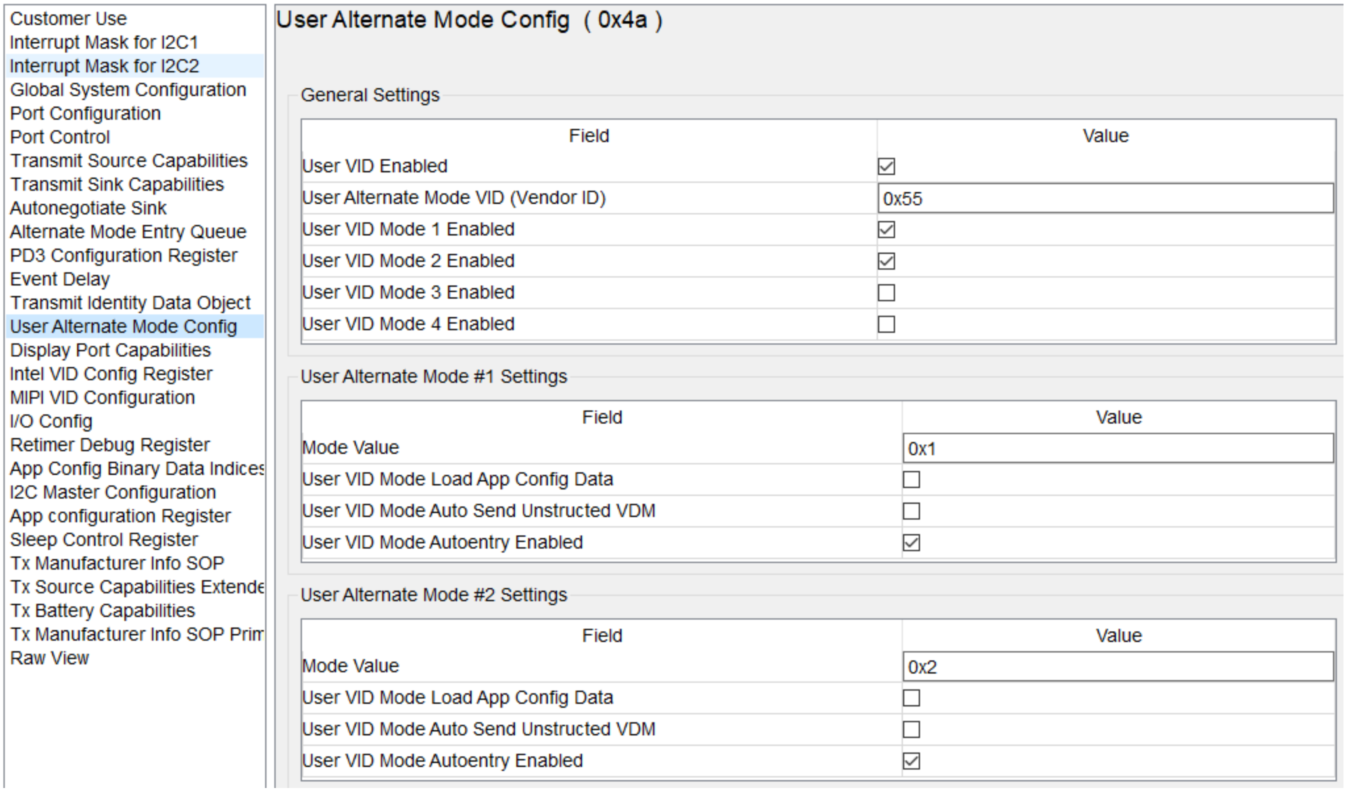
Both these mode examples use an unstructured VDM to exchange proprietary information with their port partner. Per PD specification, Bit-14:0 of an ‘Unstructured VDM Header’ is available for vendor’s use, and the content of this field can be defined by the vendors.

This example defines Bit-14:0 of the unstructured VDM header as below:

```
/*!
 *  \brief UVDM Header Structure
 */
typedef struct __attribute__((packed))
{
    uint32_t    cmdtype     : 2;
    uint32_t    mode        : 3;
    uint32_t    reserved    : 3;
    uint32_t    command     : 4;
    uint32_t    totalvdos   : 3;
    uint32_t    vdmtype     : 1;
    uint32_t    svid        : 16;

}s_TPS_uvdmHeader;
```

The example in this section configures the device to support a Custom-VID ‘0x55’ with two alternate modes. The two alternate modes are enabled and their mode values are 0x1 and 0x2. Auto-Entry is enabled for both the modes.

Figure 8. Advanced Configuration with EC - User Alternate Mode Register


Customer Use
Interrupt Mask for I2C1
Interrupt Mask for I2C2
Global System Configuration
Port Configuration
Port Control
Transmit Source Capabilities
Transmit Sink Capabilities
Autonegotiate Sink
Alternate Mode Entry Queue
PD3 Configuration Register
Event Delay
Transmit Identity Data Object
User Alternate Mode Config
Display Port Capabilities
Intel VID Config Register
MIPI VID Configuration
I/O Config
Retimer Debug Register
App Config Binary Data Indices
I2C Master Configuration
App configuration Register
Sleep Control Register
Tx Manufacturer Info SOP
Tx Source Capabilities Extende
Tx Battery Capabilities
Tx Manufacturer Info SOP Prim
Raw View

User Alternate Mode Config (0x4a)

General Settings

Field	Value
User VID Enabled	☑
User Alternate Mode VID (Vendor ID)	0x55
User VID Mode 1 Enabled	☑
User VID Mode 2 Enabled	☑
User VID Mode 3 Enabled	□
User VID Mode 4 Enabled	□

User Alternate Mode #1 Settings

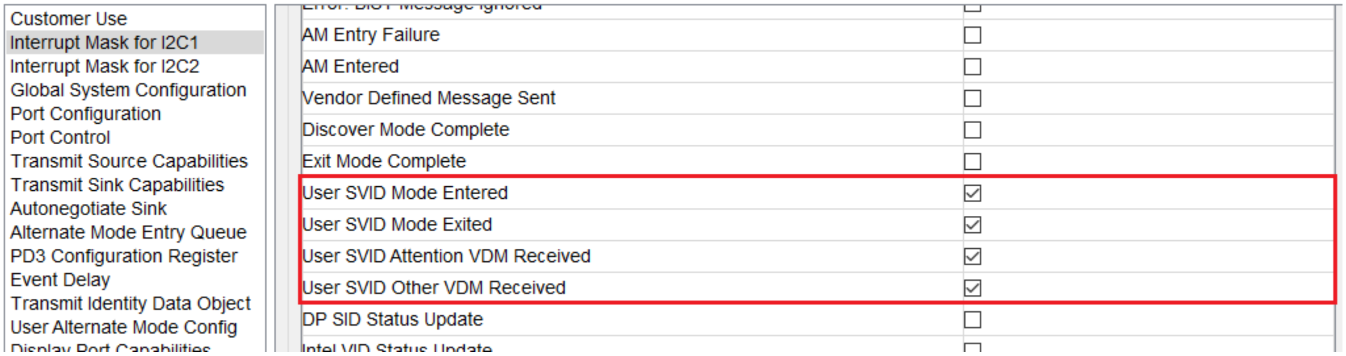
Field	Value
Mode Value	0x1
User VID Mode Load App Config Data	□
User VID Mode Auto Send Unstructed VDM	□
User VID Mode Autoentry Enabled	☑

User Alternate Mode #2 Settings

Field	Value
Mode Value	0x2
User VID Mode Load App Config Data	□
User VID Mode Auto Send Unstructed VDM	□
User VID Mode Autoentry Enabled	☑

The device is also configured to generate below events and notify the host on mode entry/exit and the reception of the vendor defined message. The host application shall read and process the content of ‘Rx User VID Attention VDM (0x60)’ and ‘Rx User VID Other VDM (0x61)’ registers depending on the generated event.

