USB Power Delivery Software Framework(PSF) User Guide

Version 0.92

Table of Contents

1 Revision History

REV	DATE	DESCRIPTION OF CHANGE
0.92	11-Dec-19	Initial Revision

2 Introduction

USB Power Delivery Software Framework (PSF) with USB-PD Port Controller UPD350 is an effective USB-PD solution compliant to USB-PD 3.0 Specification. It is MCU-agnostic as it is designed to run on different target SoCs. Versatility towards different HW platform is achieved through flexibility towards configurability and easy integration of PSF to different platforms. It provides a high level configurable options to configure it for various PD functionalities like Power role, Data role.

This documents serves a User guide covering

- PSF overview and its architecture
- PSF stack directory structure
- Supported/Not supported features & messages
- Configuration Options available
- Requirements for integration of PSF to new platform
- Steps to integrate PSF

2.1 Feature Overview

Following are the key features of PSF,

- Compliant to USB Power Delivery 3.0 Specification Revision 1.2 & USB Type-C Specification Revision 1.3

- Multi port support up to 4.

- USB-PD Source-only or Sink-only port specific configurability

- FW update through CC support compliant to PD FW Update Specification Revision 1.0 .

2.2 References

- Microchip UPD350 Datasheet
- USB Power Delivery 3.0 Specification Revision 1.2
- USB Type-C Specification Revision 1.3
- PD FW Update Specification Revision 1.0

2.3 Terms and Abbreviations

Term	Definition
USB-PD	USB Power Delivery
SPI	Serial Peripheral Interface bus - Full-duplex bus utilizing a single-master/multi-slave architecture. SPI is a 4-wire bus consisting of SPI CLK, SPI MOSI, SPI MISO and SPI SS

UPD350	Microchip UPD350 Power Delivery Port Controller
Sink Role	USB Type-C Port which sinks power from its Port Partner
Source Role	USB Type-C Port which sources power to its Port Partner
USER_APPLICATION	The software platform that runs on SoC where the PSF is integrated
SoC	system-on-chip
FW	Firmware
PDFW	Power Delivery Firmware update

3 Legal Information

Software License Agreement

Copyright © [2019] Microchip Technology Inc. and its subsidiaries.

Subject to your compliance with these terms, you may use Microchip software and any derivatives exclusively with Microchip products. It is your responsibility to comply with third party license terms applicable to your use of third party software (including open source software) that may accompany Microchip software.

THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE.

IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE, INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.

3

4 PSF Architecture Overview

PSF is a generic USB-PD solution with UPD350. The architecture and configurability of PSF is designed in such a way that it doesn't limit the port functionality. It also provides flexibility for future enhancements by the user.
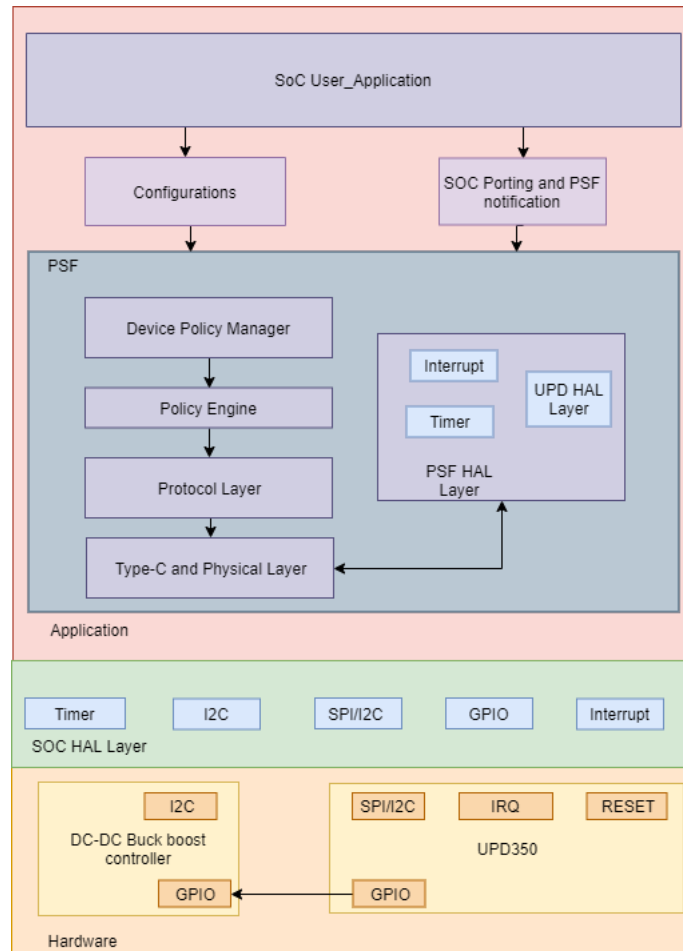
SoC User_Application:

- SoC User_Application interacts with PSF only through "Configuration (PSF_Config.h)" and "SoC Porting and PSF notification (PSF_APIHooks.h)".

SoC HAL Layer:

- It has HAL drivers for modules required for PSF functionality like Timer, Interrupt, SPI/I2C Interface and GPIO.

Device Policy Manager:

The Device Policy Manager is responsible for the following,

- The DPM role is to maintain the Local Policy of the device.
- Controls the Source/Sink in the device.

- For a USB-PD Source, it monitors Source's present capabilities & in case of any change in the capability it triggers notification to Policy Engine.

- For a USB-PD Sink, it evaluates & respond to capability related requests from the Policy Engine for a given port.

- Controls the USB-C Port Control module for each Port.

- It interfaces with Policy Engine Port specifically informs Policy Engine Cable/Device detection as Source/Sink.

- Takes care of any VBUS or VCONN power fault

Policy Engine:

There is one Policy Engine instance per Port that interacts with the Device Policy Manager to implement present Local

Policy for that Port.

- Policy Engine drives message sequences for various operations.

- It is responsible for establishing Explicit Contract by negotiating power based on Local Policy.

Protocol Layer:

- Protocol Layer handles message construction, transmission, & reception, reset operation, message error handling.

- It sends/receives messages through UPD350 using SPI wrapper functions.

- It acts as an interface between Policy Engine & UPD350.

Type-C Connector Management:

Type-C Management includes following operation as defined in the USB Type-C v1.3 specification,

- Source-to-Sink attach/detach detection

- Plug orientation/cable twist detection

- Initial power (Source-to-Sink) detection and enabling PD communication

- VBUS Detection & Type C Error Recovery Mechanism

Port Power Management:

Port power management as a Source/Sink for multiple ports supports,

- Configurable Fixed PDOs per port up to 100W

- Over current sense on ports

PSF HAL Layer:

Apart from the layers specified by USB-PD Specification for PSF, PSF has HAL layer to interact and access UPD350 HW as well as for software functionality like software timer. Interrupt handles all the external interrupt from UPD350 Silicon. Timer involves handling of all the active software timer's timeouts based on the interrupt from periodic hardware timer.

Hardware:

For driving higher voltages, an external DC-DC buck boost controller is required. It can either be driven by UPD350 PIOs or SoC's I2C controlled. UPD350 requires SoC to control its SPI/I2C communication bus for interaction, IRQ and RESET lines.

5 Supported/Non Supported PD Features

List of supported and non supported USB-PD features by PSF.

Features	Supported/Not Supported
USB-PD Source only	Supported
USB-PD Sink only	Supported
Power negotiation up to 100 watts	Supported
PDFU support through CC	Supported
VCONN Power support	Supported
Extended message support via Chunking	Supported
Fixed PDOs	Supported
PPS	Not Supported
FRS support	Not Supported
UVDM & USB Type-C Bridging	Not Supported
Dynamic power balancing	Not Supported
Dual-Role Power(DRP)	Not Supported
Dual-Role Data(DRD)	Not Supported
Alternate Mode	Not Supported
Unchunked Extended message	Not Supported

5

6 Supported/Non Supported PD Messages

Control Message

USB-PD Control messages	Supported/Not Supported
GoodCRC	Supported
Accept	Supported
Reject	Supported
Ping	Supported
PS_RDY	Supported
Get_Source_Cap (For Sink Role only)	Supported
Get_Sink_Cap (For Source Role only)	Supported
VCONN_Swap	Supported
Wait	Supported
Soft_Reset	Supported
GotoMin	Not Supported
DR_SWAP	Not Supported
PR_SWAP	Not Supported
Get_Source_Cap_Extended	Not Supported
Get_Status	Not Supported
FR_Swap	Not Supported
Get_PPS_Status	Not Supported
Get_Country_Codes	Not Supported

Data Message

USB-PD Data messages	Supported/Not Supported
Source_Capabilities (Source Role only)	Supported
Request	Supported
BIST	Supported
Sink_Capabilities (Sink Role only)	Supported
Battery_Status	Not Supported
Alert	Not Supported
Get_Country_Info	Not Supported
Vendor_Defined	Not Supported

6

Extended Message

USB-PD Extended Message	Supported/Not Supported
Firmware_Update_Request	Supported
Firmware_Update_Response	Supported
Source_Capabilities_Extended	Not Supported
Status	Not Supported
Get_Battery_Cap	Not Supported
Get_Battery_Status	Not Supported
Battery_Capabilities	Not Supported
Manufacturer_Info	Not Supported
Security_Request	Not Supported
Security_Response	Not Supported
PPS_Status	Not Supported
Country_Info	Not Supported
Country_Codes	Not Supported

7 Directory Structure

PSF directory is organized as

- ../Demo - Contains all the power delivery application and PSF configuration files
- ../SOC_Portable - Contains SoC specific integration, PSF APIs porting and drivers files
- ../Source - Contains PSF Source and header file

PSF has two user configurable files to enable user various level of configurability and porting:

- PSF_Config.h -> To configure PSF to different PD features and functionality in a PSF integrated SOC platform
- PSF_APIHook.h -> To port and integrate to any new SoC platform

```
PSF
  Demo
    NewBoard_App
    PSF_EVB_Source
  SOC_Portable
    New_SOC
    SAMD20
  Source
    include
    PD3_0
```
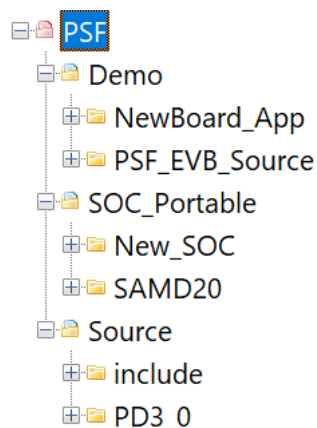
7.1 Source folder

Source folder contains USB-PD 3.0 functionality source files.

It is organized further as

- ..\PSF\Source\include - contains all the header files
- ..\PSF\Source\PD3_0 - contains all the source file for PD3.0 functionality

File descriptions of all the files present under include and PD3_0

Application	File Name	File description
Device Policy Manager(DPM)	policy_manager.c policy_manager.h	Maintains DPM functionality

Policy Engine	policy_engine.h policy_engine.c policy_engine_src.c policy_engine_snk.c	• policy_engine.c maintains PE functionality common to both Source & Sink whereas policy_engine_src.c and policy_engine_snk.c maintains functionality specific to Source and Sink respectively. • policy_engine.h is single header file for all the PE related source files.
Protocol Layer	protocol_layer.c protocol_layer.h	APIs for Protocol layer functionality
Type-C Connector Management	typeC_control.c typeC_control.h	Type-C control functionality & connector management
Timer management	pd_timer.c pd_timer.h	Contains APIs to manage multiple Software Timers
Interrupts Management	upd_interrupts.c upd_interrupts.h	UPD350 Alert Interrupt management (port specific)
Debug support	debug.c debug.h	Debug interface support APIs file
UPD350 Hardware	upd_hw.c upd_hw.h	APIs to access for UPD350 Hardware GPIO, Registers, PIO override
Configurable globals	cfg_globals.c cfg_globals.h	Maintains configurable globals. Globals in this file are configured based on User_application input.
Internal globals	int_globals.c int_globals.h	Maintains globals internal to PSF.
Generic Defines	generic_defs.h	contains generic defines for data types
Standard include file	psf_stdinc.h	Standard include file for PSF
PSF main source file	pd_main.c	contains APIs to be called by User_Application
Power Deliver Firmware update	policy_engine_fwup.c policy_engine_fwup.h	Maintains PDFU functionality
Port Power control	portpower_control.c portpower_control.h	Maintains PSF default port power control support
PSF version	ProjectVersion.h	Maintains PSF release version

7.2 SOC_Portable folder

SOC_Portable contains all the drivers porting files required for PSF functionality from SoC layer. It varies depending on the SoC and user application. List of APIs required for PSF integration and porting is available as defines in PSF_APIHook.h at ..\PSF\SOC_Portable\New_SOC\PSF_APIHook.h.

7.3 Demo folder

Demo folder contains user application demo projects specific to the boards. It contains

- project files
- build procedure documents
- Application and board specific files and documents
- PSF_Config header file.

With these user can build and see a application lively in the demo board. Any PD functionality can be configured using PSF_Config.h .available at ..\Demo\NewBoard_App\PSF_Config.h. User has to configure this file to customize the available PD application.

7

8 PSF Configuration Options

PSF provides configuration flexibility based on required USB-PD feature.

It allows to configure the following,

- Number of ports
- Power Role - Source only or Sink only
- Data Role - DFP or UFP
- Port Power control management
- Source & Sink PDOs
- USB-PD & Type-C Timeouts
- Compile time Code & Data RAM inclusion/exclusion based on USB-PD Specification Revision & Power Role.

All this configurable options are present in PSF_Config.h. header file by PSF. PSF_Config.h shall be defined as per PD application and included as part of User Application Directory. It is recommended to include this file path as one of preprocessor include path. The template of PSF_Config.h for new demo application is available at ..\PSF\Demo\NewBoard_App\

8.1 Include/Exclude Features

Macros

	Name	Description
↦	INCLUDE_PD_3_0	USB-PD V3.0 support code inclusion.
↦	INCLUDE_PD_SOURCE	Source support code inclusion.
↦	INCLUDE_PD_SINK	Sink support code inclusion.
↦	INCLUDE_VCONN_SWAP_SUPPORT	VCONN Support code inclusion.
↦	INCLUDE_POWER_FAULT_HANDLING	Power Fault Hanlding code inclusion.
↦	INCLUDE_UPD_PIO_OVERRIDE_SUPPORT	PIO Override Feature code inclusion.
↦	INCLUDE_POWER_MANAGEMENT_CTRL	Power Management Control Support code inclusion.
↦	INCLUDE_PDFU	PD Firmware update code inclusion.

Description

This parameter is used to configure USB-PD features that is to be included or excluded in stack. User can choose to include/exclude any of the listed features based on the functional requirements. Based on this definition, part of code will be included/excluded for compilation process. This can be used effectively to reduce PSF code size if specific features aren't required.

8.1.1 INCLUDE_PD_3_0

C

```c
#define INCLUDE_PD_3_0 1
```

Description

Setting the INCLUDE_PD_3_0 as 1, enables PSF to include USB Power delivery 3.0 specification features Collision

Avoidance, Extended message support via chunking along with PD 2.0 features at the compile. User can set this define to 0 to reduce code size, if none of the PD enabled ports require PD 3.0 specific features and operates only at PD 2.0 specification.

Remarks

Recommended default value is '1'.

Example

```
#define INCLUDE_PD_3_0  1(Include USB PD 3.0 specific features to PSF)
#define INCLUDE_PD_3_0  0(Exclude USB PD 3.0 specific features from PSF)
```

8.1.2 INCLUDE_PD_SOURCE

C

```
#define INCLUDE_PD_SOURCE 1
```

Description

Setting the INCLUDE_PD_SOURCE as 1, enables PSF to include the USB PD Source functionality at compile time. User can set this define to 0 to reduce code size, if none of the PD enabled ports in the system are configured for Source operation.

Remarks

Recommended default value is '1' for Source Application.

Example

```
#define INCLUDE_PD_SOURCE   1(Include USB PD Source functionality in PSF)
#define INCLUDE_PD_SOURCE   0(Exclude USB PD Source functionality from PSF)
```

8.1.3 INCLUDE_PD_SINK

C

```
#define INCLUDE_PD_SINK 1
```

Description

Setting the INCLUDE_PD_SINK as 1, enables PSF to include USB PD Sink functionality at the compile time. User can set this define to 0 to reduce code size, if none of the PD enabled ports are configured for Sink operation.

Remarks

Recommended default value is '1' for Sink Application.

Example

```
#define INCLUDE_PD_SINK 1(Include USB PD Sink functionality in PSF)
#define INCLUDE_PD_SINK 0(Exclude USB PD Sink functionality from PSF)
```

8.1.4 INCLUDE_VCONN_SWAP_SUPPORT

C

```
#define INCLUDE_VCONN_SWAP_SUPPORT 1
```

Description

Setting the INCLUDE_VCONN_SWAP_SUPPORT as 1, enables PSF to include the VCONN Swap functionality at the compile time. User can set this define to 0 to reduce code size, if none of the PD enabled ports requires VCONN Swap

functionality.

Remarks

Recommended default value is 1.

Example

```
#define INCLUDE_VCONN_SWAP_SUPPORT  1(Include VCONN Swap functionality in PSF)
#define INCLUDE_VCONN_SWAP_SUPPORT  0(Exclude VCONN Swap functionality from PSF)
```

8.1.5 INCLUDE_POWER_FAULT_HANDLING

C

```
#define INCLUDE_POWER_FAULT_HANDLING 1
```

Description

Setting the INCLUDE_POWER_FAULT_HANDLING as 1, enables PSF to handle Power faults (Source and Sink over voltage, Source OCS, Sink under voltage) as per Power Delivery specification Rev3.0 as applicable. User can set this define to 0 to reduce code size, if PSF based power fault handling is not required.