Figure 9. Advanced Configuration with EC - Interrupt Mask Register

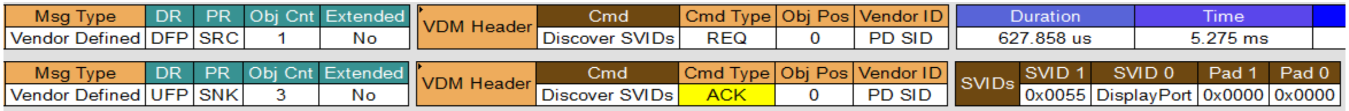


Customer Use
Interrupt Mask for I2C1
Interrupt Mask for I2C2
Global System Configuration
Port Configuration
Port Control
Transmit Source Capabilities
Transmit Sink Capabilities
Autonegotiate Sink
Alternate Mode Entry Queue
PD3 Configuration Register
Event Delay
Transmit Identity Data Object
User Alternate Mode Config
Display Port Capabilities

Error: BIST Message Ignored	□
AM Entry Failure	□
AM Entered	□
Vendor Defined Message Sent	□
Discover Mode Complete	□
Exit Mode Complete	□
User SVID Mode Entered	☑
User SVID Mode Exited	☑
User SVID Attention VDM Received	☑
User SVID Other VDM Received	☑
DP SID Status Update	□
Intel VID Status Update	□

When this port is connected to a PD partner that supports all these modes, the ports negotiate the alternate mode contract, and the PD message exchange between the ports will be as below:

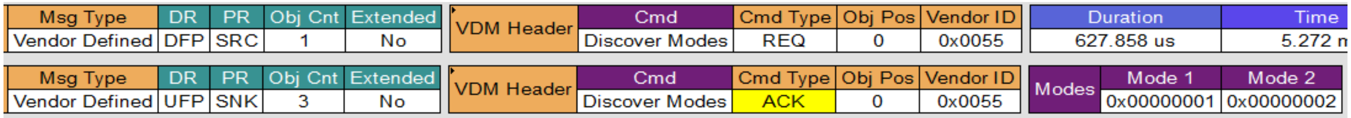
1. The port-partner will share the information about all the SVIDs that it supports in the acknowledgment to the device's 'Discover SVIDs' command.



Msg Type	DR	PR	Obj Cnt	Extended	VDM Header	Cmd	Cmd Type	Obj Pos	Vendor ID	Duration	Time
Vendor Defined	DFP	SRC	1	No		Discover SVIDs	REQ	0	PD SID	627.858 us	5.275 ms

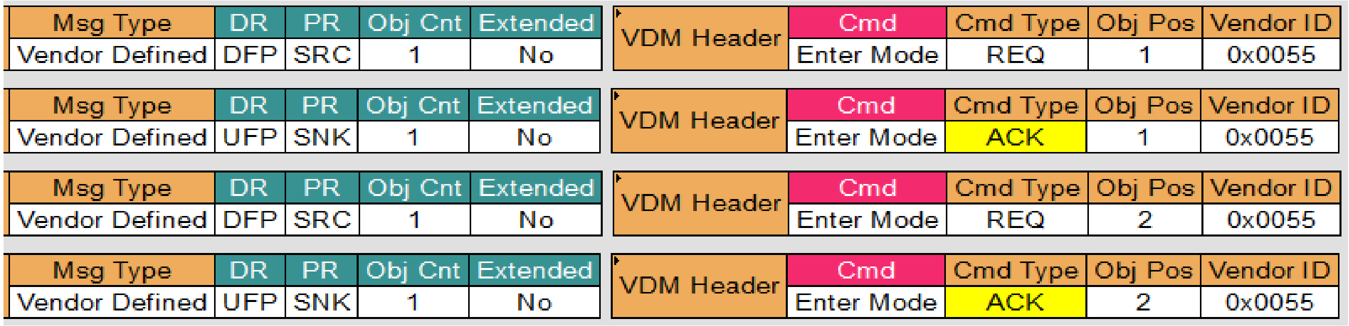
Msg Type	DR	PR	Obj Cnt	Extended	VDM Header	Cmd	Cmd Type	Obj Pos	Vendor ID	SVIDs	SVID 1	SVID 0	Pad 1	Pad 0
Vendor Defined	UFP	SNK	3	No		Discover SVIDs	ACK	0	PD SID		0x0055	DisplayPort	0x0000	0x0000

2. The port partner will share the information about all the modes that it supports for this custom SVID '0x0055' in the acknowledgment to the device's 'Discover Modes' command, and the device automatically enters Mode-1 and Mode-2. The mode numbers returned by the UFP correspond to the 'Mode Value' field of the configuration register.



Msg Type	DR	PR	Obj Cnt	Extended	VDM Header	Cmd	Cmd Type	Obj Pos	Vendor ID	Duration	Time
Vendor Defined	DFP	SRC	1	No		Discover Modes	REQ	0	0x0055	627.858 us	5.272 n

Msg Type	DR	PR	Obj Cnt	Extended	VDM Header	Cmd	Cmd Type	Obj Pos	Vendor ID	Modes	Mode 1	Mode 2
Vendor Defined	UFP	SNK	3	No		Discover Modes	ACK	0	0x0055		0x00000001	0x00000002



Msg Type	DR	PR	Obj Cnt	Extended	VDM Header	Cmd	Cmd Type	Obj Pos	Vendor ID
Vendor Defined	DFP	SRC	1	No		Enter Mode	REQ	1	0x0055

Msg Type	DR	PR	Obj Cnt	Extended	VDM Header	Cmd	Cmd Type	Obj Pos	Vendor ID
Vendor Defined	UFP	SNK	1	No		Enter Mode	ACK	1	0x0055

Msg Type	DR	PR	Obj Cnt	Extended	VDM Header	Cmd	Cmd Type	Obj Pos	Vendor ID
Vendor Defined	DFP	SRC	1	No		Enter Mode	REQ	2	0x0055

Msg Type	DR	PR	Obj Cnt	Extended	VDM Header	Cmd	Cmd Type	Obj Pos	Vendor ID
Vendor Defined	UFP	SNK	1	No		Enter Mode	ACK	2	0x0055

The device generates an interrupt on mode entry/exit and on receiving user defined attention/non-attention message – The host application shall read 'User VID Status (0x57)', 'Rx User VID Attention VDM (0x60)' and 'Rx User VID Other VDM (0x61)' registers depending on the generated events and process the content.

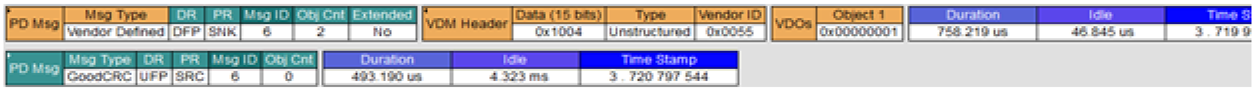
The below example code demonstrates how the events shall be used for the host application:

```
/*
 * I2Cx_IRQ Handler
 */
static int32_t ProcessEvent()
{
    s_TPS_intevent  *pSetEvent      = NULL;
    s_TPS_intevent  *pClrEvent      = NULL;

    uint8_t     outdata[MAX_BUF_BSIZE] = {0};
    uint8_t     indata[MAX_BUF_BSIZE]  = {0};

    int32_t     retVal = -1;

    retVal = ReadReg(REG_ADDR_INTEVENT1, REG_LEN_INTEVENT1, &outdata[0]);
    ASSERT_ON_ERROR(retVal);
    pSetEvent = (s_TPS_intevent*)((uint8_t*)&outdata[1]);
    pClrEvent = (s_TPS_intevent *)&indata[0];

    if(0 != pSetEvent->uservidaltmodeentered)
    {
        SignalEvent(APP_EVENT_USER_AM_ENTERED);
        pClrEvent->uservidaltmodeentered = 1;
    }

    if(0 != pSetEvent->uservidaltmodeothervdm)
    {
        SignalEvent(APP_EVENT_UVDM_RCVD);
        pClrEvent->uservidaltmodeothervdm = 1;
    }

    if(0 != pSetEvent->uservidaltmodeattnvdm)
    {
        SignalEvent(APP_EVENT_ATTN_RCVD);
        pClrEvent->uservidaltmodeattnvdm = 1;
    }

    if(0 != pSetEvent->uservidaltmodeexited)
    {
        SignalEvent(APP_EVENT_USER_AM_EXITED);
        pClrEvent->uservidaltmodeexited = 1;
    }

    retVal = WriteReg(REG_ADDR_INTCLEAR1, REG_LEN_INTCLEAR1, &indata[0]);
    RETURN_ON_ERROR(retVal);

    return retVal;
}

/*
 * Called by application on receiving 'uservidaltmodeentered' event
 * from the device
 */
static int32_t UserAMEntry()
{
    s_TPS_uservidstatus *p_uservidstatus    = NULL;
    uint8_t outdata[MAX_BUF_BSIZE] = {0};
    int32_t retVal = -1;

    retVal = ReadReg(REG_ADDR_USERVIDSTATUS, REG_LEN_USERVIDSTATUS,&outdata[0]);
    RETURN_ON_ERROR(retVal);

    p_uservidstatus = (s_TPS_uservidstatus *)&outdata[1];

    /*!
     *  Configure application according to the entered mode
     */
    if(ACTIVE == p_uservidstatus->usermode1status)
    {
        /*
         * Application specific configuration #1
         */
    }

    if(ACTIVE == p_uservidstatus->usermode2status)
    {
        /*
         * Application specific configuration #2
         */
    }

    return 0;
}
/*
 * Called by application on receiving 'uservidaltmodeattnvdm' event
 * from the device
 */
static int32_t ProcessRxVDMAttnEvents()
{
    s_TPS_uservidstatus     *p_uservidstatus    = NULL;
    s_TPS_uvdmHeader        *p_uvdmheader       = NULL;
    s_TPS_rxattention       *p_rxattention      = NULL;

    uint8_t     outdata[MAX_BUF_BSIZE] = {0} ;

    uint32_t    rxattentiondo1  = 0;
    uint32_t    rxattentiondo2  = 0;

    int32_t     retVal  = -1;

    /*
     *  Read the contents of received VDM packet
     */
    retVal = ReadReg(REG_ADDR_RXUSERVIDATTENTIONVDM,\
                     REG_LEN_RXUSERVIDATTENTIONVDM, &outdata[0]);
    RETURN_ON_ERROR(retVal);
    /* outdata[0] has size */
    p_rxattention = (s_TPS_rxattention *)(&outdata[1]);
    rxattentiondo1 = p_rxattention->rxattentiondo1;
    rxattentiondo2 = p_rxattention->rxattentiondo2;
    /*
     * User defined UVDM Header - See Table 6-24 of the PD specification
     */
    p_uvdmheader = (s_TPS_uvdmHeader *)rxattentiondo1;

    /*
     *  Application specific implementation
     */

    return retVal;
}

/*
 * Called by application on receiving 'uservidaltmodeothervdm' event
 * from the device
 */
static int32_t ProcessRxVDMEvents()
{
    s_TPS_uservidstatus     *p_uservidstatus    = NULL;
    s_TPS_uvdmHeader        *p_uvdmheader       = NULL;
    s_TPS_rxvdm             *p_rxvdm            = NULL;

    uint8_t     outdata[MAX_BUF_BSIZE] = {0} ;

    uint32_t    rxvdmdo1  = 0;
    uint32_t    rxvdmdo2  = 0;

    int32_t     retVal  = -1;

    /*!
     *  Read the contents of received VDM packet
     */
    retVal = ReadReg(REG_ADDR_RXUSERVIDOTHERVDM,\
                     REG_LEN_RXUSERVIDOTHERVDM, &outdata[0]);
    RETURN_ON_ERROR(retVal);
    /* outdata[0] has size */
    p_rxvdm = (s_TPS_rxvdm *)(&outdata[1]);
    rxvdmdo1 = p_rxvdm->rxvdmdo1;
    rxvdmdo2 = p_rxvdm->rxvdmdo2;

    /*
     * User defined UVDM Header - See Table 6-24 of the PD specification
     */
    p_uvdmheader = (s_TPS_uvdmHeader *)rxvdmdo1;

    /*
     * Application specific implementation
     */

    return retVal;
}
/*
 * Example code showning how the device could be commanded (using VDMs)
 * to send a unstructured message to the far-end.
 * Application Specific Example - Switch-1 triggers Port-A to send
 * unstructured VDM command to Port-B for toggling LED1
 */
static int32_t Switch1Event(void)
{
    s_TPS_uvdmHeader    *p_uvdmheader  = NULL;
    s_TPS_vdms  vdmsInData  = {0};

    uint8_t     outdata[MAX_BUF_BSIZE] = {0} ;

    uint32_t    uvdmheader  =  0;
    int32_t     retVal  = -1;

    UART_PRINT(" SW1 - Command the far-end to drive a GPIO\n\r");

    uvdmheader = vdmsInData.vdmheader;
    p_uvdmheader = (s_TPS_uvdmHeader *)&uvdmheader;

    p_uvdmheader->cmdtype   = REQ;
    p_uvdmheader->mode      = Mode_1;
    p_uvdmheader->command   = TOGGLE_LED;
    p_uvdmheader->totalvdos = 1;
    p_uvdmheader->vdmtype   = UNSTRUCTURED_VDM;
    p_uvdmheader->svid      = USER_SVID;

    vdmsInData.numdos       = 2;       /* (userheader.totalvdos) + 1 */
    vdmsInData.soptarget    = SOP;
    vdmsInData.vdmheader    = uvdmheader;
    vdmsInData.vdo2         = LED1;

    retVal = ExecCmd(VDMs, sizeof(s_TPS_vdms), (int8_t *)&vdmsInData,\
                     TASK_RETURN_STATUS_LEN, &outdata[0]);
    RETURN_ON_ERROR(retVal);
    if(0 != outdata[1])
    {
        UART_PRINT("[%d]: Operation Failed.!\n", outdata[1]);
        return -1;
    }

    return 0;
}

/*
```

```
 * Example code showning how Port-B shall process the received
 * unstructured VDM
 * Application Specific Example - Port-B received the 'Toggle-LED' command
 * from Port-A, and below snippet processes it
 */
static int32_t Mode1Events() /* Like 'ProcessRxVDMEvents' above */
{
    s_TPS_status        *p_status_reg   = NULL;
    s_TPS_rxvdm         *p_rxvdm_reg    = NULL;
    s_TPS_uvdmHeader    *p_uvdmheader   = NULL;
    s_TPS_vdms          vdmsInData      = {0};

    uint8_t     outdata[MAX_BUF_BSIZE] = {0} ;

    uint32_t    uvdmheader  =  0;
    int32_t     retVal      = -1;
    int32_t     dataRole    = -1;

    UART_PRINT("Received UDVM Mode-1 Event - Process it\n\r");

    retVal = ReadReg(REG_ADDR_STATUS, REG_LEN_STATUS, &outdata[0]);
    p_status_reg = (s_TPS_status *)(&outdata[1]);
    dataRole = p_status_reg->datarole ;
    /* DFP sent the command, and UFP is processing it here in this example */
    if(UFP_DATA_ROLE == dataRole)
    {
        retVal = ReadReg(REG_ADDR_RXVDM, REG_LEN_RXVDM, &outdata[0]);
        p_rxvdm_reg = (s_TPS_rxvdm *)(&outdata[1]);
        /*
         * 'rxvdmdo2' was populated w/ LED1 in function 'Switch1Event' above
         * 'rxvdmdo1' contains VDM header - Application can interpret as
         * type 's_TPS_uvdmHeader' and ensure the received command is
         * TOGGLE_LED'
         */
        if(LED1 == p_rxvdm_reg->rxvdmdo2)
        {
            GPIO_IF_LedToggle(LED1);
        }

        /*
         * ACK the incoming message.!
         */
        p_uvdmheader    = (s_TPS_uvdmHeader *)&uvdmheader;
        p_uvdmheader->cmdtype   = ACK;
        p_uvdmheader->mode      = Mode_1;
        p_uvdmheader->command   = TOGGLE_LED;
        p_uvdmheader->totalvdos = 1;
        p_uvdmheader->vdmtype   = UNSTRUCTURED_VDM;
        p_uvdmheader->svid      = USER_SVID;
        vdmsInData.numdos       = 1;
        vdmsInData.soptarget    = SOP;
        vdmsInData.vdmheader    = (int32_t)uvdmheader;
        retVal = ExecCmd(VDMs, sizeof(s_TPS_vdms), (int8_t *)&vdmsInData,\
                         TASK_RETURN_STATUS_LEN, &outdata[0]);
        RETURN_ON_ERROR(retVal);
    }

    return 0;
}
```