Remarks

Recommended default value is 1.

Example

```
#define INCLUDE_POWER_FAULT_HANDLING    1(Include Power Fault handling to PSF)
#define INCLUDE_POWER_FAULT_HANDLING    0(Exclude Power Fault handling from PSF )
```

8.1.6 INCLUDE_UPD_PIO_OVERRIDE_SUPPORT

C

```
#define INCLUDE_UPD_PIO_OVERRIDE_SUPPORT 1
```

Description

PIO override is UPD350 specific feature which changes the state of a PIO without software intervention. PSF use this feature to disable EN_VBUS instantly on detection of a Power Fault Condition. Setting the INCLUDE_UPD_PIO_OVERRIDE_SUPPORT as 1 enables this feature. User can set this define to 0 to reduce code size of PSF, if PIO override based power faulting is not required.

Remarks

To use this feature, EN_VBUS and FAULT_IN Pin of the system should be UPD350 PIOs. It is also confined to INCLUDE_POWER_FAULT_HANDLING define, thus INCLUDE_POWER_FAULT_HANDLING should be declared as 1 for INCLUDE_UPD_PIO_OVERRIDE_SUPPORT define to be effective. Recommended default value is 1 if UPD350 PIOs are used for EN_VBUS and FAULT_IN.

Example

```
#define INCLUDE_UPD_PIO_OVERRIDE_SUPPORT    1(Include UPD350 PIO Override support for Power
                                               fault to PSF)
#define INCLUDE_UPD_PIO_OVERRIDE_SUPPORT    0(Exclude UPD350 PIO Override support for Power
                                               fault from PSF)
```

8

8.1.7 INCLUDE_POWER_MANAGEMENT_CTRL

C

```
#define INCLUDE_POWER_MANAGEMENT_CTRL 1
```

Description

Setting the INCLUDE_POWER_MANAGEMENT_CTRL as 1, enables PSF to include the functionality that puts the UPD350 into low power mode if UPD350 is inactive for CONFIG_PORT_UPD_IDLE_TIMEOUT_MS time and PSF notifies the same via the call back MCHP_PSF_NOTIFY_CALL_BACK. User can set this define to 0 to reduce code size of the PSF, if low power mode operation of UPD350 is not required for the application.

Remarks

Recommended default value is 1.

Example

```
#define INCLUDE_POWER_MANAGEMENT_CTRL   1(Include power management feature)
#define INCLUDE_POWER_MANAGEMENT_CTRL   0(Exclude power management feature)
```

8.1.8 INCLUDE_PDFU

C

```
#define INCLUDE_PDFU 0
```

Description

Setting the INCLUDE_PDFU as 1, includes the state machine code for PD Firmware Update feature as per USB Power Delivery FW Update Specification v1.0. User can set this define to 0 to reduce code size of the PSF, application doesnot use Firmware update feature.

Remarks

Recommended default value is 0 unless Firmware update feature is used.

Example

```
#define INCLUDE_PDFU    1(Include PDFU feature)
#define INCLUDE_PDFU    0(Exclude PDFU feature)
```

8.2 Power Delivery IDs Configuration

Macros

	Name	Description
⊸	CONFIG_VENDOR_ID	Vendor Identifier value.
⊸	CONFIG_PRODUCT_ID	Product Identifier value.
⊸	CONFIG_HWMAJOR_VERSION	Hardware Major Version.
⊸	CONFIG_HWMINOR_VERSION	Hardware Minor Version.
⊸	CONFIG_SILICON_VERSION	Silicon Base Version.

Description

This section explains the configurable Power delivery IDs and versions mostly used in PDFU descriptors.

8.2.1 CONFIG_VENDOR_ID

C

```
#define CONFIG_VENDOR_ID
```

Description

CONFIG_VENDOR_ID field defines Vendor Identifier value. It is used by the PD Firmware Update state-machine during Enumeration phase. This information is shared with the PDFU Initiator as part of GET_FW_ID command's response.

Remarks

The user definition of this macro is mandatory when INCLUDE_PDFU is defined as '1'.It should always be two byte width. It should always be two byte width.

Example

```
#define CONFIG_VENDOR_ID    0x0424u
```

8.2.2 CONFIG_PRODUCT_ID

C

```
#define CONFIG_PRODUCT_ID
```

Description

CONFIG_PRODUCT_ID is the Product Identifier value. It is used by the PD Firmware Update state-machine during Enumeration phase. This information is shared with the PDFU Initiator as part of GET_FW_ID command's response.

Remarks

The user definition of this macro is mandatory when INCLUDE_PDFU is defined as '1'.It should always be two byte width.

Example

```
#define CONFIG_PRODUCT_ID 0x301Cu
```

8.2.3 CONFIG_HWMAJOR_VERSION

C

```
#define CONFIG_HWMAJOR_VERSION
```

Description

CONFIG_HWMAJOR_VERSION defines Hardware Major Version details of the product. It is used by the PD Firmware Update state-machine during Enumeration phase. This information is shared with the PDFU Initiator as part of GET_FW_ID command's response.

Remarks

This is a 4-bit entity. (Valid values are 0x0 to 0xF). The user definition of this macro is mandatory when INCLUDE_PDFU is defined as '1'.

Example

```
#define CONFIG_HWMAJOR_VERSION    0x1
```

8

8.2.4 CONFIG_HWMINOR_VERSION

C

```
#define CONFIG_HWMINOR_VERSION
```

Description

CONFIG_HWMAJOR_VERSION defines Hardware Minor Version details of the product. It is used by the PD Firmware Update state-machine during Enumeration phase. This information is shared with the PDFU Initiator as part of GET_FW_ID command's response.

Remarks

This is a 4-bit entity. (Valid values are 0x0 to 0xF). The user definition of this macro is mandatory when INCLUDE_PDFU is defined as '1'.

Example

```
#define CONFIG_HWMINOR_VERSION    0x0
```

8.2.5 CONFIG_SILICON_VERSION

C

```
#define CONFIG_SILICON_VERSION
```

Description

CONFIG_SILICON_VERSION is UPD301 Silicon Base Version. It is used by the PD Firmware Update state-machine during Enumeration phase. This information is shared with the PDFU Initiator as part of GET_FW_ID command's response.

Remarks

The user definition of this macro is mandatory when INCLUDE_PDFU is defined as '1'. It should be a byte width.

Example

```
#define CONFIG_SILICON_VERSION    0x01u
```

8.3 System Level Configuration

Macros

	Name	Description
⊸	CONFIG_PD_PORT_COUNT	Power Delivery Enabled ports count.
⊸	CONFIG_DEFINE_UPD350_HW_INTF_SEL	HW Communication interface between SOC and UPD350.

Description

This section explain PSF configurability available at overall system level.

8.3.1 CONFIG_PD_PORT_COUNT

C

```
#define CONFIG_PD_PORT_COUNT 2
```

Description

CONFIG_PD_PORT_COUNT defines the number of Power delivery enabled ports. The maximum number of ports PSF can be configured is '4'.

Remarks

The max and min value for CONFIG_PD_PORT_COUNT is '4' and '1' respectively. PSF refers the Port number in the call backs as 0 to (CONFIG_PD_PORT_COUNT - 1). The default value is 2 and it can be defined based on the user application.

Example

```
#define CONFIG_PD_PORT_COUNT      2 (Number of PD ports enabled in PSF Stack is 2)
```

8.3.2 CONFIG_DEFINE_UPD350_HW_INTF_SEL

C

```
#define CONFIG_DEFINE_UPD350_HW_INTF_SEL
```

Description

CONFIG_DEFINE_UPD350_HW_INTF_SEL defines the Hardware interface for communication between the SOC and UPD350. It can take either CONFIG_UPD350_SPI or CONFIG_UPD350_I2C as value. <b>CONFIG_UPD350_SPI - SPI is the communication interface between SOC and UPD350. SPI interface is supported by UPD350 B and D parts alone. <b>CONFIG_UPD350_I2C - I2C is the communication interface between SOC and UPD350. I2C interface is supported by UPD350 A and C parts alone.

Remarks

CONFIG_DEFINE_UPD350_HW_INTF_SEL should be defined based on UPD350 silicon part used for the application. All the ports in a system should use either I2C supported or SPI supported UPD350 part. Different part for each port, for example SPI supported UPD350 for Port 1 and I2C supported UPD350 part for Port 2 is not supported.

Example

```
#define CONFIG_DEFINE_UPD350_HW_INTF_SEL    CONFIG_UPD350_SPI
#define CONFIG_DEFINE_UPD350_HW_INTF_SEL    CONFIG_UPD350_I2C
```

8.4 Port Specific Configuration

This section cover USB-PD and Type C configuration specific to each port.

8.4.1 Basic Port Configuration

Macros

	Name	Description
⊸	CONFIG_PORT_n_ENDIS	Port Enable disable macro.
⊸	CONFIG_PORT_n_POWER_ROLE	Port's Power Role.
⊸	CONFIG_PORT_n_RP_CURRENT_VALUE	Rp Current value for the port

Description

This section covers basic port configurations like Power role, Type C current configuration option available in PSF.

8.4.1.1 CONFIG_PORT_n_ENDIS

C

```
#define CONFIG_PORT_n_ENDIS 1
```

Description

CONFIG_PORT_n_ENDIS defines whether a port to be enabled or disabled. n can take values between 0 and (CONFIG_PD_PORT_COUNT-1). If CONFIG_PORT_n_ENDIS defined as 1, port is enabled. If CONFIG_PORT_n_ENDIS defined as 0, port is disabled.

Remarks

By default, all ports are enabled.

Example

```
#define CONFIG_PORT_0_ENDIS  0 (Type-C Port 0 Disabled)
#define CONFIG_PORT_0_ENDIS  1 (Type-C Port 0 Enabled)
```

8.4.1.2 CONFIG_PORT_n_POWER_ROLE

C

```
#define CONFIG_PORT_n_POWER_ROLE
```

Description

CONFIG_PORT_n_POWER_ROLE defines the Power role of nth port. n can take values between 0 and (CONFIG_PD_PORT_COUNT-1). Setting CONFIG_PORT_n_POWER_ROLE as 1, configures the nth port as Source or Setting CONFIG_PORT_n_POWER_ROLE as 0, configures the nth port as Sink.

Remarks

The default Data Role for a port is determined based on the Power role configured via this through this define. Data role is configured as 'DFP' for Source and 'UFP' for Sink respectively. By default, all the ports are configured as Source i.e. CONFIG_PORT_0_POWER_ROLE defined as 1.

Example

```
#define CONFIG_PORT_0_POWER_ROLE    1 (Configuring the Port 0 as Source)
#define CONFIG_PORT_0_POWER_ROLE    0 (Configuring the Port 0 as Sink)
```

8.4.1.3 CONFIG_PORT_n_RP_CURRENT_VALUE

C

```
#define CONFIG_PORT_n_RP_CURRENT_VALUE
```

Description

CONFIG_PORT_n_RP_CURRENT_VALUE defines the Rp Value of nth port. n can take values between 0 and (CONFIG_PD_PORT_COUNT-1). CONFIG_PORT_n_RP_CURRENT_VALUE can take following values 0 - Rp termination is disabled; For Sink, CONFIG_PORT_n_RP_CURRENT_VALUE should be set '0' 1 - Rp termination is set to default USB Power 2 - Rp termination is set to 1.5A current 3 - Rp termination is set to 3A current

Remarks

If CONFIG_PORT_n_POWER_ROLE set as 0 (Sink), CONFIG_PORT_n_RP_CURRENT_VALUE should be defined as 0. If CONFIG_PORT_n_POWER_ROLE set as 1 (Source), CONFIG_PORT_n_RP_CURRENT_VALUE should take value other than 0. By default, all the ports are configured as Source & Rp termination set to 3A.

Example

```
#define CONFIG_PORT_0_RP_CURRENT_VALUE  0 (Configuring the Port 0 Rp Value as Disabled)
#define CONFIG_PORT_0_RP_CURRENT_VALUE  1 (Configuring the Port 0 Rp Value as DEFAULT)
```

```
#define CONFIG_PORT_0_RP_CURRENT_VALUE  2 (Configuring the Port 0 Rp Value as CURRENT_15)
#define CONFIG_PORT_0_RP_CURRENT_VALUE  3 (Configuring the Port 0 Rp Value as CURRENT_30)
```

8.4.2 Source Port Configuration

Macros

	Name	Description
⚬	CONFIG_PORT_n_SOURCE_NUM_OF_PDOS	Number of Source PDOs.
⚬	CONFIG_PORT_n_SOURCE_UNCONSTRAINED_PWR	Source PDO's Unconstrained Power field.
⚬	CONFIG_PORT_n_SOURCE_USB_COM	Source PDO's USB Communication capable field.
⚬	CONFIG_PORT_n_SOURCE_USB_SUSPEND	Source PDO's USB Suspend field.
⚬	CONFIG_PORT_n_SOURCE_PDO_x_CURRENT	Source PDO's Current field.
⚬	CONFIG_PORT_n_SOURCE_PDO_x_VOLTAGE	Source PDO's Voltage field.

Description

This section covers all the configuration option available to configure the Source PDOs.

8.4.2.1 CONFIG_PORT_n_SOURCE_NUM_OF_PDOS

C

```
#define CONFIG_PORT_n_SOURCE_NUM_OF_PDOS
```

Description

CONFIG_PORT_n_SOURCE_NUM_OF_PDOS refers to the number PDOs supported by the nth source port. n can take values between 0 and (CONFIG_PD_PORT_COUNT - 1).

Remarks

CONFIG_PORT_n_SOURCE_NUM_OF_PDOS can take only values from 1 to 7. Default value for CONFIG_PORT_n_SOURCE_NUM_OF_PDOS is 1.

Example

```
#define CONFIG_PORT_0_SOURCE_NUM_OF_PDOS     4 (Port 0 has 4 Source PDOs)
```

8.4.2.2 CONFIG_PORT_n_SOURCE_UNCONSTRAINED_PWR

C

```
#define CONFIG_PORT_n_SOURCE_UNCONSTRAINED_PWR
```

Description

CONFIG_PORT_n_SOURCE_UNCONSTRAINED_PWR defines the Unconstrained Power bit in fixed PDO of nth source port. As per PD specification, this field is exposed for PDO1 alone for rest of the fixed PDOs it is Zero. CONFIG_PORT_n_SOURCE_UNCONSTRAINED_PWR can be configured as 0 or 1. n can take values between 0 and (CONFIG_PD_PORT_COUNT - 1).

Remarks

By default, this define is set to 1.

Example

```
#define CONFIG_PORT_0_SOURCE_UNCONSTRAINED_PWR  1 (Port 0 is unconstrained power capable)
#define CONFIG_PORT_0_SOURCE_UNCONSTRAINED_PWR  0 (Port 0 is not unconstrained power
capable)
```

8.4.2.3 CONFIG_PORT_n_SOURCE_USB_COM

C

```
#define CONFIG_PORT_n_SOURCE_USB_COM
```

Description

CONFIG_PORT_n_SOURCE_USB_COM defines the USB communication enable bit in PDO of nth source port. As per PD specification, this field is exposed for PDO1 alone for rest of the fixed PDOs it is Zero. CONFIG_PORT_n_SOURCE_USB_COM can be configured as 0 or 1. n can take values between 0 and (CONFIG_PD_PORT_COUNT - 1).

Remarks

By default, this define is set to 0.

Example

```
#define CONFIG_PORT_0_SOURCE_USB_COM        1 (Port 0 is USB communication capable)
#define CONFIG_PORT_0_SOURCE_USB_COM        0 (Port 0 is not USB communication capable)
```

8.4.2.4 CONFIG_PORT_n_SOURCE_USB_SUSPEND

C

```
#define CONFIG_PORT_n_SOURCE_USB_SUSPEND
```

Description

CONFIG_PORT_n_SOURCE_USB_SUSPEND defines the USB Suspend supported bit in fixed PDO of nth source port. As per PD specification, this field is exposed for PDO1 alone for rest of the fixed PDOs it is Zero. CONFIG_PORT_n_SOURCE_PDO_1_USB_SUSPEND can be configured as 0 or 1. n can take values between 0 and (CONFIG_PD_PORT_COUNT - 1).

Remarks

By default, it is defined as '0'.

Example

```
#define CONFIG_PORT_0_SOURCE_USB_SUSPEND        0 (Port 0 is not USB suspend capable)
#define CONFIG_PORT_0_SOURCE_USB_SUSPEND        0 (Port 0 is USB suspend capable)
```

8.4.2.5 CONFIG_PORT_n_SOURCE_PDO_x_CURRENT

C

```
#define CONFIG_PORT_n_SOURCE_PDO_x_CURRENT
```

Description

CONFIG_PORT_n_SOURCE_PDO_x_CURRENT defines the maximum current value in xth PDO of nth source port. As per PD specification there can be 7 PDOs, thus x takes value from 1 to 7. n can take values between 0 and(CONFIG_PD_PORT_COUNT - 1). This define is expressed in mA units.

Remarks

By default, PDO 1 of all the port is defined as 3000 mA and rest as 0 mA.