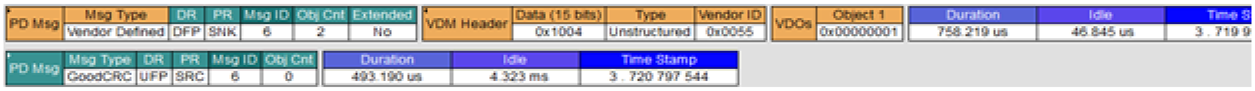
The PD message exchanges between Port-A and Port-B when the above example code is executed on the host application(s) is as below:

1. Port-A sends 'TOGGLE_LED' command to Port-B with 'Object 1' as 'LED1', and Port-B acknowledges the request



PD Msg	Msg Type	DR	PR	Msg ID	Obj Cnt	Extended	VDM Header	Data (15 bits)	Type	Vendor ID	VDOs	Object 1	Duration	Idle	Time S
	Vendor Defined	DFP	SNK	6	2	No		0x1004	Unstructured	0x0055		0x00000001	758.219 us	46.845 us	3 . 719 9

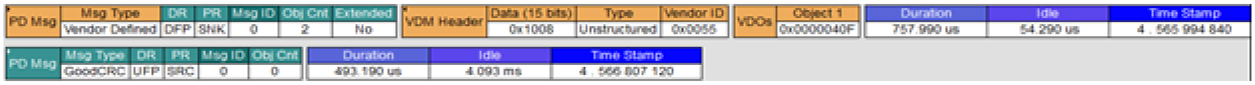
PD Msg	Msg Type	DR	PR	Msg ID	Obj Cnt	Duration	Idle	Time Stamp
	GoodCRC	UFP	SRC	6	0	493.190 us	4.323 ms	3 . 720 797 544



PD Msg	Msg Type	DR	PR	Msg ID	Obj Cnt	Extended	VDM Header	Data (15 bits)	Type	Vendor ID	VDOs	Object 1	Duration	Idle	Time S
	Vendor Defined	DFP	SNK	6	2	No		0x1004	Unstructured	0x0055		0x00000001	758.219 us	46.845 us	3 . 719 9

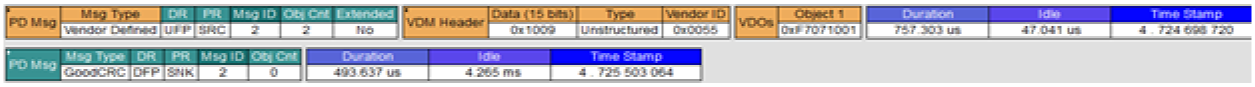
PD Msg	Msg Type	DR	PR	Msg ID	Obj Cnt	Duration	Idle	Time Stamp
	GoodCRC	UFP	SRC	6	0	493.190 us	4.323 ms	3 . 720 797 544

2. Port-A sends 'READ_REG' command to Port-B with 'Object 1' as 'REGISTER-NUMBER', and Port-B responds with its PD firmware version '0xF7070001



PD Msg	Msg Type	DR	PR	Msg ID	Obj Cnt	Extended	VDM Header	Data (15 bits)	Type	Vendor ID	VDOs	Object 1	Duration	Idle	Time Stamp
	Vendor Defined	DFP	SNK	0	2	No		0x1008	Unstructured	0x0055		0x0000040F	757.990 us	54.290 us	4 . 565 994 840

PD Msg	Msg Type	DR	PR	Msg ID	Obj Cnt	Duration	Idle	Time Stamp
	GoodCRC	UFP	SRC	0	0	493.190 us	4.093 ms	4 . 566 807 120



PD Msg	Msg Type	DR	PR	Msg ID	Obj Cnt	Extended	VDM Header	Data (15 bits)	Type	Vendor ID	VDOs	Object 1	Duration	Idle	Time Stamp
	Vendor Defined	UFP	SRC	2	2	No		0x1009	Unstructured	0x0055		0xF7071001	757.303 us	47.041 us	4 . 724 698 720

PD Msg	Msg Type	DR	PR	Msg ID	Obj Cnt	Duration	Idle	Time Stamp
	GoodCRC	DFP	SNK	2	0	493.637 us	4.265 ms	4 . 725 503 064

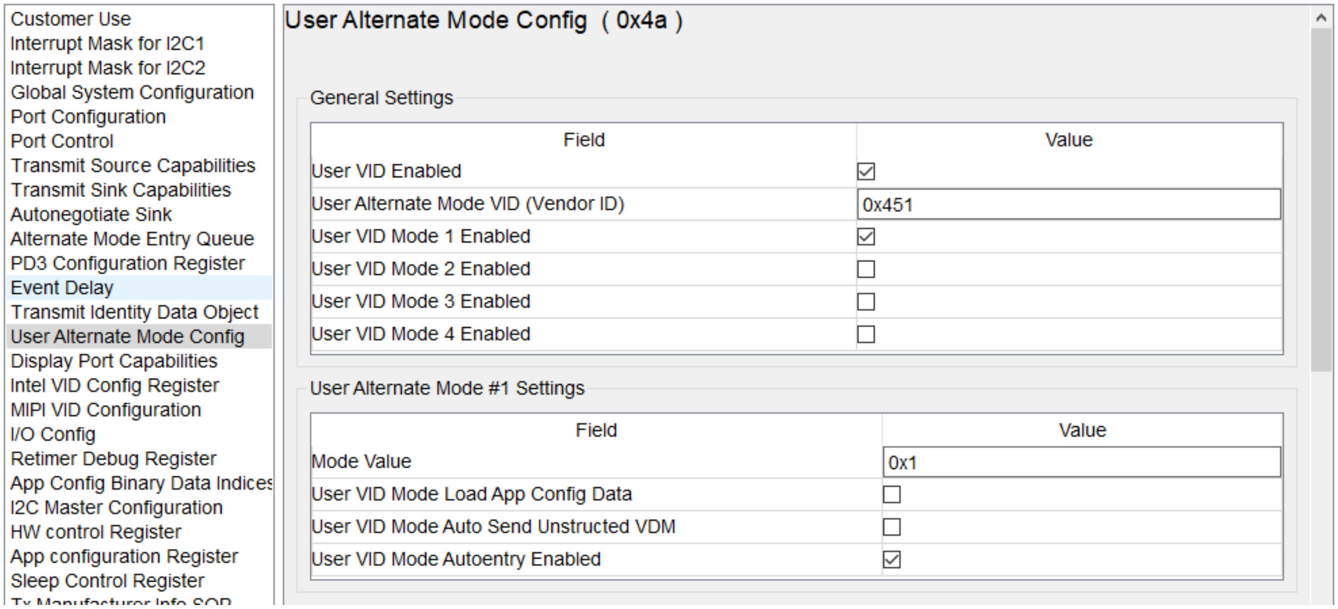
5.2 Example 2

The previous generation PD controllers from TI (TPS65981, TPS65982 and so forth) had support for PDIO Alternate Mode which allows users to transmit or receive up to four unique digital signals between two systems connected through USB Type-C. The support for this alternate mode is removed from this variant of the device as vendors have an option to implement PDIO-like functionality using user alternate mode as detailed in this section.

The device will however need to support TI’s PDIO mode to inter-operate with the earlier generation devices supporting this feature. This example lists the steps for implementing this feature using user alternate modes. This mode uses both structured and unstructured VDM to exchange proprietary information with their port partner.

The example in this section configures the device to support TI SVID ‘0x0451' with one alternate modes. The alternate modes is enabled and its mode values is 0x1/TI-PDIO. Auto-Entry is enabled for this mode.

Figure 10. Advanced Configuration with EC - User Alternate Mode Register


Customer Use
Interrupt Mask for I2C1
Interrupt Mask for I2C2
Global System Configuration
Port Configuration
Port Control
Transmit Source Capabilities
Transmit Sink Capabilities
Autonegotiate Sink
Alternate Mode Entry Queue
PD3 Configuration Register
Event Delay
Transmit Identity Data Object
User Alternate Mode Config
Display Port Capabilities
Intel VID Config Register
MIPI VID Configuration
I/O Config
Retimer Debug Register
App Config Binary Data Indices
I2C Master Configuration
HW control Register
App configuration Register
Sleep Control Register
Tx Manufacturer Info SOP

User Alternate Mode Config (0x4a)

General Settings

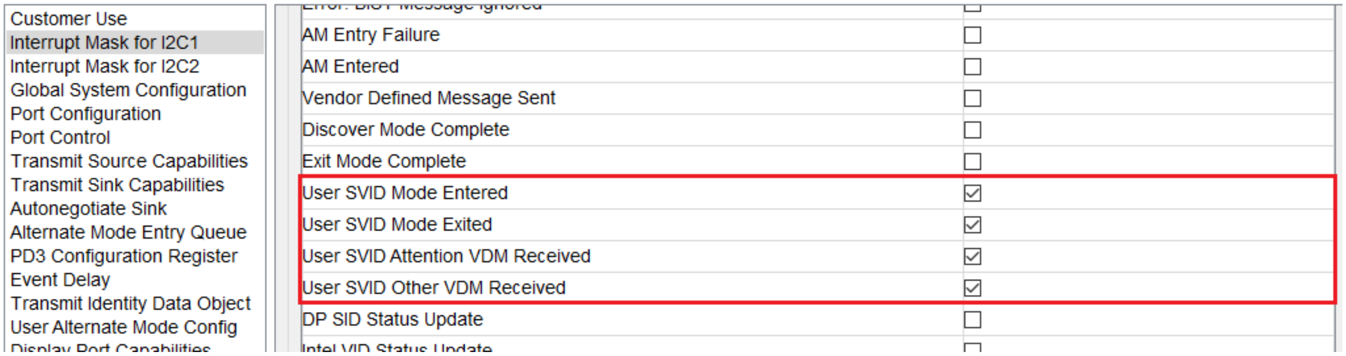
Field	Value
User VID Enabled	☑
User Alternate Mode VID (Vendor ID)	0x451
User VID Mode 1 Enabled	☑
User VID Mode 2 Enabled	☐
User VID Mode 3 Enabled	☐
User VID Mode 4 Enabled	☐

User Alternate Mode #1 Settings

Field	Value
Mode Value	0x1
User VID Mode Load App Config Data	☐
User VID Mode Auto Send Unstructed VDM	☐
User VID Mode Autoentry Enabled	☑

The device is also configured to generate below events and notify the host on mode entry/exit and the reception of the vendor defined message. The host application shall read and process the content of ‘Rx User VID Attention VDM (0x60)’ and ‘Rx User VID Other VDM (0x61)’ registers depending on the generated event.