Example

```
#define CONFIG_PORT_0_SOURCE_PDO_1_CURRENT        3000 (Maximum current value is configured
                                                        as 3A for PDO1 of Port-0)
```

8.4.2.6 CONFIG_PORT_n_SOURCE_PDO_x_VOLTAGE

C

```
#define CONFIG_PORT_n_SOURCE_PDO_x_VOLTAGE
```

Description

CONFIG_PORT_n_SOURCE_PDO_1_VOLTAGE defines the voltage supported in xth PDO of nth source port. As per PD specification there can be 7 PDOs, So x takes value from 1 to 7. n can take values between 0 and (CONFIG_PD_PORT_COUNT - 1).

Remarks

Units are expressed in milliVolts(mV). It is mandatory to define PDO1 as vSafe5V (5000). By default, PDO 1 of all the port is defined as 5000mV and rest of PDOs voltage as 0 mV.

Example

```
#define CONFIG_PORT_0_SOURCE_PDO_1_VOLTAGE      5000 (PDO1 voltage of Port 1 is 5V)
```

8.4.3 Sink Port Configuration

Macros

	Name	Description
⊸	CONFIG_PORT_n_SINK_NUM_OF_PDOS	Number of Sink PDOs.
⊸	CONFIG_PORT_n_SINK_HIGHER_CAPABILITY	Sink PDO's Higher capability field.
⊸	CONFIG_PORT_n_SINK_UNCONSTRAINED_PWR	Sink PDO's Unconstrained Power field.
⊸	CONFIG_PORT_n_SINK_USB_COM	Sink PDO's USB Communication capable field.
⊸	CONFIG_PORT_n_SINK_USB_SUSPEND	Sink USB suspend field.
⊸	CONFIG_PORT_n_SINK_PDO_x_CURRENT	Sink PDO's Current field.
⊸	CONFIG_PORT_n_SINK_PDO_x_VOLTAGE	Sink PDO's Voltage field.

Description

This section covers all the configuration option available to configure the Sink PDOs.

8.4.3.1 CONFIG_PORT_n_SINK_NUM_OF_PDOS

C

```
#define CONFIG_PORT_n_SINK_NUM_OF_PDOS
```

Description

CONFIG_PORT_n_SINK_NUM_OF_PDOS defines the number PDOs supported by the nth sink port. n can take values between 0 and (CONFIG_PD_PORT_COUNT - 1).

Remarks

CONFIG_PORT_n_SINK_NUM_OF_PDOS can be configured from 1 to 7. Default value for CONFIG_PORT_n_SINK_NUM_OF_PDOS is 1.

Example

```
#define CONFIG_PORT_0_SINK_NUM_OF_PDOS      4 (Port 0 has 4 Sink PDOs)
```

8.4.3.2 CONFIG_PORT_n_SINK_HIGHER_CAPABILITY

C

```
#define CONFIG_PORT_n_SINK_HIGHER_CAPABILITY
```

Description

CONFIG_PORT_n_SINK_HIGHER_CAPABILITY defines the Higher Capability bit in fixed PDO in nth sink port. As per PD specification, this field is exposed for PDO1 alone for rest of the fixed PDOs it is Zero. CONFIG_PORT_0_SINK_HIGHER_CAPABILITY can be configured as 0 or 1. n can take values between 0 and (CONFIG_PD_PORT_COUNT - 1).

Remarks

If the define is set as '1', Sink is higher capability capable & if it is set as '0', Sink is not higher capability capable. The default value is '1'.

Example

```
#define CONFIG_PORT_0_SINK_HIGHER_CAPABILITY        1
```

8.4.3.3 CONFIG_PORT_n_SINK_UNCONSTRAINED_PWR

C

```
#define CONFIG_PORT_n_SINK_UNCONSTRAINED_PWR
```

Description

CONFIG_PORT_n_SINK_UNCONSTRAINED_PWR defines the Unconstrained Power bit in fixed PDO of nth sink port. As per PD specification, this field is exposed for PDO1 alone for rest of the fixed PDOs it is Zero. CONFIG_PORT_n_SINK_UNCONSTRAINED_PWR can be configured as 0 or 1. n can take values between 0 and (CONFIG_PD_PORT_COUNT - 1).

Remarks

By default, this define is set to 0.

Example

```
#define CONFIG_PORT_0_SINK_UNCONSTRAINED_PWR    1 (Port 0 is unconstrained power capable)
#define CONFIG_PORT_0_SINK_UNCONSTRAINED_PWR    0 (Port 0 is not unconstrained power
capable)
```

8.4.3.4 CONFIG_PORT_n_SINK_USB_COM

C

```
#define CONFIG_PORT_n_SINK_USB_COM 0
```

Description

CONFIG_PORT_n_SINK_USB_COM defines the USB communication enable bit in PDO of nth sink port. As per PD specification, this field is exposed for PDO1 alone for rest of the fixed PDOs it is Zero. CONFIG_PORT_n_SINK_USB_COM can be configured as 0 or 1. n can take values between 0 and (CONFIG_PD_PORT_COUNT - 1).

Remarks

By default, this define is set to 0.

Example

```
#define CONFIG_PORT_0_SINK_USB_COM     1 (Port 0 is USB communication capable)
#define CONFIG_PORT_0_SINK_USB_COM     0 (Port 0 is not USB communication capable)
```

8.4.3.5 CONFIG_PORT_n_SINK_USB_SUSPEND

C

```
#define CONFIG_PORT_n_SINK_USB_SUSPEND
```

Description

CONFIG_PORT_n_SINK_USB_SUSPEND defines the USB Suspend supported bit in fixed PDO of nth sink port. As per PD specification, this field is exposed for PDO1 alone for rest of the fixed PDOs it is Zero. CONFIG_PORT_n_SINK_PDO_1_USB_SUSPEND can be configured as 0 or 1. n can take values between 0 and (CONFIG_PD_PORT_COUNT - 1).

Remarks

By default, it is defined as '0'.

Example

```
#define CONFIG_PORT_0_SINK_USB_SUSPEND      0 (Port 0 is not USB suspend capable)
#define CONFIG_PORT_0_SINK_USB_SUSPEND      0 (Port 0 is USB suspend capable)
```

8.4.3.6 CONFIG_PORT_n_SINK_PDO_x_CURRENT

C

```
#define CONFIG_PORT_n_SINK_PDO_x_CURRENT
```

Description

CONFIG_PORT_n_SINK_PDO_x_CURRENT defines the maximum current value in xth PDO of nth Sink port. As per PD specification there can be 7 PDOs, thus x takes value from 1 to 7. n can take values between 0 and(CONFIG_PD_PORT_COUNT - 1). This define is expressed in mA units.

Remarks

By default, PDO 1 of all the port is defined as 3000 mA and rest as 0 mA.

Example

```
#define CONFIG_PORT_0_SINK_PDO_1_CURRENT      3000 (Maximum current value is configured
                                                    as 3A for PDO1 of sink Port-0)
```

8.4.3.7 CONFIG_PORT_n_SINK_PDO_x_VOLTAGE

C

```
#define CONFIG_PORT_n_SINK_PDO_x_VOLTAGE
```

Description

CONFIG_PORT_n_SINK_PDO_1_VOLTAGE defines the voltage supported in xth PDO of nth sink port. As per PD specification there can be 7 PDOs, So x takes value from 1 to 7. n can take values between 0 and (CONFIG_PD_PORT_COUNT - 1).

Remarks

Units are expressed in milliVolts(mV). It is mandatory to define PDO1 as vSafe5V (5000). By default, PDO 1 of all the port is defined as 5000mV and rest of PDOs voltage as 0 mV.

Example

```
#define CONFIG_PORT_0_SINK_PDO_1_VOLTAGE      5000 (PDO1 voltage of sink Port 1 is 5V)
```

8.5 Power Fault Configuration

Macros

	Name	Description
⊸	CONFIG_OVER_VOLTAGE_FACTOR	Over voltage factor.
⊸	CONFIG_UNDER_VOLTAGE_FACTOR	Under Voltage factor.
⊸	CONFIG_FAULT_IN_OCS_DEBOUNCE_MS	Fault In OCS Debounce value in milliseconds.
⊸	CONFIG_VCONN_OCS_ENABLE	VCONN OCS Enable.
⊸	CONFIG_VCONN_OCS_DEBOUNCE_IN_MS	VCONN OCS Debounce value in milliseconds.
⊸	CONFIG_POWER_GOOD_TIMER_MS	Power Good Timer value in milliseconds.
⊸	CONFIG_MAX_VBUS_POWER_FAULT_COUNT	Maximum VBUS Power fault count.
⊸	CONFIG_MAX_VCONN_FAULT_COUNT	Maximum VCONN Power fault count.

Description

This section explains configurations required for Power fault handling by PSF.

8.5.1 CONFIG_OVER_VOLTAGE_FACTOR

C

```
#define CONFIG_OVER_VOLTAGE_FACTOR 1.15
```

Description

CONFIG_OVER_VOLTAGE_FACTOR is percentage of PDO voltage to be considered as Over Voltage for that PDO. As per PD specification desired range for fixed PDO voltage is (0.95 * PDO Voltage) to (1.05 * PDO Voltage), So CONFIG_OVER_VOLTAGE_FACTOR should be greater than the desired range.

Remarks

If 115% of the PDO voltage has to be considered as overvoltage for that PDO voltage, then define CONFIG_OVER_VOLTAGE_FACTOR as 1.15. It is mandatory to define CONFIG_OVER_VOLTAGE_FACTOR when INCLUDE_POWER_FAULT_HANDLING is defined as '1'. Default value for this macro is 1.15 indicating 115%.

Example

```
#define CONFIG_OVER_VOLTAGE_FACTOR        1.15
(CONFIG_PORT_0_SOURCE_PDO_1_VOLTAGE is 5000, then for PDO 1 Over voltage is 5750mV)
```

8.5.2 CONFIG_UNDER_VOLTAGE_FACTOR

C

```
#define CONFIG_UNDER_VOLTAGE_FACTOR 0.85
```

Description

CONFIG_UNDER_VOLTAGE_FACTOR is percentage of PDO voltage to be considered as under Voltage for that PDO. As per PD specification desired range for fixed PDO voltage is (0.95 * PDO Voltage) to (1.05 * PDO Volatge), So CONFIG_OVER_VOLTAGE_FACTOR should be less than the desired range.

Remarks

If 85% of the PDO voltage has to be considered as under voltage for that PDO voltage, then define

8

CONFIG_UNDER_VOLTAGE_FACTOR as 0.85. CONFIG_UNDER_VOLTAGE_FACTOR must be defined when INCLUDE_POWER_FAULT_HANDLING is defined. As an exceptional case this factor is not considered for VSafe5V.

For Source VSafe5V, CONFIG_VSINKDISCONNECT_VOLTAGE is considered as Vsafe5V undervoltage instead of (CONFIG_UNDER_VOLTAGE_FACTOR * TYPEC_VBUS_5V).

For Sink, VSafe5V under voltage is not applicable as when voltage is less than or equal to CONFIG_VSINKDISCONNECT_VOLTAGE, sink becomes disconnected.

The default value for CONFIG_UNDER_VOLTAGE_FACTOR is 0.85 indicating 85%.

Example

```
#define CONFIG_UNDER_VOLTAGE_FACTOR        0.85
(CONFIG_PORT_0_SOURCE_PDO_2_VOLTAGE is 9000, then for PDO 2 Over voltage is 7650mV)
```

8.5.3 CONFIG_FAULT_IN_OCS_DEBOUNCE_MS

C

```
#define CONFIG_FAULT_IN_OCS_DEBOUNCE_MS 5
```

Description

CONFIG_FAULT_IN_OCS_DEBOUNCE_MS is debounce timer value in terms of milliseconds for VBUS overcurrent fault conditions before reacting and entering fault recovery routine. It is applicable only for OCS detection via FAULT_IN configured UPD350 pin.

Remarks

The default debounce for Fault IN OCS detection is 5ms.

Example

```
#define CONFIG_FAULT_IN_OCS_DEBOUNCE_MS        5 (Debounce is 5ms)
```

8.5.4 CONFIG_VCONN_OCS_ENABLE

C

```
#define CONFIG_VCONN_OCS_ENABLE 1
```

Description

PSF uses UPD350 internal comparator to detect VCONN Overcurrent fault. CONFIG_VCONN_OCS_ENABLE is to enable or disable the internal VCONN OCS detection logic. Setting CONFIG_VCONN_OCS_ENABLE as '1' enables the VCONN OCS detection and setting as '0' disables the VCONN OCS detection.

Remarks

The default value for CONFIG_VCONN_OCS_ENABLE is '1'.

Example

```
#define CONFIG_VCONN_OCS_ENABLE 1(Enables VCONN OCS detection)
#define CONFIG_VCONN_OCS_ENABLE 0(Disbales VCONN OCS detection)
```

8.5.5 CONFIG_VCONN_OCS_DEBOUNCE_IN_MS

C

```
#define CONFIG_VCONN_OCS_DEBOUNCE_IN_MS 2
```

Description

CONFIG_VCONN_OCS_DEBOUNCE_IN_MS is debounce timer value in terms of milliseconds for VCONN overcurrent fault conditions before reacting and entering fault recovery routine.

Remarks

The default value for CONFIG_VCONN_OCS_DEBOUNCE_IN_MS is 2ms.

Example

```
#define CONFIG_VCONN_OCS_DEBOUNCE_IN_MS        2 (Debounce is 2ms)
```

8.5.6 CONFIG_POWER_GOOD_TIMER_MS

C

```
#define CONFIG_POWER_GOOD_TIMER_MS MILLISECONDS_TO_TICKS(10000)
```

Description

After an automatic fault recovery, a CONFIG_POWER_GOOD_TIMER_MS is ran to determine whether power remains in a good state for the duration of the timer, then the Fault Counter is reset. If another fault occurs before the Power Good Timer expires, then the Fault Counter is incremented.

For power Source, it is the time a power source must consistently provide power without a power to determine the power is good and a fault condition does not exist. For power Sink, it is the time after the sink established a contract and its consistently drawing power from VBUS without a power fault to determine that power is good and a fault condition does not exist.

Remarks

It shall be expressed in MILLISECONDS_TO_TICKS defines. By default, it is configured to 10Seconds.

Example

```
#define CONFIG_POWER_GOOD_TIMER_MS        MILLISECONDS_TO_TICKS(10000)
```

8.5.7 CONFIG_MAX_VBUS_POWER_FAULT_COUNT

C

```
#define CONFIG_MAX_VBUS_POWER_FAULT_COUNT 3
```

Description

CONFIG_MAX_VBUS_POWER_FAULT_COUNT is the maximum number of back-to-back VBUS faults allowed before shut down of the port. A back-to-back fault is a second fault which occurs within the CONFIG_POWER_GOOD_TIMER_MS after a port is automatically re-enabled from a previous fault condition. During port shutdown due to occurrent fault, the device removes its CC termination and wait for port partner to get detached physically from the port to resume its normal operation.

Remarks

By default, it is configured to count 3.

Example

```
#define CONFIG_MAX_VBUS_POWER_FAULT_COUNT        3
```

8

8.5.8 CONFIG_MAX_VCONN_FAULT_COUNT

C

```
#define CONFIG_MAX_VCONN_FAULT_COUNT 3
```

Description

CONFIG_MAX_VCONN_POWER_FAULT_COUNT is the maximum number of back-to-back VCONN faults allowed before it disables the VCONN. A back-to-back fault is a second fault which occurs within the CONFIG_POWER_GOOD_TIMER_MS after a port is automatically re-enabled from a previous fault condition. If VCONN disabled due to occurrent VCONN power fault, VCONN will be enabled only after a physical detach and re-attach.

Remarks

By default, it is configured to count 3.

Example

```
#define CONFIG_MAX_VCONN_FAULT_COUNT          3
```

8.6 Vsafe5v Configuration for Source and Sink

Macros

	Name	Description
⊸	CONFIG_SRC_VSAFE5V_DESIRED_MAX_VOLTAGE	Vsafe5V Maximum acceptable limit for Source.
⊸	CONFIG_SRC_VSAFE5V_DESIRED_MIN_VOLTAGE	Vsafe5V Minimum acceptable limit for Source.
⊸	CONFIG_SNK_VSAFE5V_DESIRED_MAX_VOLTAGE	Vsafe5V Maximum acceptable limit for Sink.
⊸	CONFIG_SNK_VSAFE5V_DESIRED_MIN_VOLTAGE	Vsafe5V Minimum acceptable limit for Sink.
⊸	CONFIG_VSINKDISCONNECT_VOLTAGE	vSinkDisconnect.

Description

This section explains the configurability available to define VSafe5v threshold range.

8.6.1 CONFIG_SRC_VSAFE5V_DESIRED_MAX_VOLTAGE

C

```
#define CONFIG_SRC_VSAFE5V_DESIRED_MAX_VOLTAGE 5500
```

Description

CONFIG_SRC_VSAFE5V_DESIRED_MAX_VOLTAGE is maximum voltage acceptable for VSafe5V expressed in terms of millivolts for source. The voltage will be considered as valid Vsafe5V only if it is equal to or greater than CONFIG_SRC_VSAFE5V_DESIRED_MIN_VOLTAGE & less than CONFIG_SRC_VSAFE5V_DESIRED_MAX_VOLTAGE. CONFIG_OVER_VOLTAGE_FACTOR * 5000mV will be considered as overvoltage for Vsafe5V for Source.

Valid Vsafe5V condition: CONFIG_SRC_VSAFE5V_DESIRED_MIN_VOLTAGE <= Valid Vsafe5V < CONFIG_SRC_VSAFE5V_DESIRED_MAX_VOLTAGE

Vsafe5V overvoltage condition: VBUS >= CONFIG_OVER_VOLTAGE_FACTOR * 5000mV

Remarks

It is mandatory to define CONFIG_SRC_VSAFE5V_DESIRED_MAX_VOLTAGE. By default, it is defined as 5500 mV. It

8

must be defined in such a way that following condition is met. CONFIG_SRC_VSAFE5V_DESIRED_MAX_VOLTAGE < CONFIG_OVER_VOLTAGE_FACTOR * TYPEC_VBUS_5V.

Example

```
#define CONFIG_SRC_VSAFE5V_DESIRED_MAX_VOLTAGE        5500
```

8.6.2 CONFIG_SRC_VSAFE5V_DESIRED_MIN_VOLTAGE

C

```
#define CONFIG_SRC_VSAFE5V_DESIRED_MIN_VOLTAGE 4750
```

Description

CONFIG_SRC_VSAFE5V_DESIRED_MIN_VOLTAGE is minimum voltage acceptable for VSafe5V expressed in terms of millivolts for source. The voltage will be considered as valid Vsafe5V only if it is equal to or greater than CONFIG_SRC_VSAFE5V_DESIRED_MIN_VOLTAGE & less than CONFIG_SRC_VSAFE5V_DESIRED_MAX_VOLTAGE.

Valid Vsafe5V condition: CONFIG_SRC_VSAFE5V_DESIRED_MIN_VOLTAGE <= Valid Vsafe5V < CONFIG_SRC_VSAFE5V_DESIRED_MAX_VOLTAGE

Remarks

It is mandatory to define CONFIG_SRC_VSAFE5V_DESIRED_MIN_VOLTAGE. The default value for this macro is 4750mV. It must be defined in such a way that following condition is met: CONFIG_SRC_VSAFE5V_DESIRED_MIN_VOLTAGE > CONFIG_VSINKDISCONNECT_VOLTAGE.

Example

```
#define CONFIG_SRC_VSAFE5V_DESIRED_MIN_VOLTAGE        4750
```

8.6.3 CONFIG_SNK_VSAFE5V_DESIRED_MAX_VOLTAGE

C

```
#define CONFIG_SNK_VSAFE5V_DESIRED_MAX_VOLTAGE 5500
```

Description

CONFIG_SNK_VSAFE5V_DESIRED_MAX_VOLTAGE is maximum voltage acceptable for VSafe5V expressed in terms of millivolts for sink. The voltage will be considered as valid Vsafe5V only if it is equal to or greater than CONFIG_SNK_VSAFE5V_DESIRED_MIN_VOLTAGE & less than CONFIG_SNK_VSAFE5V_DESIRED_MAX_VOLTAGE. CONFIG_OVER_VOLTAGE_FACTOR * 5000mV will be considered as overvoltage for Vsafe5V for sink.

Valid Vsafe5V condition: CONFIG_SNK_VSAFE5V_DESIRED_MAX_VOLTAGE > Valid Vsafe5V <= CONFIG_SNK_VSAFE5V_DESIRED_MIN_VOLTAGE Overvoltage condition: Vsafe5V >= CONFIG_OVER_VOLTAGE_FACTOR * 5000

Remarks

It is mandatory to define CONFIG_SNK_VSAFE5V_DESIRED_MAX_VOLTAGE. The default value for this macro is 5500mV. It must be defined in such a way that following condition is met. CONFIG_SNK_VSAFE5V_DESIRED_MAX_VOLTAGE < CONFIG_OVER_VOLTAGE_FACTOR * TYPEC_VBUS_5V.

Example

```
#define CONFIG_SNK_VSAFE5V_DESIRED_MAX_VOLTAGE        5500
```

8.6.4 CONFIG_SNK_VSAFE5V_DESIRED_MIN_VOLTAGE

C

```
#define CONFIG_SNK_VSAFE5V_DESIRED_MIN_VOLTAGE 4400
```

Description

CONFIG_SNK_VSAFE5V_DESIRED_MIN_VOLTAGE is minimum voltage acceptable for VSafe5V expressed in terms of millivolts for Sink. The voltage will be considered as valid Vsafe5V only if it is equal to or greater than CONFIG_SNK_VSAFE5V_DESIRED_MIN_VOLTAGE & less than CONFIG_SNK_VSAFE5V_DESIRED_MAX_VOLTAGE.

Valid Vsafe5V condition: CONFIG_SNK_VSAFE5V_DESIRED_MAX_VOLTAGE > Valid Vsafe5V <= CONFIG_SNK_VSAFE5V_DESIRED_MIN_VOLTAGE

Remarks

It is mandatory to define CONFIG_SRC_VSAFE5V_DESIRED_MIN_VOLTAGE. By default, it is defined as 4400mV. It must be defined in such a way that following condition is met. CONFIG_SNK_VSAFE5V_DESIRED_MIN_VOLTAGE > CONFIG_VSINKDISCONNECT_VOLTAGE.

Example

```
#define CONFIG_SNK_VSAFE5V_DESIRED_MIN_VOLTAGE        4400
```

8.6.5 CONFIG_VSINKDISCONNECT_VOLTAGE

C

```
#define CONFIG_VSINKDISCONNECT_VOLTAGE 3670
```

Description

CONFIG_VSINKDISCONNECT_VOLTAGE is the vSinkDisconnect mentioned in Type c specification v1.3. Specification defines it as threshold used for transition from Attached.SNK to Unattached.SNK. In PSF, CONFIG_VSINKDISCONNECT_VOLTAGE is considered as undervoltage for Vsafe5V in case of source. For Sink, if the voltage is below CONFIG_VSINKDISCONNECT_VOLTAGE, it is considered as VBUS disconnect.

For Sink: If Voltage <= CONFIG_VSINKDISCONNECT_VOLTAGE, then Sink disconnected For Source: If Voltage <= CONFIG_VSINKDISCONNECT_VOLTAGE, then Source undervoltage

Remarks

By default, it is defined as 3.67V

Example

```
#define CONFIG_VSINKDISCONNECT_VOLTAGE        3670
```

8.7 MCU Idle Timeout Configuration

Macros

	Name	Description
⊸	CONFIG_PORT_UPD_IDLE_TIMEOUT_MS	UPD350 Idle Timeout value in milliseconds.

8.7.1 CONFIG_PORT_UPD_IDLE_TIMEOUT_MS

C

```
#define CONFIG_PORT_UPD_IDLE_TIMEOUT_MS MILLISECONDS_TO_TICKS(15000)
```

Description

CONFIG_PORT_UPD_IDLE_TIMEOUT_MS is the idle time after which UPD350 is put to low power mode by the power management control if there is no activity or interrupt in UPD350.

Remarks

It shall be expressed in MILLISECONDS_TO_TICKS define. CONFIG_PORT_UPD_IDLE_TIMEOUT_MS is valid only if INCLUDE_POWER_MANAGEMENT_CTRL set as 1. By default, this define is set to to 15seconds.

Example

```
#define CONFIG_PORT_UPD_IDLE_TIMEOUT_MS      MILLISECONDS_TO_TICKS(15000) (Timeout is 15
seconds)
```

8.8 DC-DC Buck Boost Controller Configuration

Macros

	Name	Description
⊸	CONFIG_DCDC_CTRL	DC DC Buck Boost Controller default configuration option.

Description

The section explains the option to configure DC-DC control.

8.8.1 CONFIG_DCDC_CTRL

C

```
#define CONFIG_DCDC_CTRL PWRCTRL_DEFAULT_PSF_GPIO_CONFIG
```

Description

CONFIG_DCDC_CTRL is to define the default DC-DC control provided by the PSF stack. If CONFIG_DCDC_CTRL defined as PWRCTRL_DEFAULT_PSF_GPIO_CONFIG, default GPIO based DC-DC controller is used. If left undefined, default stack's DC-DC control option is not used. User has to config via Power control APIs provided by the stack.

Remarks

None.

Example

```
#define CONFIG_DCDC_CTRL     PWRCTRL_DEFAULT_PSF_GPIO_CONFIG (Uses default GPIO based DC-DC
contol)
#define CONFIG_DCDC_CTRL     (If undefined, Default DC DC control provided by stack is not
used)
```

8

8.8.2 GPIO Based DC-DC Control Configuration

Enumerations

	Name	Description
◆	eFAULT_IN_MODE_TYPE	UPD350 Fault_IN GPIO mode enum.
◆	eUPD_OUTPUT_PIN_MODES_TYPE	UPD350 GPIO Output mode enum.

Macros

	Name	Description
⊸	CONFIG_PORT_n_UPD_FAULT_IN_MODE	UPD350 Fault_In PIO mode.
⊸	CONFIG_PORT_n_UPD_FAULT_IN_PIO_NO	UPD350 Fault_In PIO.
⊸	CONFIG_PORT_n_UPD_DC_DC_EN_PIO_MODE	"DC DC Enable" UPD350 PIO mode.
⊸	CONFIG_PORT_n_UPD_DC_DC_EN_PIO_NO	"DC DC Enable" UPD350 PIO.
⊸	CONFIG_PORT_n_UPD_EN_VBUS	"Enable VBUS" UPD350 PIO.
⊸	CONFIG_PORT_n_UPD_EN_VBUS_PIO_MODE	"Enable VBUS" UPD350 PIO mode.
⊸	CONFIG_PORT_n_UPD_VBUS_DIS_PIO_MODE	"VBUS Discharge" UPD350 PIO mode.
⊸	CONFIG_PORT_n_UPD_VBUS_DIS_PIO_NO	"VBUS Discharge" UPD350 PIO.
⊸	CONFIG_PORT_n_UPD_VSELx_PIO_MODE	"VBUS Voltage Selection" UPD350 PIO mode.
⊸	CONFIG_PORT_n_UPD_VSELx_PIO_NO	"VBUS Voltage Selection" UPD350 PIO.
⊸	CONFIG_PORT_n_VSAFE0V_VSEL_MAPPING	VSEL Truth table for VSafe0V voltage drive.
⊸	CONFIG_PORT_n_PDO_x_VSEL_MAPPING	VSEL Truth table for PDO voltage drive.

Description

The section describes the various configurable option available GPIO based DC-DC control.

8.8.2.1 eFAULT_IN_MODE_TYPE Enumeration

C

```
typedef enum {
    eFAULT_IN_ACTIVE_LOW = 0x20U,
    eFAULT_IN_ACTIVE_HIGH = 0x10U,
    eFAULT_IN_ACTIVE_LOW_PU = 0xA0U,
    eFAULT_IN_ACTIVE_HIGH_PD = 0x50U
} eFAULT_IN_MODE_TYPE;
```

Description

eFAULT_IN_MODE_TYPE enum defines the various combination modes applicable for UPD350 GPIO in input mode.

Members

Members	Description
eFAULT_IN_ACTIVE_LOW = 0x20U	Active low input signal
eFAULT_IN_ACTIVE_HIGH = 0x10U	Active high input signal
eFAULT_IN_ACTIVE_LOW_PU = 0xA0U	Active low signal with internal pull up
eFAULT_IN_ACTIVE_HIGH_PD = 0x50U	Active high signal with internal pull down

Remarks

None

8.8.2.2 eUPD_OUTPUT_PIN_MODES_TYPE Enumeration

C

```
typedef enum {
    ePUSH_PULL_ACTIVE_HIGH = 0x0CU,
    ePUSH_PULL_ACTIVE_LOW = 0x04U,
    eOPEN_DRAIN_ACTIVE_HIGH = 0x08U,
    eOPEN_DRAIN_ACTIVE_LOW = 0x00U,
    eOPEN_DRAIN_ACTIVE_HIGH_PU = 0x88U,
    eOPEN_DRAIN_ACTIVE_LOW_PU = 0x80U
} eUPD_OUTPUT_PIN_MODES_TYPE;
```

Description

eUPD_OUTPUT_PIN_MODES_TYPE enum defines the various combination modes applicable for UPD350 GPIO in output mode.

Members

Members	Description
ePUSH_PULL_ACTIVE_HIGH = 0x0CU	Active High output signal
ePUSH_PULL_ACTIVE_LOW = 0x04U	Active low output signal
eOPEN_DRAIN_ACTIVE_HIGH = 0x08U	Active High Open Drain output signal
eOPEN_DRAIN_ACTIVE_LOW = 0x00U	Active Low Open Drain output signal
eOPEN_DRAIN_ACTIVE_HIGH_PU = 0x88U	Active High Open Drain output signal with internal pull up
eOPEN_DRAIN_ACTIVE_LOW_PU = 0x80U	Active Low Open Drain output signal with internal pull up

Remarks

None

8.8.2.3 CONFIG_PORT_n_UPD_FAULT_IN_MODE

C

```
#define CONFIG_PORT_n_UPD_FAULT_IN_MODE eFAULT_IN_ACTIVE_LOW
```

Description

CONFIG_PORT_n_UPD_FAULT_IN_MODE defines the PIO mode of the UPD350 PIO FAULT_IN defined in CONFIG_PORT_n_UPD_FAULT_IN_PIO_NO. It takes only values from enum eFAULT_IN_MODE_TYPE. n can take values between 0 and (CONFIG_PD_PORT_COUNT-1).

Remarks

By default, it is configured to ePUSH_PULL_ACTIVE_HIGH.

Example

```
#define CONFIG_PORT_n_UPD_FAULT_IN_MODE          eFAULT_IN_ACTIVE_LOW
            (FAULT_IN is configured as Active low in input mode)
```

8.8.2.4 CONFIG_PORT_n_UPD_FAULT_IN_PIO_NO

C

```
#define CONFIG_PORT_n_UPD_FAULT_IN_PIO_NO eUPD_PIO5
```

Description

CONFIG_PORT_n_UPD_FAULT_IN_PIO_NO defines the UPD PIO used as FAULT_IN pin. FAULT_IN detects over-current

fault or under/over-voltage fault from external sensing device based on its configuration CONFIG_PORT_n_UPD_FAULT_IN_MODE. It takes value from 0 to 15 and to disable the pin functionality from stack, user can define it as 0xFF. It is applicable only when CONFIG_DCDC_CTRL is defined as PWRCTRL_DEFAULT_PSF_GPIO_CONFIG and INCLUDE_POWER_FAULT_HANDLING defined as '1'.

Remarks

By default, it is defined as 5.n can take values between 0 and (CONFIG_PD_PORT_COUNT-1).

Example

```
#define CONFIG_PORT_0_UPD_FAULT_IN           5 (FAULT_IN is PIO5)
#define CONFIG_PORT_0_UPD_FAULT_IN           0xFF (FAULT_IN functinality disabled)
```

8.8.2.5 CONFIG_PORT_n_UPD_DC_DC_EN_PIO_MODE

C

```
#define CONFIG_PORT_n_UPD_DC_DC_EN_PIO_MODE ePUSH_PULL_ACTIVE_HIGH
```

Description

CONFIG_PORT_n_UPD_DC_DC_EN_PIO_MODE defines the PIO mode of the UPD350 PIO DC_DC_EN defined in CONFIG_PORT_n_UPD_DC_DC_EN_PIO_NO. It takes only values from enum eUPD_OUTPUT_PIN_MODES_TYPE. n can take values between 0 and (CONFIG_PD_PORT_COUNT-1).

Remarks

By default, it is configured to ePUSH_PULL_ACTIVE_HIGH.

Example

```
#define CONFIG_PORT_0_UPD_DC_DC_EN_PIO_MODE        ePUSH_PULL_ACTIVE_HIGH
(DC_DC_EN is configured as Active signal in output mode)
```

8.8.2.6 CONFIG_PORT_n_UPD_DC_DC_EN_PIO_NO

C

```
#define CONFIG_PORT_n_UPD_DC_DC_EN_PIO_NO eUPD_PIO6
```

Description

CONFIG_PORT_n_UPD_DC_DC_EN_PIO_NO defines the UPD350 PIO to enable DC-DC controller. It is asserted as per CONFIG_PORT_n_UPD_DC_DC_EN_PIO_MODE during initialization and de-asserted during error condition to reset the DC-DC controller n can take values between 0 and (CONFIG_PD_PORT_COUNT-1). It takes value from 0 to 15 and to disable the pin functionality from stack, user can define it as 0xFF. It is applicable only when CONFIG_DCDC_CTRL is defined as PWRCTRL_DEFAULT_PSF_GPIO_CONFIG.

Remarks

By default, it is configured to PIO6. User can also use stack's enum eUPD_PIO_NUM_TYPE to define this.

Example

```
#define CONFIG_PORT_0_UPD_VBUS_DIS_PIO_NO        6 (DC_DC_EN is PIO6)
#define CONFIG_PORT_0_UPD_VBUS_DIS_PIO_NO        0xFF (DC_DC_EN functinality disabled)
```

8.8.2.7 CONFIG_PORT_n_UPD_EN_VBUS

C

```
#define CONFIG_PORT_n_UPD_EN_VBUS eUPD_PIO3
```

Description

CONFIG_PORT_n_EN_VBUS_UPD_PIO refers to the UPD350 PIO number used for EN_VBUS pin functionality for the nth Port. EN_VBUS is to enable VBUS drive out of DC-DC controller. EN_VBUS pin connects to a load switch device such as a

power FET or load switch IC. It is driven as per CONFIG_PORT_n_UPD_EN_VBUS_PIO_MODE configuration mode whenever stack requires VBUS to driven high as well as low. n can take values between 0 and CONFIG_PD_PORT_COUNT - 1. It takes value from 0 to 15 and to disable the pin functionality from stack, user can define it as 0xFF. It is applicable only when CONFIG_DCDC_CTRL is defined as PWRCTRL_DEFAULT_PSF_GPIO_CONFIG and for Source operation only. By defining INCLUDE_UPD_PIO_OVERRIDE_SUPPORT as '1', The PIO Override feature of the UPD350 shall be utilized in this pin to ensure that fast and autonomous action is taken by the UPD350 in a fault condition.