Figure 11. Advanced Configuration with EC - Interrupt Mask Register

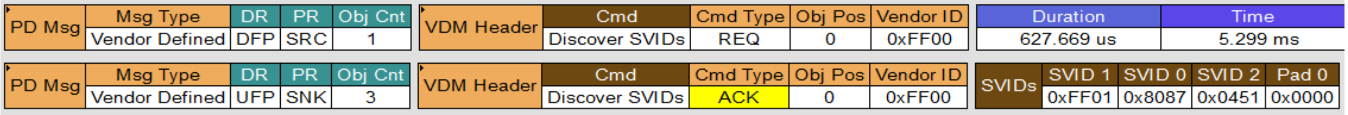


Customer Use
Interrupt Mask for I2C1
Interrupt Mask for I2C2
Global System Configuration
Port Configuration
Port Control
Transmit Source Capabilities
Transmit Sink Capabilities
Autonegotiate Sink
Alternate Mode Entry Queue
PD3 Configuration Register
Event Delay
Transmit Identity Data Object
User Alternate Mode Config
Display Port Capabilities

Error: BIST Message Ignored	□
AM Entry Failure	□
AM Entered	□
Vendor Defined Message Sent	□
Discover Mode Complete	□
Exit Mode Complete	□
User SVID Mode Entered	☑
User SVID Mode Exited	☑
User SVID Attention VDM Received	☑
User SVID Other VDM Received	☑
DP SID Status Update	□
Intel VID Status Update	□

When this port is connected to a PD partner that supports legacy TI-PDIO mode, the ports negotiate the alternate mode contract, and the PD message exchange between the ports will be as below:

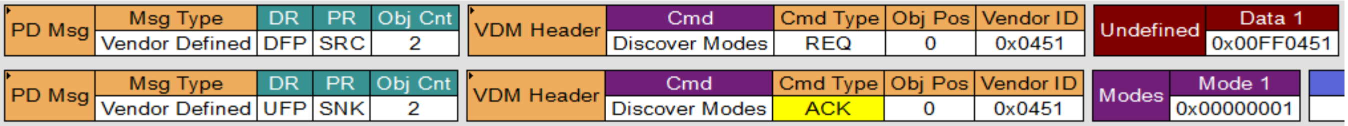
1. The port-partner will share the information about all the SVIDs that it supports in the acknowledgment to the device's 'Discover SVIDs' command



PD Msg	Msg Type	DR	PR	Obj Cnt	VDM Header	Cmd	Cmd Type	Obj Pos	Vendor ID	Duration	Time
	Vendor Defined	DFP	SRC	1		Discover SVIDs	REQ	0	0xFF00	627.669 us	5.299 ms

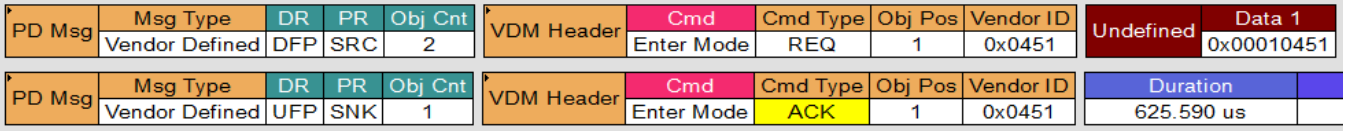
PD Msg	Msg Type	DR	PR	Obj Cnt	VDM Header	Cmd	Cmd Type	Obj Pos	Vendor ID	SVIDs	SVID 1	SVID 0	SVID 2	Pad 0
	Vendor Defined	UFP	SNK	3		Discover SVIDs	ACK	0	0xFF00		0xFF01	0x8087	0x0451	0x0000

2. The port partner will share the information about all the modes that it supports TI-SVID '0x0451' in the acknowledgment to the device's 'Discover Modes' command, and the device automatically enters TI-PDIO mode if its supported by the port partner



PD Msg	Msg Type	DR	PR	Obj Cnt	VDM Header	Cmd	Cmd Type	Obj Pos	Vendor ID	Undefined	Data 1
	Vendor Defined	DFP	SRC	2		Discover Modes	REQ	0	0x0451		0x00FF0451

PD Msg	Msg Type	DR	PR	Obj Cnt	VDM Header	Cmd	Cmd Type	Obj Pos	Vendor ID	Modes	Mode 1
	Vendor Defined	UFP	SNK	2		Discover Modes	ACK	0	0x0451		0x00000001



PD Msg	Msg Type	DR	PR	Obj Cnt	VDM Header	Cmd	Cmd Type	Obj Pos	Vendor ID	Undefined	Data 1
	Vendor Defined	DFP	SRC	2		Enter Mode	REQ	1	0x0451		0x00010451

PD Msg	Msg Type	DR	PR	Obj Cnt	VDM Header	Cmd	Cmd Type	Obj Pos	Vendor ID	Duration
	Vendor Defined	UFP	SNK	1		Enter Mode	ACK	1	0x0451	625.590 us

The device generates an interrupt on mode entry/exit and on receiving user defined attention/non-attention message – The host application shall read 'User VID Status (0x57)', 'Rx User VID Attention VDM (0x60)' and 'Rx User VID Other VDM (0x61)' registers depending on the generated events and process the content.