Remarks

By default, it is configured to PIO3. User can also use stack's enum eUPD_PIO_NUM_TYPE to define this.

Example

```
#define CONFIG_PORT_0_UPD_EN_VBUS        3   (EN_VBUS pin is PIO3)
#define CONFIG_PORT_0_UPD_EN_VBUS        0xFF (EN_VBUS functinality disabled)
```

8.8.2.8 CONFIG_PORT_n_UPD_EN_VBUS_PIO_MODE

C

```
#define CONFIG_PORT_n_UPD_EN_VBUS_PIO_MODE ePUSH_PULL_ACTIVE_HIGH
```

Description

CONFIG_PORT_n_UPD_EN_VBUS_PIO_MODE defines the PIO mode of the UPD350 PIO EN_VBUS defined in CONFIG_PORT_n_UPD_EN_VBUS. It takes only values from enum eUPD_OUTPUT_PIN_MODES_TYPE. n can take values between 0 and CONFIG_PD_PORT_COUNT - 1.

Remarks

By default, it is configured to ePUSH_PULL_ACTIVE_HIGH.

Example

```
#define CONFIG_PORT_0_UPD_EN_VBUS_PIO_MODE        ePUSH_PULL_ACTIVE_HIGH
(EN_VBUS is configured as Active signal in output mode)
```

8.8.2.9 CONFIG_PORT_n_UPD_VBUS_DIS_PIO_MODE

C

```
#define CONFIG_PORT_n_UPD_VBUS_DIS_PIO_MODE ePUSH_PULL_ACTIVE_HIGH
```

Description

CONFIG_PORT_n_UPD_VBUS_DIS_PIO_MODE defines the PIO mode of the UPD350 PIO VBUS_DIS defined in CONFIG_PORT_n_UPD_VBUS_DIS_PIO_NO. It takes only values from enum eUPD_OUTPUT_PIN_MODES_TYPE. n can take values between 0 and (CONFIG_PD_PORT_COUNT-1).

Remarks

By default, it is configured to ePUSH_PULL_ACTIVE_HIGH.

Example

```
#define CONFIG_PORT_0_UPD_VBUS_DIS_PIO_MODE        ePUSH_PULL_ACTIVE_HIGH
(VBUS_DIS is configured as Active signal in output mode)
```

8.8.2.10 CONFIG_PORT_n_UPD_VBUS_DIS_PIO_NO

C

```
#define CONFIG_PORT_n_UPD_VBUS_DIS_PIO_NO eUPD_PIO4
```

Description

CONFIG_PORT_n_UPD_VBUS_DIS_PIO_NO defines the UPD350 PIO for VBUS discharge functionality. It is a control for discharging VBUS (connecting VBUS to GND). It asserts as per CONFIG_PORT_n_UPD_VBUS_DIS_PIO_MODE

whenever VBUS voltage must transition from a high voltage to a lower voltage state and when VBUS is disabled. n can take values between 0 and (CONFIG_PD_PORT_COUNT-1). It takes value from 0 to 15 and to disable the pin functionality from stack, user can define it as 0xFF. It is applicable only when CONFIG_DCDC_CTRL is defined as PWRCTRL_DEFAULT_PSF_GPIO_CONFIG.

Remarks

By default, it is configured to PIO4. User can also use stack's enum eUPD_PIO_NUM_TYPE to define this.

Example

```
#define CONFIG_PORT_0_UPD_VBUS_DIS_PIO_NO        eUPD_PIO4 (VBUS_DIS is PIO4)
#define CONFIG_PORT_0_UPD_VBUS_DIS_PIO_NO        0xFF (EN_VBUS functinality disabled)
```

8.8.2.11 CONFIG_PORT_n_UPD_VSELx_PIO_MODE

C

```
#define CONFIG_PORT_n_UPD_VSELx_PIO_MODE ePUSH_PULL_ACTIVE_HIGH
```

Description

CONFIG_PORT_n_UPD_VSELx_PIO_MODE defines the PIO mode of the UPD350 PIO VSELx defined in CONFIG_PORT_n_UPD_VSELx_PIO_NO. It takes only values from enum eUPD_OUTPUT_PIN_MODES_TYPE. n can take values between 0 and (CONFIG_PD_PORT_COUNT-1). x takes vlaue between 0 to 2.

Remarks

By default, it is configured to ePUSH_PULL_ACTIVE_HIGH.

Example

```
#define CONFIG_PORT_0_UPD_VSEL0_PIO_MODE        ePUSH_PULL_ACTIVE_HIGH
(VSEL0 is configured as Active signal in output mode)
```

8.8.2.12 CONFIG_PORT_n_UPD_VSELx_PIO_NO

C

```
#define CONFIG_PORT_n_UPD_VSELx_PIO_NO eUPD_PIO7
```

Description

CONFIG_PORT_n_UPD_VSELx_PIO_NO defines the UPD350 PIO as voltage selector pins(VSEL[2:0]). PSF stack provides provision for three Voltage selector pin VSEL[2:0]. It is used to control the output voltage of the DC/DC controller. In a typical application, these pins are used to switch in different resistors into the feedback loop to vary the output voltage. n can take values between 0 and CONFIG_PD_PORT_COUNT - 1. x takes vlaue between 0 to 2. This define takes value from 0 to 15 and to disable the pin functionality from stack, user can define it as 0xFF. It is applicable only when CONFIG_DCDC_CTRL is defined as PWRCTRL_DEFAULT_PSF_GPIO_CONFIG.

Remarks

By default, VSEL0, VSEL1, VSEL2 is configured to PIO7, PIO8 and PIO9 respectively. User can also use stack's enum eUPD_PIO_NUM_TYPE to define this. It is applicable only for source operation.

Example

```
#define CONFIG_PORT_0_UPD_VSEL0_PIO_NO        7 (VSEL0 for port 0 is PIO7)
#define CONFIG_PORT_0_UPD_VSEL1_PIO_NO        8 (VSEL1 for port 0 is PIO8)
#define CONFIG_PORT_0_UPD_VSEL2_PIO_NO        9 (VSEL2 for port 0 is PIO9)
#define CONFIG_PORT_0_UPD_VSEL0_PIO_NO        0xFF (VSEL0 for port 0 is disabled)
```

8.8.2.13 CONFIG_PORT_n_VSAFE0V_VSEL_MAPPING

C

```
#define CONFIG_PORT_n_VSAFE0V_VSEL_MAPPING 0x00
```

Description

CONFIG_PORT_n_VSAFE0V_VSEL_MAPPING defines the assertion and de-assertion to be driven on VSEL[2:0] pins(defined in CONFIG_PORT_n_UPD_VSELx_PIO_NO) by the PSF as per CONFIG_PORT_n_UPD_VSELx_PIO_MODE to have a output voltage of VSafe0V out of DC-Dc controller. n can take values between 0 and (CONFIG_PD_PORT_COUNT-1).

Remarks

By default, it is configured to 0x00. It is applicable only for source operation.

Example

```
#define CONFIG_PORT_0_VSAFE0V_VSEL_MAPPING        0x00
```

8.8.2.14 CONFIG_PORT_n_PDO_x_VSEL_MAPPING

C

```
#define CONFIG_PORT_n_PDO_x_VSEL_MAPPING 0x00
```

Description

CONFIG_PORT_n_VSAFE0V_VSEL_MAPPING defines the assertion and de-assertion to be driven on VSEL[2:0] pins(defined in CONFIG_PORT_n_UPD_VSELx_PIO_NO) by the PSF stack as per CONFIG_PORT_n_UPD_VSELx_PIO_MODE to have a output voltage of PDO voltage defined in CONFIG_PORT_n_SOURCE_PDO_x_VOLTAGE out of DC-DC controller. n can take values between 0 and (CONFIG_PD_PORT_COUNT-1). x takes value between 1 to 7 as by PD specification, there can only be 7 PDOs. It is applicable only for source.

Remarks

By default, a 1 pin per voltage implementation is implemented. VSEL[2:0]: 000 - 5V (No pins asserted) 001 - 9V (VSEL0 asserted) 010 - 15V (VSEL1 asserted) 100 - 20V (VSEL2 asserted)

Example

```
#define CONFIG_PORT_0_PDO_1_VSEL_MAPPING    0x00
#define CONFIG_PORT_0_PDO_2_VSEL_MAPPING    0x01
#define CONFIG_PORT_0_PDO_3_VSEL_MAPPING    0x02
#define CONFIG_PORT_0_PDO_4_VSEL_MAPPING    0x04
#define CONFIG_PORT_0_PDO_5_VSEL_MAPPING    0x00
#define CONFIG_PORT_0_PDO_6_VSEL_MAPPING    0x00
#define CONFIG_PORT_0_PDO_7_VSEL_MAPPING    0x00
```

8.9 PDFU Configuration

Macros

	Name	Description
⊷	CONFIG_PDFU_SUPPORTED	Power Delivery Firmware Update Supported field.
⊷	CONFIG_PDFU_VIA_USBPD_SUPPORTED	Power Delivery Firmware Update Supported via USB config.
⊷	CONFIG_MAX_FIRMWARE_IMAGESIZE	Maximum Firmware image size.
⊷	CONFIG_RECONFIG_PHASE_WAITTIME	Reconfig phase wait time value.
⊷	CONFIG_TRANSFER_PHASE_WAITTIME	Transfer phase wait time value.
⊷	CONFIG_UPDATABLE_IMAGEBANK_INDEX	Index of Updatable image.
⊷	CONFIG_VALIDATION_PHASE_WAITTIME	Validation phase wait time value.

Description

This section explains the configurable options available for PDFU.

8.9.1 CONFIG_PDFU_SUPPORTED

C

```
#define CONFIG_PDFU_SUPPORTED 1
```

Description

CONFIG_PDFU_SUPPORTED is set to '0' if firmware is not updatable during Run time. Otherwise shall be set to 1. It is used by the PD Firmware Update state-machine during Enumeration phase. This information is shared with the PDFU Initiator as part of GET_FW_ID command's response.

Remarks

The user definition of this macro is mandatory when INCLUDE_PDFU is '1'. By default, it is defined as '1'.

Example

```
#define CONFIG_PDFU_SUPPORTED     1
```

8.9.2 CONFIG_PDFU_VIA_USBPD_SUPPORTED

C

```
#define CONFIG_PDFU_VIA_USBPD_SUPPORTED 1
```

Description

CONFIG_PDFU_VIA_USBPD_SUPPORTED Set to '1' to indicate support for PDFU via USB PD Firmware Update flow.Otherwise shall be set to '0'. It is used by the PD Firmware Update state-machine during Enumeration phase. This information is shared with the PDFU Initiator as part of GET_FW_ID command's response.

Remarks

The user definition of this macro is mandatory when INCLUDE_PDFU is '1'. The default value is '1'.

Example

```
#define CONFIG_PDFU_VIA_USBPD_SUPPORTED  1
```

8.9.3 CONFIG_MAX_FIRMWARE_IMAGESIZE

C

```
#define CONFIG_MAX_FIRMWARE_IMAGESIZE 0x8800UL
```

Description

CONFIG_MAX_FIRMWARE_IMAGESIZE defines the ROM size allocated for the Updatable application. PDFU Initiator shall flash entire size during every re-flash operation. Flashing lesser or more than this Size results in error response.

Remarks

Choose Firmware Image size in such a way that integral multiple of 256. The definition of this function is mandatory when INCLUDE_PDFU is '1' and shall expressed in terms of bytes. By default, the value is 0x8800UL(32KB).

Example

```
#define CONFIG_MAX_FIRMWARE_IMAGESIZE 38*1024 (38*1024 bytes for 38KB Updatable
application).
```

8.9.4 CONFIG_RECONFIG_PHASE_WAITTIME

C

```
#define CONFIG_RECONFIG_PHASE_WAITTIME 0x00u
```

Description

CONFIG_RECONFIG_PHASE_WAITTIME specifies the Wait time required for the Reconfigure state, i.e. the PDFU_Initiate request processing takes "Wait time" millisecond, and next request can be issued by the PDFU_Initiator after the specified wait time. This information is shared with the PDFU Initiator as part of PDFU_INITIATE command's response.

Remarks

The user definition of this macro is mandatory when INCLUDE_PDFU is TRUE. It can have values from 0x00 to 0xFF. By default, it is defined as '0x00'.

Example

```
#define CONFIG_RECONFIG_PHASE_WAITTIME   0x03u //3ms wait time required
```

8.9.5 CONFIG_TRANSFER_PHASE_WAITTIME

C

```
#define CONFIG_TRANSFER_PHASE_WAITTIME 0x64u
```

Description

CONFIG_TRANSFER_PHASE_WAITTIME Species the Wait time required during the Transfer state, i.e. the PDFU Data request processing takes "Wait time" millisecond, and next PDFU_DATA request to be issued by the initiator after the specified wait time. This information is shared with the PDFU Initiator as part of PDFU_DATA command's response.

Remarks

The user definition of this macro is mandatory when INCLUDE_PDFU is TRUE. It can have values from 0x00 to 0xFF. By default, it is defined as '0x03'.

Example

```
#define CONFIG_TRANSFER_PHASE_WAITTIME   0x03u //3ms required for processing PDFU_DATA
request
```

8.9.6 CONFIG_UPDATABLE_IMAGEBANK_INDEX

C

```
#define CONFIG_UPDATABLE_IMAGEBANK_INDEX 0x03u
```

Description

CONFIG_UPDATABLE_IMAGEBANK_INDEX specifies the Image bank index for which firmware upgrade is requested (or) in other words it corresponds to the image bank index of the Updatable application as mentioned by Architecture 2 of PD FW Update Specification.

This information is used during the Reconfiguration phase to determine what application is currently executing and whether application switching to Fixed Application is required or not.

Remarks

The user definition of this macro is mandatory when INCLUDE_PDFU is '1'. By default, this macro is defined as 0x03.

Example

```
#define CONFIG_UPDATABLE_IMAGEBANK_INDEX   0x03u (3 image banks are available)
```

8.9.7 CONFIG_VALIDATION_PHASE_WAITTIME

C

```
#define CONFIG_VALIDATION_PHASE_WAITTIME 0x03u
```

Description

CONFIG_VALIDATION_PHASE_WAITTIME specifies the wait time macro for the validation state,i.e. the PDFU_Validate command's processing takes "Wait time" millisecond, and next request can be issued by the Initiator after the specified wait time.

Remarks

The user definition of this macro is mandatory when INCLUDE_PDFU is TRUE. It can have values from 0x00 to 0xFF. By default, it is defined as '0x03'.

Example

```
#define CONFIG_VALIDATION_PHASE_WAITTIME   0x03u
```

8.10 Type-C and USB-PD Specification defined Timeout Configuration

Macros

Name	Description
CONFIG_TYPEC_ERRORRECOVERY_TIMEOUT_MS	tErrorRecovery.
CONFIG_TYPEC_TCCDEBOUNCE_TIMEOUT_MS	tCCDebounce.
CONFIG_TYPEC_TPDEBOUNCE_TIMEOUT_MS	tPDDebounce.
CONFIG_TYPEC_VBUS_OFF_TIMER_MS	tVBUSOFF.
CONFIG_TYPEC_VBUS_ON_TIMER_MS	tVBUSON.
CONFIG_TYPEC_VCONNDISCHARGE_TIMEOUT_MS	tVCONNDischarge.
CONFIG_TYPEC_VCONNOFF_TIMEOUT_MS	tVCONNOFF.
CONFIG_TYPEC_VCONNON_TIMEOUT_MS	tVCONNON.
CONFIG_PRL_BIST_CONTMODE_TIMEOUT_MS	tBISTContMode.
CONFIG_PRL_CHUNKSENDERREQUEST_TIMEOUT_MS	tChunkSenderRequest.
CONFIG_PRL_CHUNKSENDERRESPONSE_TIMEOUT_MS	tChunkSenderResponse.
CONFIG_PRL_SINKTX_TIMEOUT_MS	tSinkTx.
CONFIG_PE_NORESPONSE_TIMEOUT_MS	tNoResponse.
CONFIG_PE_PSHARDRESET_TIMEOUT_MS	tPSHardReset.
CONFIG_PE_PSTRANSITION_TIMEOUT_MS	tPSTransition.
CONFIG_PE_SENDERRESPONSE_TIMEOUT_MS	tSenderResponse.
CONFIG_PE_SINKREQUEST_TIMEOUT_MS	tSinkRequest.
CONFIG_PE_SINKWAITCAP_TIMEOUT_MS	tTypeCSinkWaitCap.
CONFIG_PE_SOURCECAPABILITY_TIMEOUT_MS	tTypeCSendSourceCap.
CONFIG_PE_SRC_READY_TIMEOUT_MS	tSrcReady.
CONFIG_PE_SRCRECOVER_TIMEOUT_MS	tSrcRecover.

8

⊸	CONFIG_PE_SRCTRANSISTION_TIMEOUT_MS	tSrcTransistionTimer.
⊸	CONFIG_PE_VCONNOFF_TIMEOUT_MS	tVCONNOFF.
⊸	CONFIG_PE_VCONNON_SELF_TIMEOUT_MS	Self tVCONNSourceOn.
⊸	CONFIG_PE_VCONNON_TIMEOUT_MS	tVCONNSourceOn.
⊸	CONFIG_PE_VDMRESPONSE_TIMEOUT_MS	tVDMSenderResponse.

Description

PSF provides defines for the user to configure various Timeouts specified by Type-C and USB Power Delivery Specification.

8.10.1 CONFIG_TYPEC_ERRORRECOVERY_TIMEOUT_MS

C

```
#define CONFIG_TYPEC_ERRORRECOVERY_TIMEOUT_MS MILLISECONDS_TO_TICKS(500)
```

Description

CONFIG_TYPEC_ERRORRECOVERY_TIMEOUT_MS defines the tErrorRecovery timeout specified in the USB Type C Specification. Default value of CONFIG_TYPEC_ERRORRECOVERY_TIMEOUT_MS is set as 500 milliseconds.

Remarks

CONFIG_TYPEC_ERRORRECOVERY_TIMEOUT_MS can be configured depending on the microcontroller platform platform used, for the device to be USB Type C Compliant. It shall always be expressed in define MILLISECONDS_TO_TICKS.

Example

```
#define CONFIG_TYPEC_ERRORRECOVERY_TIMEOUT_MS     MILLISECONDS_TO_TICKS(500)
```

8.10.2 CONFIG_TYPEC_TCCDEBOUNCE_TIMEOUT_MS

C

```
#define CONFIG_TYPEC_TCCDEBOUNCE_TIMEOUT_MS MILLISECONDS_TO_TICKS(150)
```

Description

CONFIG_TYPEC_TCCDEBOUNCE_TIMEOUT_MS defines the tCCDebounce timeout specified in the USB Type C Specification. Default value of CONFIG_TYPEC_TCCDEBOUNCE_TIMEOUT_MS is set as 150 milliseconds.

Remarks

CONFIG_TYPEC_TCCDEBOUNCE_TIMEOUT_MS can be configured depending on the microcontroller platform platform used, for the device to be USB Type C Compliant. It shall always be expressed in define MILLISECONDS_TO_TICKS.

Example

```
#define CONFIG_TYPEC_TCCDEBOUNCE_TIMEOUT_MS     MILLISECONDS_TO_TICKS(150)
```

8.10.3 CONFIG_TYPEC_TPDEBOUNCE_TIMEOUT_MS

C

```
#define CONFIG_TYPEC_TPDEBOUNCE_TIMEOUT_MS MILLISECONDS_TO_TICKS(10)
```

Description

CONFIG_TYPEC_TPDEBOUNCE_TIMEOUT_MS defines the tPDDebounce timeout specified in the USB Type C Specification. Default value of this macro is set as 10 milliseconds.

Remarks

CONFIG_TYPEC_TPDEBOUNCE_TIMEOUT_MS can can be configured depending on the microcontroller platform platform used, for the device to be USB Type C Compliant. It shall always be expressed in define MILLISECONDS_TO_TICKS.

Example

```
#define CONFIG_TYPEC_TPDEBOUNCE_TIMEOUT_MS       MILLISECONDS_TO_TICKS(10)
```

8.10.4 CONFIG_TYPEC_VBUS_OFF_TIMER_MS

C

```
#define CONFIG_TYPEC_VBUS_OFF_TIMER_MS MILLISECONDS_TO_TICKS(650)
```

Description

CONFIG_TYPEC_VBUS_OFF_TIMER_MS defines the tVBUSOFF specified in the USB-TypeC Specification. Default value of CONFIG_TYPEC_VBUS_OFF_TIMER_MS is set as 650 milliseconds.

Remarks

CONFIG_TYPEC_VBUS_OFF_TIMER_MS can be configured depending on the microcontroller platform used, for the device to be USB Type C Compliant. It shall always be expressed in define MILLISECONDS_TO_TICKS.

Example

```
#define CONFIG_TYPEC_VBUS_OFF_TIMER_MS             MILLISECONDS_TO_TICKS(650)
```

8.10.5 CONFIG_TYPEC_VBUS_ON_TIMER_MS

C

```
#define CONFIG_TYPEC_VBUS_ON_TIMER_MS MILLISECONDS_TO_TICKS(275)
```

Description

CONFIG_TYPEC_VBUS_ON_TIMER_MS defines the tVBUSON specified in the USB-TypeC Specification. Default value of CONFIG_TYPEC_VBUS_ON_TIMER_MS is set as 275 milliseconds.

Remarks

CONFIG_TYPEC_VBUS_ON_TIMER_MS can be configured depending on the microcontroller platform used, for the device to be USB Type C Compliant. It shall always be expressed in define MILLISECONDS_TO_TICKS.

Example

```
#define CONFIG_TYPEC_VBUS_ON_TIMER_MS                  MILLISECONDS_TO_TICKS(275)
```

8.10.6
CONFIG_TYPEC_VCONNDISCHARGE_TIMEOUT_MS

C

```
#define CONFIG_TYPEC_VCONNDISCHARGE_TIMEOUT_MS MILLISECONDS_TO_TICKS(35)
```

Description

CONFIG_TYPEC_VCONNDISCHARGE_TIMEOUT_MS defines the tVCONNDischarge timeout specified in the USB Type C Specification. Default value of CONFIG_TYPEC_VCONNDISCHARGE_TIMEOUT_MS is set as 35 milliseconds.

Remarks

CONFIG_TYPEC_VCONNDISCHARGE_TIMEOUT_MS can be configured depending on the microcontroller platform platform used, for the device to be USB Type C Compliant. It shall always be expressed in define MILLISECONDS_TO_TICKS.

Example

```
#define CONFIG_TYPEC_VCONNDISCHARGE_TIMEOUT_MS     MILLISECONDS_TO_TICKS(35)
```

8.10.7 CONFIG_TYPEC_VCONNOFF_TIMEOUT_MS

C

```
#define CONFIG_TYPEC_VCONNOFF_TIMEOUT_MS MILLISECONDS_TO_TICKS(25)
```

Description

CONFIG_TYPEC_VCONNOFF_TIMEOUT_MS defines the tVCONNOFF specified in the USB-Type C Specification. Default value of CONFIG_TYPEC_VCONNOFF_TIMEOUT_MS is set as 25 milliseconds.

Remarks

CONFIG_TYPEC_VCONNOFF_TIMEOUT_MS can be configured depending on the microcontroller platform used, for the device to be USB PD Compliant. It shall always be expressed in define MILLISECONDS_TO_TICKS.

Example

```
#define CONFIG_TYPEC_VCONNOFF_TIMEOUT_MS          MILLISECONDS_TO_TICKS(25)
```

8.10.8 CONFIG_TYPEC_VCONNON_TIMEOUT_MS

C

```
#define CONFIG_TYPEC_VCONNON_TIMEOUT_MS MILLISECONDS_TO_TICKS(10)
```

Description

CONFIG_TYPEC_VCONNON_TIMEOUT_MS defines the tVCONNON specified in the USB-Type C Specification. Default value of CONFIG_TYPEC_VCONNON_TIMEOUT_MS is set as 2 milliseconds.

Remarks

CONFIG_TYPEC_VCONNON_TIMEOUT_MS can be configured depending on the microcontroller platform used, for the device to be USB Type-C Compliant. It shall always be expressed in define MILLISECONDS_TO_TICKS.

Example

```
#define CONFIG_TYPEC_VCONNON_TIMEOUT_MS           MILLISECONDS_TO_TICKS(10)
```

8.10.9 CONFIG_PRL_BIST_CONTMODE_TIMEOUT_MS

C

```
#define CONFIG_PRL_BIST_CONTMODE_TIMEOUT_MS MILLISECONDS_TO_TICKS(45)
```

Description

CONFIG_PRL_BIST_CONTMODE_TIMEOUT_MS defines the BISTContModeTimer specified in the USB-PD Specification. Default value of CONFIG_PRL_BIST_CONTMODE_TIMEOUT_MS is set as 45 milliseconds.

Remarks

CONFIG_PRL_BIST_CONTMODE_TIMEOUT_MS can be configured depending on the microcontroller platform used, for

the device to be USB PD Compliant. It shall always be expressed in define MILLISECONDS_TO_TICKS.

Example

```
#define CONFIG_PRL_BIST_CONTMODE_TIMEOUT_MS          MILLISECONDS_TO_TICKS(45)
```

8.10.10
CONFIG_PRL_CHUNKSENDERREQUEST_TIMEOUT_MS

C

```
#define CONFIG_PRL_CHUNKSENDERREQUEST_TIMEOUT_MS MILLISECONDS_TO_TICKS(26)
```

Description

CONFIG_PRL_CHUNKSENDERREQUEST_TIMEOUT_MS defines the ChunkSenderRequestTimer specified in the USB-PD Specification. Default value of CONFIG_PRL_CHUNKSENDERREQUEST_TIMEOUT_MS is set as 26 milliseconds.

Remarks

CONFIG_PRL_CHUNKSENDERREQUEST_TIMEOUT_MS can be configured depending on the microcontroller platform used, for the device to be USB PD Compliant. It shall always be expressed in define MILLISECONDS_TO_TICKS.

Example

```
#define CONFIG_PRL_CHUNKSENDERREQUEST_TIMEOUT_MS        MILLISECONDS_TO_TICKS(26)
```

8.10.11
CONFIG_PRL_CHUNKSENDERRESPONSE_TIMEOUT_MS

C

```
#define CONFIG_PRL_CHUNKSENDERRESPONSE_TIMEOUT_MS MILLISECONDS_TO_TICKS(26)
```

Description

CONFIG_PRL_CHUNKSENDERRESPONSE_TIMEOUT_MS defines the ChunkSenderResponseTimer specified in the USB-PD Specification. Default value of CONFIG_PRL_CHUNKSENDERRESPONSE_TIMEOUT_MS is set as 26 milliseconds.

Remarks

CONFIG_PRL_CHUNKSENDERRESPONSE_TIMEOUT_MS can be configured depending on the microcontroller platform used, for the device to be USB PD Compliant. It shall always be expressed in define MILLISECONDS_TO_TICKS.

Example

```
#define CONFIG_PRL_CHUNKSENDERRESPONSE_TIMEOUT_MS     MILLISECONDS_TO_TICKS(26)
```

8.10.12 CONFIG_PRL_SINKTX_TIMEOUT_MS

C

```
#define CONFIG_PRL_SINKTX_TIMEOUT_MS MILLISECONDS_TO_TICKS(16)
```

Description

CONFIG_PRL_SINKTX_TIMEOUT_MS defines the SinkTxTimer specified in the USB-PD Specification. Default value of CONFIG_PRL_SINKTX_TIMEOUT_MS is set as 16 milliseconds.

Remarks

CONFIG_PRL_SINKTX_TIMEOUT_MS can be configured depending on the microcontroller platform used, for the device to be USB PD Compliant. It shall always be expressed in define MILLISECONDS_TO_TICKS.

Example

```
#define CONFIG_PRL_SINKTX_TIMEOUT_MS                        MILLISECONDS_TO_TICKS(16)
```

8.10.13 CONFIG_PE_NORESPONSE_TIMEOUT_MS

C

```
#define CONFIG_PE_NORESPONSE_TIMEOUT_MS MILLISECONDS_TO_TICKS(5500)
```

Description

CONFIG_PE_NORESPONSE_TIMEOUT_MS defines the NoResponseTimer specified in the USB-PD Specification. Default value of CONFIG_PE_NORESPONSE_TIMEOUT_MS is set as 5.5 seconds.

Remarks

CONFIG_PE_NORESPONSE_TIMEOUT_MS an be configured depending on the microcontroller platform used, for the device to be USB PD Compliant. It shall always be expressed in define MILLISECONDS_TO_TICKS.

Example

```
#define CONFIG_PE_NORESPONSE_TIMEOUT_MS         MILLISECONDS_TO_TICKS(5500)
```

8.10.14 CONFIG_PE_PSHARDRESET_TIMEOUT_MS

C

```
#define CONFIG_PE_PSHARDRESET_TIMEOUT_MS MILLISECONDS_TO_TICKS(28)
```

Description

CONFIG_PE_PSHARDRESET_TIMEOUT_MS defines the PSHardResetTimer specified in the USB-PD Specification. Default value of CONFIG_PE_PSHARDRESET_TIMEOUT_MS is set as 28 milliseconds.

Remarks

CONFIG_PE_PSHARDRESET_TIMEOUT_MS can be configured depending on the microcontroller platform used, for the device to be USB PD Compliant. It shall always be expressed in define MILLISECONDS_TO_TICKS.

Example

```
#define CONFIG_PE_PSHARDRESET_TIMEOUT_MS                 MILLISECONDS_TO_TICKS(28)
```

8.10.15 CONFIG_PE_PSTRANSITION_TIMEOUT_MS

C

```
#define CONFIG_PE_PSTRANSITION_TIMEOUT_MS MILLISECONDS_TO_TICKS(500)
```

Description

CONFIG_PE_PSTRANSITION_TIMEOUT_MS defines the PSTransitionTimer specified in the USB-PD Specification. Default value of CONFIG_PE_PSTRANSITION_TIMEOUT_MS is set as 500 milliseconds.

Remarks

CONFIG_PE_PSTRANSITION_TIMEOUT_MS can be configured depending on the microcontroller platform used, for the device to be USB PD Compliant. It shall always be expressed in define MILLISECONDS_TO_TICKS.

Example

```
#define CONFIG_PE_PSTRANSITION_TIMEOUT_MS          MILLISECONDS_TO_TICKS(500)
```

8.10.16 CONFIG_PE_SENDERRESPONSE_TIMEOUT_MS

C

```
#define CONFIG_PE_SENDERRESPONSE_TIMEOUT_MS MILLISECONDS_TO_TICKS(26)
```

Description

CONFIG_PE_SENDERRESPONSE_TIMEOUT_MS defines the SenderResponseTimer specified in the USB-PD Specification. Default value of CONFIG_PE_SENDERRESPONSE_TIMEOUT_MS is set as 26 milliseconds.

Remarks

CONFIG_PE_SENDERRESPONSE_TIMEOUT_MS can be configured depending on the microcontroller platform used, for the device to be USB PD Compliant. It shall always be expressed in define MILLISECONDS_TO_TICKS.

Example

```
#define CONFIG_PE_SENDERRESPONSE_TIMEOUT_MS        MILLISECONDS_TO_TICKS(26)
```

8.10.17 CONFIG_PE_SINKREQUEST_TIMEOUT_MS

C

```
#define CONFIG_PE_SINKREQUEST_TIMEOUT_MS MILLISECONDS_TO_TICKS(100)
```

Description

CONFIG_PE_SINKREQUEST_TIMEOUT_MS defines the SinkRequestTimer specified in the USB-PD Specification. Default value of CONFIG_PE_SINKREQUEST_TIMEOUT_MS is set as 100 milliseconds.

Remarks

CONFIG_PE_SINKREQUEST_TIMEOUT_MS can be configured depending on the microcontroller platform used, for the device to be USB PD Compliant. It shall always be expressed in define MILLISECONDS_TO_TICKS.

Example

```
#define CONFIG_PE_SINKREQUEST_TIMEOUT_MS           MILLISECONDS_TO_TICKS(100)
```

8.10.18 CONFIG_PE_SINKWAITCAP_TIMEOUT_MS

C

```
#define CONFIG_PE_SINKWAITCAP_TIMEOUT_MS MILLISECONDS_TO_TICKS(465)
```

Description

CONFIG_PE_SINKWAITCAP_TIMEOUT_MS defines the SinkWaitCapTimer specified in the USB-PD Specification. Default value of CONFIG_PE_SINKWAITCAP_TIMEOUT_MS is set as 465 milliseconds.

Remarks

CONFIG_PE_SINKWAITCAP_TIMEOUT_MS can be configured depending on the microcontroller platform used, for the device to be USB PD Compliant. It shall always be expressed in define MILLISECONDS_TO_TICKS.

Example

```
#define CONFIG_PE_SINKWAITCAP_TIMEOUT_MS           MILLISECONDS_TO_TICKS(465)
```

8.10.19 CONFIG_PE_SOURCECAPABILITY_TIMEOUT_MS

C

```
#define CONFIG_PE_SOURCECAPABILITY_TIMEOUT_MS MILLISECONDS_TO_TICKS(150)
```

Description

CONFIG_PE_SOURCECAPABILITY_TIMEOUT_MS defines the SourceCapabilityTimer specified in the USB-PD Specification. Default value of CONFIG_PE_SOURCECAPABILITY_TIMEOUT_MS is set as 150 milliseconds.

Remarks

CONFIG_PE_SOURCECAPABILITY_TIMEOUT_MS can be configured depending on the microcontroller platform used, for the device to be USB PD Compliant. It shall always be expressed in define MILLISECONDS_TO_TICKS.

Example

```
#define CONFIG_PE_SOURCECAPABILITY_TIMEOUT_MS      MILLISECONDS_TO_TICKS(150)
```

8.10.20 CONFIG_PE_SRC_READY_TIMEOUT_MS

C

```
#define CONFIG_PE_SRC_READY_TIMEOUT_MS MILLISECONDS_TO_TICKS(285)
```

Description

CONFIG_PE_SRC_READY_TIMEOUT_MS defines the tSrcReady specified in the PD 3.0 Specification. Default value of CONFIG_PE_SRC_READY_TIMEOUT_MS is set as 285 milliseconds.

Remarks

CONFIG_PE_SRC_READY_TIMEOUT_MS can be configured depending on the microcontroller platform used, for the device to be USB PD Compliant. It shall always be expressed in define MILLISECONDS_TO_TICKS.