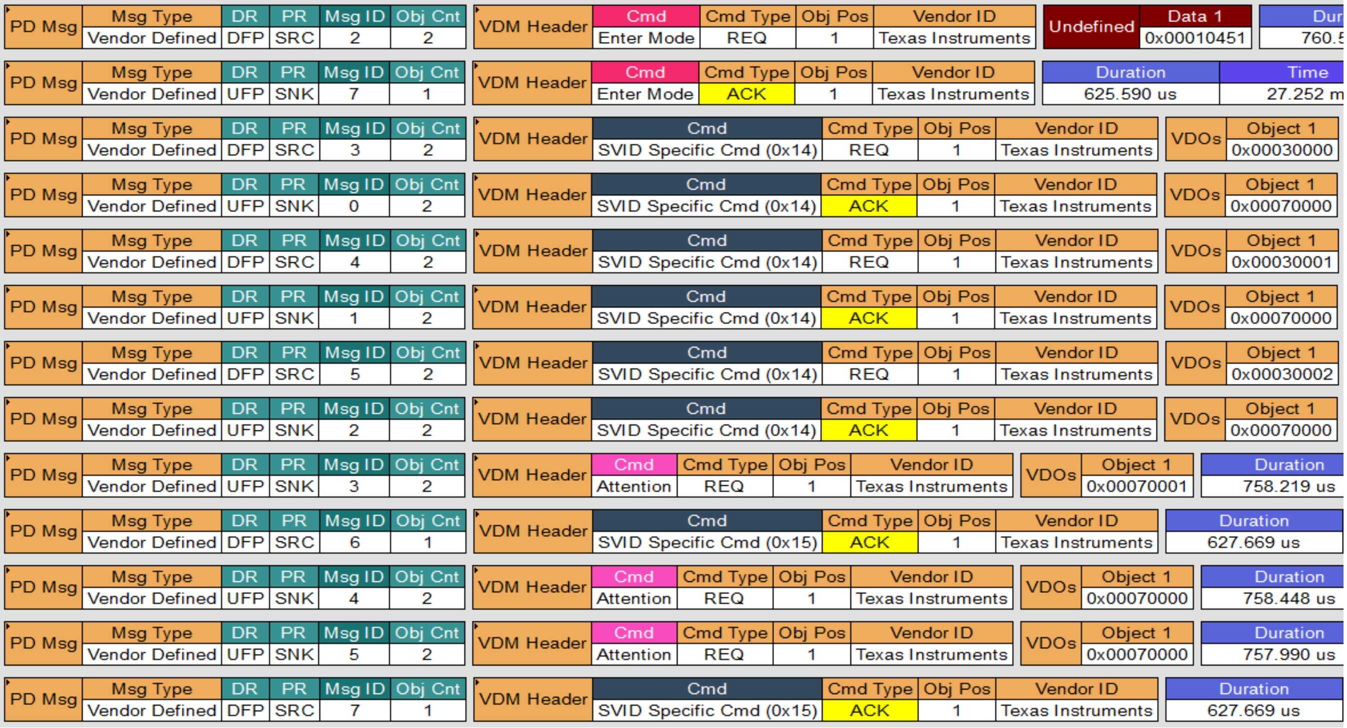
The below example code demonstrates how the events shall be used for the host application:

```
/*
 * I2Cx_IRQ Handler
 */
static int32_t ProcessEvent()
{
    s_TPS_intevent  *pSetEvent      = NULL;
    s_TPS_intevent  *pClrEvent      = NULL;

    uint8_t     outdata[MAX_BUF_BSIZE] = {0};
    uint8_t     indata[MAX_BUF_BSIZE]  = {0};

    int32_t     retVal = -1;

    retVal = ReadReg(REG_ADDR_INTEVENT1, REG_LEN_INTEVENT1, &outdata[0]);
    ASSERT_ON_ERROR(retVal);
    pSetEvent = (s_TPS_intevent*)((uint8_t*)&outdata[1]);
    pClrEvent = (s_TPS_intevent *)&indata[0];
    if(0 != pSetEvent->uservidaltmodeentered)
    {
        SignalEvent(APP_EVENT_USER_AM_ENTERED);
        pClrEvent->uservidaltmodeentered = 1;
    }

    if(0 != pSetEvent->uservidaltmodeothervdm)
    {
        SignalEvent(APP_EVENT_UVDM_RCVD);
        pClrEvent->uservidaltmodeothervdm = 1;
    }

    if(0 != pSetEvent->uservidaltmodeattnvdm)
    {
        SignalEvent(APP_EVENT_ATTN_RCVD);
        pClrEvent->uservidaltmodeattnvdm = 1;
    }

    if(0 != pSetEvent->uservidaltmodeexited)
    {
        SignalEvent(APP_EVENT_USER_AM_EXITED);
        pClrEvent->uservidaltmodeexited = 1;
    }

    retVal = WriteReg(REG_ADDR_INTCLEAR1, REG_LEN_INTCLEAR1, &indata[0]);
    RETURN_ON_ERROR(retVal);

    return retVal;
}

/*
 * Called by application on receiving 'uservidaltmodeentered' event
 * from the device
 */
static int32_t UserAMEntry()
{
    s_TPS_uservidstatus *p_uservidstatus    = NULL;
    uint8_t outdata[MAX_BUF_BSIZE] = {0};
    int32_t retVal = -1;

    retVal = ReadReg(REG_ADDR_USERVIDSTATUS,REG_LEN_USERVIDSTATUS,&outdata[0]);
    RETURN_ON_ERROR(retVal);

    p_uservidstatus = (s_TPS_uservidstatus *)&outdata[1];

    /*!
     * Check if User Alternate Mode 1 is entered.
     * Send PDIO Status to far-end on entering the mode
     * if the port's data-role is DFP - Not shown here.!
     */
    if(ACTIVE == p_uservidstatus->usermode1status)
    {
        UART_PRINT("User Alternate Mode - Mode 1 entered.\n\r");
        retVal = SendPDIOStatus();
        RETURN_ON_ERROR(retVal);
    }

    return 0;
}
/*
 * Sends PDIO status to the far-end/UFP on entering the mode
 */
static int32_t SendPDIOStatus()
{
    s_TPS_vdmheadersstruct  *p_vdmheader    = NULL;
    s_TPS_vdms              vdmsInData      = {0};

    uint8_t     outdata[MAX_BUF_BSIZE]  = {0};

    uint32_t    vdmheader    =  0;
    int32_t     retVal  = -1;

    UART_PRINT("Send PDIO Status\n\r");

    vdmheader = vdmsInData.vdmheader;
    p_vdmheader = (s_TPS_vdmheadersstruct *)&vdmheader;

    p_vdmheader->command = SVDM_SendPDIO_Status;    //0x14
    p_vdmheader->commandtype = CMD_TYPE_REQ;        //0x0
    p_vdmheader->objpos = 0x1;
    p_vdmheader->structuredvdmversion = 0x1;
    p_vdmheader->vdmtype = 0x1;
    p_vdmheader->svid = TI_SVID;    //0x0451

    vdmsInData.numdos       = 2;
    vdmsInData.vdmheader    = vdmheader;
    /*
     * PDIO_IN<x> is 1 for enable, and 0 for disable
     * #define PDIO_IN_EVENTS ((PDIO_IN3 << 3) | (PDIO_IN2 << 2) |
     *                         (PDIO_IN1 << 1) |(PDIO_IN0 << 0))
     */
    vdmsInData.vdo2         = ((PDIO_IN_EVENTS) << 16);

    retVal = ExecCmd(VDMs, sizeof(s_TPS_vdms), (int8_t *)&vdmsInData,\
                     TASK_RETURN_STATUS_LEN, (int8_t *)&outdata[0]);
    RETURN_ON_ERROR(retVal);

    return retVal;
}
/*
 * SwitchEvtHandler, ProcessPDIOInEvents and SendTxPDIOStatus demonstrate
 * how PDIO_IN<x> status shall be sent to the far-end as DFP
 */
static int32_t SwitchEvtHandler(void)
{
    s_AppContext *const pCtx = &gAppCtx;
    e_BoardSwitch   switchstate = 0;

    switchstate = GPIO_IF_SwitchStatus();
    pCtx->switchstate = switchstate;
    GPIO_IF_SwitchIntDisable();

    ProcessPDIOInEvents();

    GPIO_IF_SwitchIntEnable();
    return 0;
}
/**/
static int32_t ProcessPDIOInEvents()
{
    s_AppContext *const pCtx = &gAppCtx;
    int32_t retVal = -1;

    if(SWITCH1 == (pCtx->switchstate & SWITCH1))
    {
        retVal = SendTxPDIOStatus(0x1);
        RETURN_ON_ERROR(retVal);
    }
    if(SWITCH2 == (pCtx->switchstate & SWITCH2))
    {
        retVal = SendTxPDIOStatus(0x2);
        RETURN_ON_ERROR(retVal);
    }

    return retVal;
}
/**/
static int32_t SendTxPDIOStatus(uint8_t switchstate)
{
    s_TPS_vdmheadersstruct  *p_vdmheader    = NULL;
    s_TPS_vdms              vdmsInData      = {0};

    uint8_t     outdata[MAX_BUF_BSIZE]  = {0};

    uint32_t    vdmheader    =  0;
    int32_t     retVal  = -1;

    UART_PRINT("Send TxPDIO Status\n\r");

    vdmheader = vdmsInData.vdmheader;
    p_vdmheader = (s_TPS_vdmheadersstruct *)&vdmheader;

    p_vdmheader->command = SVDM_SendPDIO_Status;    //0x14
    p_vdmheader->commandtype = CMD_TYPE_REQ;        //0x0
    p_vdmheader->objpos = 0x1;
    p_vdmheader->structuredvdmversion = 0x1;
    p_vdmheader->vdmtype = 0x1;
    p_vdmheader->svid = TI_SVID;    //0x0451

    vdmsInData.numdos       = 2;
    vdmsInData.vdmheader    = vdmheader;
    /*
     * Send the PDIO_IN status to far-end.
     * SW1 is PDIO_IN0/Bit0, SW2 is PDIO_IN1/Bit1 of vdo2
     */
    vdmsInData.vdo2         = (((PDIO_IN_EVENTS) << 16) | switchstate);

    retVal = ExecCmd(VDMs, sizeof(s_TPS_vdms), (int8_t *)&vdmsInData,\
                     TASK_RETURN_STATUS_LEN, (int8_t *)&outdata[0]);
    RETURN_ON_ERROR(retVal);

    return retVal;
}
/*
 * Called by application on receiving 'uservidaltmodeattnvdm' event
 * from the device.
 * This fucntion processes the PDIO message sent by far-end/UFP
 */
static int32_t ProcessAttnEvents()
{
    s_TPS_uservidstatus     *p_uservidstatus    = NULL;
    s_TPS_vdmheadersstruct  *p_vdmheader        = NULL;
    s_TPS_rxattention       *p_rxattention      = NULL;

    uint8_t     outdata[MAX_BUF_BSIZE] = {0} ;

    uint32_t    rxattentiondo1  = 0;
    uint32_t    rxattentiondo2  = 0;

    int32_t     retVal  = -1;

    /*!
     *  Read the contents of received VDM packet
```

```
         */
        retVal = ReadReg(REG_ADDR_RXUSERVIDATTENTIONVDM,
                         REG_LEN_RXUSERVIDATTENTIONVDM, &outdata[0]);
        RETURN_ON_ERROR(retVal);
        /* outdata[0] has length */
        p_rxattention = (s_TPS_rxattention *)(&outdata[1]);
        rxattentiondo1 = p_rxattention->rxattentiondo1;
        rxattentiondo2 = p_rxattention->rxattentiondo2;

        /*!
         *  Check whether the VDM Rx is for SVID of User Alternate Mode
         *  In this case, TI_SVID is used for User Alternate Mode
         *  Note : The SVID of User Alternate Mode may differ
         */
        p_vdmheader = (s_TPS_vdmheadersstruct *)&rxattentiondo1;
        if( (TI_SVID != p_vdmheader->svid) ||
            (1 == p_rxattention->rxattentionnumvalid) )
        {
            UART_PRINT("\n\nProcess Attn - Error1.");
            return 0 ;
        }

        /*!
         *  Check for which Mode is VDM Rx, depending on that,
         *  Call the function that will execute the events.
         */
        retVal = ReadReg(REG_ADDR_USERVIDSTATUS,REG_LEN_USERVIDSTATUS,&outdata[0]);
        RETURN_ON_ERROR(retVal);

        p_uservidstatus = (s_TPS_uservidstatus *)&outdata[1];
        if(ACTIVE == p_uservidstatus->usermode1status)
        {
            /*
             * Toggling LED here, but application shall interpret rxattentiondo2,
             * and take action per their requirement - Not shown here.!
             * Bit-3:0 indicate which PDIO_IN was set by the far-end
             */
            GPIO_IF_LedToggle(PDIO_OUT0);
            SendRxPDIOStatus();
        }

        UNUSED(rxattentiondo2);
        return retVal;
    }
    /* Send ACK to UFP's TI-SVID-Attention */
    static int32_t SendRxPDIOStatus()
    {
        s_TPS_vdmheadersstruct  *p_vdmheader    = NULL;
        s_TPS_vdms              vdmsInData      = {0};

        uint8_t     outdata[MAX_BUF_BSIZE]  = {0};

        uint32_t    vdmheader    =  0;
        int32_t     retVal  = -1;

        UART_PRINT("Send RxPDIO Status\n\r");

        vdmheader = vdmsInData.vdmheader;
        p_vdmheader = (s_TPS_vdmheadersstruct *)&vdmheader;

        p_vdmheader->command = SVDM_RxPDIO_Status;    //0x15
        p_vdmheader->commandtype = CMD_TYPE_ACK;      //0x1
        p_vdmheader->objpos = 0x1;
        p_vdmheader->structuredvdmversion = 0x1;
        p_vdmheader->vdmtype = 0x1;
        p_vdmheader->svid = TI_SVID;    //0x0451
        vdmsInData.numdos       = 1;
        vdmsInData.vdmheader    = vdmheader;

        retVal = ExecCmd(VDMs, sizeof(s_TPS_vdms), (int8_t *)&vdmsInData,\
                         TASK_RETURN_STATUS_LEN, (int8_t *)&outdata[0]);
        RETURN_ON_ERROR(retVal);

        return retVal;
    }
```