Example

```
#define CONFIG_PE_SRC_READY_TIMEOUT_MS               MILLISECONDS_TO_TICKS(285)
```

8.10.21 CONFIG_PE_SRCRECOVER_TIMEOUT_MS

C

```
#define CONFIG_PE_SRCRECOVER_TIMEOUT_MS MILLISECONDS_TO_TICKS(800)
```

Description

CONFIG_PE_SRCRECOVER_TIMEOUT_MS defines the tSrcRecover specified in the USB-PD Specification. Default value of CONFIG_PE_SRCRECOVER_TIMEOUT_MS is set as 800 milliseconds.

Remarks

CONFIG_PE_SRCRECOVER_TIMEOUT_MS can be configured depending on the microcontroller platform used, for the device to be USB PD Compliant. It shall always be expressed in define MILLISECONDS_TO_TICKS.

Example

```
#define CONFIG_PE_SRCRECOVER_TIMEOUT_MS                MILLISECONDS_TO_TICKS(800)
```

8

8.10.22 CONFIG_PE_SRCTRANSISTION_TIMEOUT_MS

C

```
#define CONFIG_PE_SRCTRANSISTION_TIMEOUT_MS MILLISECONDS_TO_TICKS(28)
```

Description

CONFIG_PE_SRCTRANSISTION_TIMEOUT_MS defines the tSrcTransistionTimer specified in the USB-PD Specification. By default it is set to 30 milliseconds.

Remarks

CONFIG_POWER_GOOD_TIMER_MS can be configured depending on the microcontroller platform used, for the device to be USB PD Compliant. It shall always be expressed in define MILLISECONDS_TO_TICKS.

Example

```
#define CONFIG_PE_SRCTRANSISTION_TIMEOUT_MS          MILLISECONDS_TO_TICKS(30)
```

8.10.23 CONFIG_PE_VCONNOFF_TIMEOUT_MS

C

```
#define CONFIG_PE_VCONNOFF_TIMEOUT_MS MILLISECONDS_TO_TICKS(35)
```

Description

CONFIG_PE_VCONNOFF_TIMEOUT_MS defines the tVCONNOFF specified in the USB-Type C Specification. Default value of CONFIG_PE_VCONNOFF_TIMEOUT_MS is set as 35 milliseconds.

Remarks

CONFIG_PE_VCONNOFF_TIMEOUT_MS can be configured depending on the microcontroller platform used, for the device to be USB PD Compliant. It shall always be expressed in define MILLISECONDS_TO_TICKS.

Example

```
#define CONFIG_PE_VCONNOFF_TIMEOUT_MS          MILLISECONDS_TO_TICKS(35)
```

8.10.24 CONFIG_PE_VCONNON_SELF_TIMEOUT_MS

C

```
#define CONFIG_PE_VCONNON_SELF_TIMEOUT_MS MILLISECONDS_TO_TICKS(150)
```

Description

CONFIG_PE_VCONNON_SELF_TIMEOUT_MS defines the tVCONNSourceOn specified in the USB PD Specification. Default value of CONFIG_PE_VCONNON_SELF_TIMEOUT_MS is set as 100 milliseconds.

Remarks

CONFIG_PE_VCONNON_SELF_TIMEOUT_MS can be configured depending on the microcontroller platform used, for the device to be USB PD Compliant. It shall always be expressed in define MILLISECONDS_TO_TICKS.

Example

```
#define CONFIG_PE_VCONNON_SELF_TIMEOUT_MS          MILLISECONDS_TO_TICKS(150)
```

8.10.25 **CONFIG_PE_VCONNON_TIMEOUT_MS**

C

```
#define CONFIG_PE_VCONNON_TIMEOUT_MS MILLISECONDS_TO_TICKS(100)
```

Description

CONFIG_PE_VCONNON_TIMEOUT_MS defines the tVCONNSourceOn specified in the USB PD Specification. Default value of CONFIG_PE_VCONNON_TIMEOUT_MS is set as 100 milliseconds.

Remarks

CONFIG_PE_VCONNON_TIMEOUT_MS can be configured depending on the microcontroller platform used, for the device to be USB PD Compliant. It shall always be expressed in define MILLISECONDS_TO_TICKS.

Example

```
#define CONFIG_PE_VCONNON_TIMEOUT_MS              MILLISECONDS_TO_TICKS(100)
```

8.10.26 **CONFIG_PE_VDMRESPONSE_TIMEOUT_MS**

C

```
#define CONFIG_PE_VDMRESPONSE_TIMEOUT_MS MILLISECONDS_TO_TICKS(28)
```

Description

CONFIG_PE_VDMRESPONSE_TIMEOUT_MS defines the VDMResponseTimer specified in the USB-PD Specification. Default value of CONFIG_PE_VDMRESPONSE_TIMEOUT_MS is set as 28 milliseconds.

Remarks

CONFIG_PE_VDMRESPONSE_TIMEOUT_MS can be configured depending on the microcontroller platform used, for the device to be USB PD Compliant. It shall always be expressed in define MILLISECONDS_TO_TICKS.

Example

```
#define CONFIG_PE_VDMRESPONSE_TIMEOUT_MS              MILLISECONDS_TO_TICKS(28)
```

8

9 System level integration of PSF

As PSF is hardware agnostic, it can be integrated to any SoC platform. This topic provides details on

- Hardware and Software requirements for system level integration of PSF
- Steps to integrate
- APIs implementation required for integration
- Notification callback from PSF

All this APIs required for PSF porting and integration are available in PSF_APIHooks.h header file. It is recommended to include this PSF_APIHooks.h file path as one of preprocessor include path. The template of PSF_APIHooks.h for new SoC integration is available at ..\PSF\SOC_Portable\New_SOC.

9.1 Hardware Requirements

This section elaborates the hardware requirements needed from SoC for PSF integration.

9.1.1 UPD350

The UPD350 is a companion Power Delivery Port Controller that provides cable plug orientation and detection for a USB Type-C receptacle. It implements baseband communication with a partner Type-C device on the opposite side of the cable.

The UPD350 communicates to an external SoC using the integrated I2C/SPI interface. One UPD350 per port is required to achieve USB-PD functionality in each port.

PSF being tightly coupled with UPD350 hardware supports the following versions of UPD350 Rev A Silicon.

Device	Hardware communication Interface
UPD350-A	I2C
UPD350-B	I2C
UPD350-C	SPI
UPD350-D	SPI

PSF requires following hardware support from SoC to control UPD350:

- Hardware communication interface either I2C or SPI to interact with UPD350
- PIOs per UPD350 to detect the UPD350 IRQ alert
- PIO to reset the UPD350s

9.1.1.1 Hardware Communication Interface

SoC shall use either SPI or I2C interface of UPD350 to access all the on-chip Control and Status Registers.

SPI Master:

External SoC shall have SPI master support for the PSF to interact with UPD350 through SPI interface. UPD350 SPI slave supports read and write instructions at a maximum SPI Clock Frequency of 25MHz. Each PD port having a UPD350 shall have a separate SPI Chip Select(CS) line for the SoC's SPI master to uniquely identify the SPI slave.
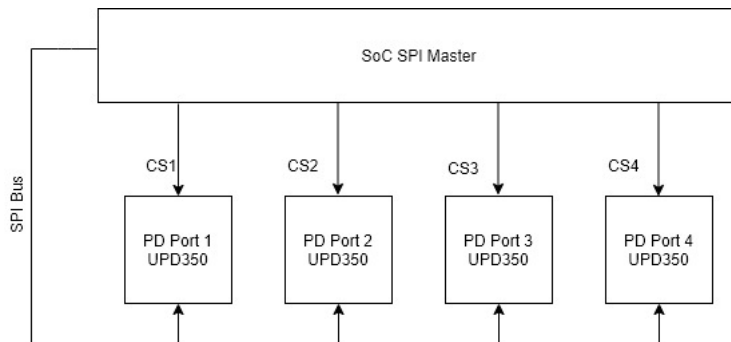
Only 2 Port Source and Sink solution has been tested at 8MHz SPI clock speed.

I2C Master:

External SoC shall have I2C master support for the PSF to interact with UPD350 through I2C interface. UPD350 I2C slave supports Standard mode(100 kbps), Fast Mode(400 kbps) and Fast Mode Plus(1 Mbps) speeds. Each PD port having a UPD350 is identified by unique I2C slave address.
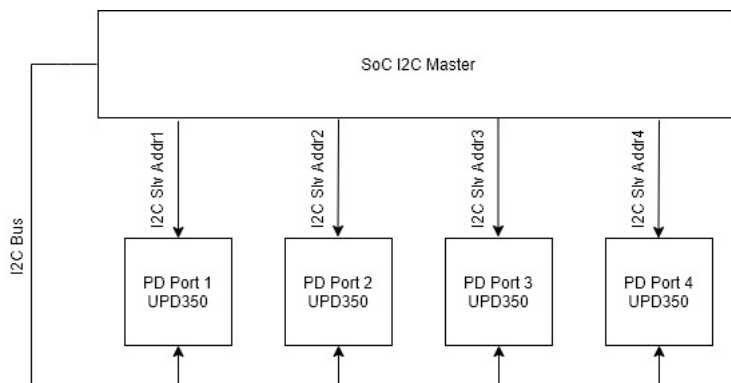
System level PD communication between PSF and UPD350 through I2C interface is untested, whereas basic I2C read/writes are tested.

9.1.1.2 PIOs for UPD350 IRQs

A PIO specific to each port's UPD350 is required for UPD350 IRQ interrupt detection. IRQ_N of UPD350 is active low signal. Thus, SoC 's User_Application shall the configure the PIO to active low interrupt detection. And it shall inform PSF about the occurrence of interrupts through APIs provided; for PSF to service the interrupt.
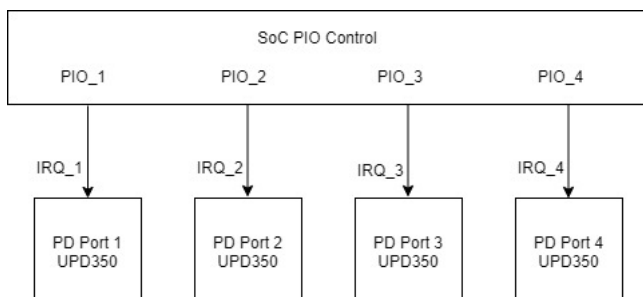
9.1.1.3 **PIO for UPD350 Reset**

PSF needs dedicated PIO from SoC to reset the UPD350s. It is recommended to use single PIO for all the UPD350s in the system. It is connected to RESET_N of UPD350, an active low signal.
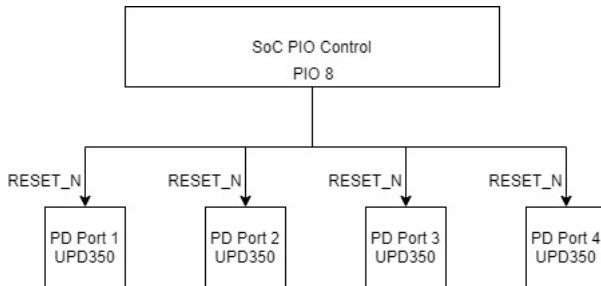
9.1.2 **Hardware Timer**

A Hardware timer with minimum resolution of 1ms is required for PSF functionality. PSF creates multiple software instance of this hardware timer. SoC User_Application shall inform the PSF the occurrence of every Timer interrupt through APIs provided.

PSF is tested with hardware timer resolution of 1ms for two port source configuration.

9.1.3 **DC-DC Buck Boost Controller**

UPD350 cannot drive higher PD voltages directly. A DC-DC buck boost controller is required to drive higher PD voltages.

For Source-only port, the setup would be as follows,
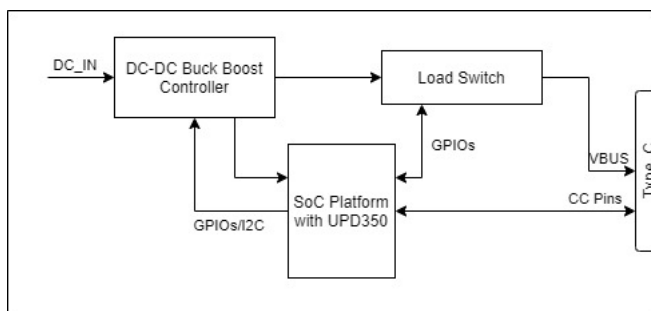
For Sink-Only port, the set up would be as follows,
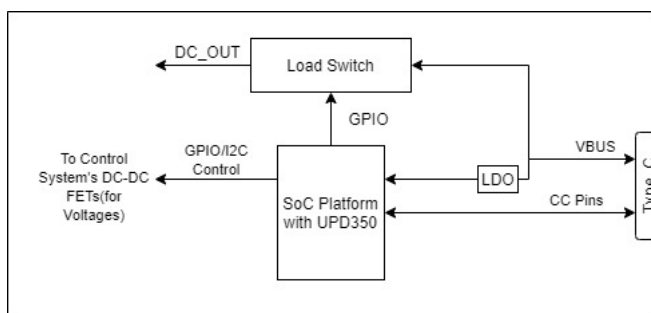
By default, PSF supports GPIO based Microchip Power Buck controller MCP19119.

9.2 Software Requirements

This sections lists the software requirements for PSF integration. It helps to decide whether PSF integration is possible with the available memory constraints in SoC.

9.2.1 Memory Requirement

- **32-bit**

In a 32-bit SoC environment,

Estimated code size of PSF for 2-Port Source only solution is 31.5KB

Estimated Data RAM size of PSF for 2-Port Source only solution is 1.2KB. Any addition of port further would cause 500 Bytes for each port.

- **16-bit**

Currently PSF is not tested for 16-bit based SoC environment.

- **8-bit**

Currently PSF is not tested for 8-bit based SoC environment.

9.2.2 Multi-Port Support

PSF supports maximum of 4 ports.

Note: The tested PSF configuration is 2 port source configuration with SPI interface.

9.2.3 Endianness

PSF supports only Little-Endian format.

9.3 Steps to integrate PSF

Steps to integrate to new SoC platform:

1. Select a compatible SoC based on Hardware Requirements and Software Requirements

2. Generate required SoC drivers. Port the mandatory APIs specified in APIs to be implemented by the User application for PSF functionality in PSF_APIHooks header file.

3. Integrate PSF APIs listed in APIs to be callled by the User Application to SoC User application for PSF complete porting and integration.

4. Choose a PD functionality for the application (for example, Source only operation, Sink only operation) and configure the PSF_Config.h file with the help of section <u>PSF Configuration Options.</u>

9

10 APIs Implementation required for SW integration

List of APIs required for PSF software integration is explained in this section.

It covers,

- APIs to be implemented by the SoC User_Application
- APIs to be called by the SoC User_Application

10.1 Data types

Pre-processor definitions	Data types
TRUE	1
FALSE	0
BOOL	unsigned char
UINT8	unsigned char
UINT16	unsigned short
UINT32	unsigned long
INT8	char
INT16	short
INT32	long
CHAR	char
UCHAR	unsigned char
SIZEOF(x)	sizeof(x)

10.2 APIs to be implemented by the User Application

This section lists out APIs that to be implemented by User for PSF functionality. Implement all the APIs mentioned as mandatory in the remarks.

10

10.2.1 UPD350 Hardware Interface Configurations

Macros

	Name	Description
⊸	MCHP_PSF_HOOK_UPDHW_INTF_INIT	Initialize the hardware interface(SPI/I2C) used for communicating with UPD350 part.
⊸	MCHP_PSF_HOOK_UPD_COMM_ENDIS	Enable or disables hardware interface(SPI/I2C) communication to the port's UPD350
⊸	MCHP_PSF_HOOK_UPD_READ	Initiates a read transfer to UPD350 via I2C/SPI
⊸	MCHP_PSF_HOOK_UPD_WRITE	Initiates a write transfer to UPD350 via I2C/SPI

10.2.1.1 MCHP_PSF_HOOK_UPDHW_INTF_INIT

C

```
#define MCHP_PSF_HOOK_UPDHW_INTF_INIT 0
```

Description

PSF needs a Hardware interface from SOC either SPI or I2C to communicate with UPD350. UPD350 supports either I2C or SPI interface depending on UPD350 part used. UPD350 A and C supports I2C interface and UPD350 B and D part supports SPI interface. This Hook is to initialize the SOC's Hardware interface for communication. It is called during initialization of PSF. Define relevant function that has no arguments but a return type UINT8 that indicates whether initialization is successful.

Preconditions

Use SPI interface for part UPD350 B and D. Use I2C interface for part UPD350 A and C.

Returns

UINT8 - Return TRUE if initialization was successful or else return FALSE. If hardware interface initialization(SPI/I2C) fails and the API returns FALSE, all the PD ports are disabled by PSF by default.

Remarks

User definition of this Hook function is mandatory.

Example

```
#define MCHP_PSF_HOOK_UPDHW_INTF_INIT()      hw_spi_init()
UINT8 hw_spi_init(void);
UINT8 hw_spi_init(void)
{
    //Intialise SOC's SPI master
    //Return TRUE if initialsation is successful else FALSE
}
#define MCHP_PSF_HOOK_UPDHW_INTF_INIT()     hw_i2c_init()
UINT8 hw_i2c_init(void);
UINT8 hw_i2c_init(void)
{
    //Initialise SOC's I2C master
    //Return TRUE if initialsation is successful else FALSE
}
```

10.2.1.2 MCHP_PSF_HOOK_UPD_COMM_ENDIS

C

```
#define MCHP_PSF_HOOK_UPD_COMM_ENDIS(u8PortNum,u8EnDis)
```

Description

This hook is called before and after the read/write operations of <u>UPD350</u> registers with port specifically enable/disable hardware interface(I2C/SPI) communication to the port's <u>UPD350</u>. For SPI interface, SPI chip select level shall be driven low/high with respect to the port number passed in the hook to enable/disable SPI communication respectively to the port's <u>UPD350</u>. For I2C interface, I2C mux shall be configured to route/disable the SOC's I2C master to the port number passed in the hook to enable/disable I2C communication respectively to the port's <u>UPD350</u>. Define relevant function that has two UINT8 argument with out return type.