The PD message exchanges between Port-A and Port-B when the above example code is executed on the host application(s) is shown below. The logs snippets show the ports exchanging status messages after entering TI-PDIO mode indicating which PDIO_IN are enabled on either sides. Port-A/DFP then sends two 'REQ' with 'Object 1' as 0x30001/PDIO_IN0 and 0x30002/PDIO_IN1 to Port-B/UFP, and Port-B acknowledges these message. Then, Port-B/UFP sends 'Attention' with 'Object 1' as 0x70001/PDIO_IN0, and Port-A acknowledges this message.

The example code and log snippets presented in this section assume that the port enters a PD contract as a DFP. If the port is UFP, the host application shall take care of sending 'Attention'/0x06 message (and not 'SVID Specific Cmd'/0x14) to indicate the port-partner about its PDIO_IN status.



PD Msg					VDM Header				VDOs	Undefined		
Msg Type	DR	PR	Msg ID	Obj Cnt	Cmd	Cmd Type	Obj Pos	Vendor ID	Object 1	Data 1	Duration	Time
Vendor Defined	DFP	SRC	2	2	Enter Mode	REQ	1	Texas Instruments		0x00010451	760.5	
Vendor Defined	UFP	SNK	7	1	Enter Mode	ACK	1	Texas Instruments			625.590 us	27.252 m
Vendor Defined	DFP	SRC	3	2	SVID Specific Cmd (0x14)	REQ	1	Texas Instruments	0x00030000			
Vendor Defined	UFP	SNK	0	2	SVID Specific Cmd (0x14)	ACK	1	Texas Instruments	0x00070000			
Vendor Defined	DFP	SRC	4	2	SVID Specific Cmd (0x14)	REQ	1	Texas Instruments	0x00030001			
Vendor Defined	UFP	SNK	1	2	SVID Specific Cmd (0x14)	ACK	1	Texas Instruments	0x00070000			
Vendor Defined	DFP	SRC	5	2	SVID Specific Cmd (0x14)	REQ	1	Texas Instruments	0x00030002			
Vendor Defined	UFP	SNK	2	2	SVID Specific Cmd (0x14)	ACK	1	Texas Instruments	0x00070000			
Vendor Defined	UFP	SNK	3	2	Attention	REQ	1	Texas Instruments	0x00070001		758.219 us	
Vendor Defined	DFP	SRC	6	1	SVID Specific Cmd (0x15)	ACK	1	Texas Instruments			627.669 us	
Vendor Defined	UFP	SNK	4	2	Attention	REQ	1	Texas Instruments	0x00070000		758.448 us	
Vendor Defined	UFP	SNK	5	2	Attention	REQ	1	Texas Instruments	0x00070000		757.990 us	
Vendor Defined	DFP	SRC	7	1	SVID Specific Cmd (0x15)	ACK	1	Texas Instruments			627.669 us	

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale (www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265