Preconditions

None.

Parameters

Parameters	Description
u8PortNum	Port number of the device. It takes value between 0 to (<u>CONFIG_PD_PORT_COUNT</u>-1).
u8EnDis	Parameter indicating whether to enable or disable the communication for the port.

Returns

None.

Remarks

User definition of this Hook function is mandatory for SPI Hardware interface

Example

```c
#define MCHP_PSF_HOOK_UPD_COMM_ENDIS (u8PortNum,u8EnDis)
                hw_spi_drive_cs (u8PortNum, u8EnDis)
void hw_spi_drive_cs (UINT8 u8PortNum,UINT8 u8EnDis);
void hw_spi_drive_cs (UINT8 u8PortNum,UINT8 u8EnDis)
{
    if (u8EnDis == TRUE)
    {
        //Set pin level low for SOC's GPIO that is connected to the u8PortNum port's
        //UPD350 SPI CS pin
    }
    else if(u8EnDis == FALSE)
    {
        //Set pin level high for SOC GPIO that is connected to the u8PortNum port's
        //UPD350 SPI CS pin
    }
}
#define MCHP_PSF_HOOK_UPD_COMM_ENDIS(u8PortNum,u8EnDis)
                    hw_port_i2cmux_routing(u8PortNum,u8EnDis)
void hw_port_i2cmux_routing(u8PortNum,u8EnDis);
void hw_port_i2cmux_routing(u8PortNum,u8EnDis)
{
    if (u8EnDis == TRUE)
    {
        //Route the I2C mux to the u8PortNum UPD350 Port
    }
    else if(u8EnDis == FALSE)
    {
        //disable the I2C mux routing to the u8PortNum UPD350 Port
    }
}
```

10.2.1.3 MCHP_PSF_HOOK_UPD_READ

C

```c
#define MCHP_PSF_HOOK_UPD_READ(u8PortNum,pu8WriteBuf,u8WriteLen,pu8ReadBuf, u8ReadLen) 0
```

Description

This hook is called to read to UPD350 registers with respect to the port. Its definition is confined CONFIG_DEFINE_UPD350_HW_INTF_SEL definition for SPI/I2C selection. Define relevant function that has UINT8, UINT8*, UINT8, UINT8*, UINT8 arguments with UINT8 return type.

Preconditions

None.

Parameters

Parameters	Description
u8PortNum	Port number of the device. It takes value between 0 to (CONFIG_PD_PORT_COUNT-1).
pu8WriteBuf	PSF shall pass the pointer to the buffer which has the data to be written on the SPI/I2C Hardware bus before read. It contains the Register address to be read. Data type of the pointer buffer must be UINT8*.
u8WriteLen	PSF shall pass the Number of bytes to be written on the SPI/I2C Hardware bus. Data type of this parameter must be UINT8.
pu8ReadBuf	PSF will pass the pointer to the buffer where data read from the SPI/I2C bus to be stored. Data type of the pointer buffer must be UINT8*.
u8ReadLen	PSF will pass the number of bytes to be read on the SPI/I2C bus. Data type of this parameter must be UINT8.

Returns

UINT8 - Return TRUE if read was successful or else return FALSE.

Remarks

User definition of this Hook function is mandatory.

Example

```
#define MCHP_PSF_HOOK_UPD_READ(u8PortNum,pu8WriteBuf,u8WriteLen,pu8ReadBuf,u8ReadLen)
        SPI_Read (u8PortNum,pu8WriteBuf,u8WriteLen,pu8ReadBuf,u8ReadLen)
void SPI_Read (UINT8 u8PortNum, UINT8 *pu8WriteBuffer, UINT8 u8Writelength,
                       UINT8 *pu8ReadBuffer, UINT8 u8Readlength);
void SPI_Read (UINT8 u8PortNum, UINT8 *pu8WriteBuffer, UINT8 u8Writelength,
                       UINT8 *pu8ReadBuffer, UINT16 u8Readlength)
{
    for(UINT8 u8Txcount = 0; u8Txcount < u16Writelength; u8Txcount++)
    {
        //Write data bytes to SPI bus
    }
    for(UINT8 u8Rxcount = 0; u8Rxcount< u8Readlength; u8Rxcount++)
    {
        //Read data from SPI bus
    }
    // Return TRUE if the read is successful; else FALSE
}
#define MCHP_PSF_HOOK_UPD_READ(u8PortNum,pu8WriteBuf,u8WriteLen,pu8ReadBuf,u8ReadLen)
             I2C_Read(u8PortNum,pu8WriteBuf,u8WriteLen,pu8ReadBuf,u8ReadLen)

void I2C_Read (UINT8 u8PortNum, UINT8 *pu8WriteBuffer, UINT8 u8Writelength,
                       UINT8 *pu8ReadBuffer, UINT16 u8Readlength);
void I2C_Read (UINT8 u8PortNum, UINT8 *pu8WriteBuffer, UINT8 u8Writelength,
                       UINT8 *pu8ReadBuffer, UINT16 u8Readlength)
{
    //Select I2C address for the UPD350 I2C slave using u8PortNum
    for(UINT8 u8Txcount = 0; u8Txcount < u16Writelength; u8Txcount++)
    {
        //Write data bytes to I2C bus
    }
    for(UINT8 u8Rxcount = 0; u8Rxcount< u8Readlength; u8Rxcount++)
    {
        //Read data from I2C bus
    }
```

10

```
                    // Return TRUE if the read is successful else  return FALSE
            }
```

10.2.1.4 MCHP_PSF_HOOK_UPD_WRITE

C

```
#define MCHP_PSF_HOOK_UPD_WRITE(u8PortNum,pu8WriteBuf,u8WriteLen) 0
```

Description

This hook is called to write to UPD350 registers specific to the port. Its definition is confined to CONFIG_DEFINE_UPD350_HW_INTF_SEL definition for SPI or I2C selection. Define relevant function that has UINT8, UINT8*, UINT8 arguments with a return type UINT8.

Preconditions

None.

Parameters

Parameters	Description
u8PortNum	Port number of the device. It takes value between 0 to (CONFIG_PD_PORT_COUNT-1).
pu8WriteBuf	PSF shall pass the pointer to the buffer which has the data to be written on the SPI/I2C Hardware bus. Data type of the pointer buffer must be UINT8*.
u8WriteLen	PSF shall pass the Number of bytes to be written on the SPI/I2C Hardware bus. Data type of this parameter must be UINT8.

Returns

UINT8 - Return TRUE if write was successful or else return FALSE.

Remarks

User definition of this Hook function is mandatory

Example

```
    #define MCHP_PSF_HOOK_UPD_WRITE(u8PortNum,pu8WriteBuf,u8WriteLen)
            SPI_Write (u8PortNum,pu8WriteBuf,u8WriteLen)
UINT8 SPI_Write(UINT8 u8PortNum, UINT8 *pu8WriteBuffer, UINT8 u8Writelength);
UINT8 SPI_Write(UINT8 u8PortNum, UINT8 *pu8WriteBuffer, UINT8 u8Writelength)
{
    for(UINT8 u8Txcount = 0; u8Txcount < u16Writelength; u8Txcount++)
    {
        //Write data bytes to SPI bus
    }
    // Return TRUE if the write is successful; else FALSE
}
    #define MCHP_PSF_HOOK_UPD_WRITE(u8PortNum,pu8WriteBuf,u8WriteLen)
            I2C_Write (u8PortNum,pu8WriteBuf,u8WriteLen)
UINT8 I2C_Write(UINT8 u8PortNum, UINT8 *pu8WriteBuffer, UINT8 u8Writelength);
UINT8 I2C_Write(UINT8 u8PortNum, UINT8 *pu8WriteBuffer, UINT8 u8Writelength)
{
    // Select I2C address for the UPD350 I2C slave using u8PortNum
    for(UINT8 u8Txcount = 0; u8Txcount < u16Writelength; u8Txcount++)
    {
        //Write data bytes to I2C bus
    }
    // Return TRUE if the write is successful; else FALSE
}
```

10

10.2.2 **PD Timer Configuration**

Macros

	Name	Description
⊸	MCHP_PSF_PDTIMER_INTERRUPT_RATE	PD Timer Interrupt Rate
⊸	MCHP_PSF_HOOK_HW_PDTIMER_INIT	Hook to Initialize and start the hardware timer module.

10.2.2.1 **MCHP_PSF_PDTIMER_INTERRUPT_RATE**

C

```
#define MCHP_PSF_PDTIMER_INTERRUPT_RATE 1000
```

Description

MCHP_PSF_PDTIMER_INTERRUPT_RATE defines the frequency of interrupt set in the hardware timer dedicated for PSF. In other words, it is the resolution of the hardware timer. It can be configured depending on the resolution of the hardware timer available.

Remarks

Resolution of the hardware timer has to be at least 1ms. Tested resolution values of hardware timer is 1ms.(Corresponding MCHP_PSF_PDTIMER_INTERRUPT_RATE value is 1000).

Example

```
#define MCHP_PSF_PDTIMER_INTERRUPT_RATE     1000
(1000 interrupts per second, with interrupt interval or resolution of 1ms)
```

10.2.2.2 **MCHP_PSF_HOOK_HW_PDTIMER_INIT**

C

```
#define MCHP_PSF_HOOK_HW_PDTIMER_INIT 0
```

Description

PSF requires a single dedicated hardware timer module for its functionality. This Hook initializes and starts the hardware timer module for MCHP_PSF_PDTIMER_INTERRUPT_RATE interrupt frequency. To inform PSF about the occurrence of hardware timer interrupt API MchpPSF_PDTimerHandler should be called by the SOC layer on every timer interrupt. Define relevant function that has no argument with UINT8 return type.

Preconditions

None.

Returns

UINT8 - Returns TRUE if initialization is successful else FALSE. If Timer initialization fails and the API returns FALSE, all the PD ports are disabled by PSF by default.

Remarks

User definition of this Hook function is mandatory

Example

```
#define MCHP_PSF_HOOK_HW_PDTIMER_INIT()     Timer_Init()
UINT8 Timer_Init(void);
UINT8 Timer_Init(void)
{
    //Initialize and start the SOC timer module for MCHP_PSF_PDTIMER_INTERRUPT_RATE
    //interrupt frequency & register MchpPSF_PDTimerHandler as callback
```

10

```
        //Return TRUE if Timer initialisation is successful else return FALSE
    }
```

10.2.3 UPD350 IRQ Control

Macros

	Name	Description
➥	MCHP_PSF_HOOK_UPD_IRQ_GPIO_INIT	Initializes the SOC GPIOs connected to the IRQ_N lines of ports' UPD350.

10.2.3.1 MCHP_PSF_HOOK_UPD_IRQ_GPIO_INIT

C

```
#define MCHP_PSF_HOOK_UPD_IRQ_GPIO_INIT(u8PortNum)
```

Description

PSF requires GPIO specific to each port for UPD350 interrupt detection via UPD350's IRQ_N lines. IRQ_N is an active low signal. This Hook shall initialize the SOC GPIOs connected to the IRQ_N lines of UPD350s in the system for interrupt notification. It is recommended to configure SOC GPIOs interrupt in edge level detection with internal pull up since the UPD350 keeps the IRQ_N line in low state until the interrupt is cleared. To inform PSF the occurrence of UPD350 interrupt, API MchpPSF_UPDIrqHandler shall be called by SOC on interrupt detection port specifically. Define relevant function that has port number as argument without return type.

Preconditions

SOC GPIO connected to IRQ_N should be wakeup capable if INCLUDE_POWER_MANAGEMENT_CTRL defined as 1.

Parameters

Parameters	Description
u8PortNum	Port number of the device. It takes value between 0 to (CONFIG_PD_PORT_COUNT-1).

Returns

None.

Remarks

User definition of this Hook function is mandatory

Example

```
    #define MCHP_PSF_HOOK_UPD_IRQ_GPIO_INIT(u8PortNum)      INT_ExtIrqInit(u8PortNum)
    void INT_ExtIrqInit(UINT8 u8PortNum);
    void INT_ExtIrqInit(UINT8 u8PortNum)
    {
        if(0 == u8PortNum)
        {
            //Initializes the SOC GPIO that is connected to Port 0's UPD350
            //Configure GPIO for internal pull up and low level edge detection
            //Register MchpPSF_UPDIrqHandler(0) as callback for interrupt occurence
        }
        if(1 == u8PortNum)
        {
            //Initializes the SOC GPIO that is connected to Port 1's UPD350
            //Configure GPIO for internal pull and low level edge detection
            //Register MchpPSF_UPDIrqHandler(1) as callback for interrupt occurrence
        }
    }
```

10

10.2.4 UPD350 Reset Control

Macros

	Name	Description
⊸	MCHP_PSF_HOOK_UPD_RESET_GPIO_INIT	Initializes the SOC GPIOs connected to the RESET_N lines of UPD350s
⊸	MCHP_PSF_HOOK_UPD_RESET_THRU_GPIO	Resets the UPD350 connected specific to the port.

10.2.4.1 MCHP_PSF_HOOK_UPD_RESET_GPIO_INIT

C

```
#define MCHP_PSF_HOOK_UPD_RESET_GPIO_INIT(u8PortNum)
```

Description

This hook initializes the SOC GPIOs connected to the RESET_N lines of Port's UPD350. It is recommended to connect a single GPIO to the reset line of all UPD350s. User can also have separate GPIO for each port. As the UPD350 RESET_N is active low signal, SOC should initialize the GPIO to be high by default. Define relevant function that has port number as argument without return type.

Preconditions

None.

Parameters

Parameters	Description
u8PortNum	Port number of the device. It takes value between 0 to (CONFIG_PD_PORT_COUNT-1).

Returns

None.

Remarks

User definition of this Hook function is mandatory

Example

```
#define MCHP_PSF_HOOK_UPD_RESET_GPIO_INIT(u8PortNum)        updreset_init(u8PortNum)
void updreset_init(UINT8 u8PortNum);
void updreset_init(UINT8 u8PortNum)
{
    // If single SOC GPIO is connected to all the UPD350's, do initialisation only once
    // when PortNum is '0'. If separate GPIOs are used for each port UPD350, do
    //initialisation port specifically
    if (0 == u8PortNum)
    {
        //Initialization of SOC GPIO connected to UPD350 reset lines
        //Make the gpio line high by default
    }
}
```

10.2.4.2 MCHP_PSF_HOOK_UPD_RESET_THRU_GPIO

C

```
#define MCHP_PSF_HOOK_UPD_RESET_THRU_GPIO(u8PortNum)
```

10

Description

This hook is to reset the UPD350 connected to the port by driving the SOC GPIO connected to the RESET_N pin of that UPD350. Since, RESET_N is active low signal, SOC GPIO should be driven low for a while and then back to default high state. It is recommended to have common reset pin for all ports. In such case user must drive the GPIO for UPD350 reset only when u8PortNum passed is '0' via this hook.Define relevant function that has port number as argument without return type.

Preconditions

None.

Parameters

Parameters	Description
u8PortNum	Port number of the device. It takes value between 0 to (CONFIG_PD_PORT_COUNT-1).

Returns

None.

Remarks

User definition of this Hook function is mandatory

Example

```
#define MCHP_PSF_HOOK_UPD_RESET_THRU_GPIO(u8PortNum)        updreset_thru_gpio(u8PortNum)
void updreset_thru_gpio(UINT8 u8PortNum);
void updreset_thru_gpio(UINT8 u8PortNum)
{
    if (0 == u8PortNum)
    {
        //Enable pull down
        // Wait for xxx uS
        // Enable pull up
    }
}
```

10.2.5 SoC Interrupt Enable/Disable

Macros

	Name	Description
⊸	MCHP_PSF_HOOK_ENABLE_GLOBAL_INTERRUPT	Enables the global interrupt.
⊸	MCHP_PSF_HOOK_DISABLE_GLOBAL_INTERRUPT	Disables the global interrupt.

10.2.5.1 MCHP_PSF_HOOK_ENABLE_GLOBAL_INTERRUPT

C

```
#define MCHP_PSF_HOOK_ENABLE_GLOBAL_INTERRUPT
```

Description

This hook is called by the PSF when exits from critical section. It must provide an implementation to enable the interrupts globally. This function must be very short, otherwise response time to the interrupt will take longer time. Define relevant function that has no arguments without return type.

Preconditions

None.

Returns

None.

Remarks

User definition of this Hook function is mandatory

Example

```
#define MCHP_PSF_HOOK_ENABLE_GLOBAL_INTERRUPT()   CRITICAL_SECTION_EXIT()
void CRITICAL_SECTION_EXIT(void);
void CRITICAL_SECTION_EXIT()
{
   //Enable global interrupts
}
```

10.2.5.2 MCHP_PSF_HOOK_DISABLE_GLOBAL_INTERRUPT

C

```
#define MCHP_PSF_HOOK_DISABLE_GLOBAL_INTERRUPT
```

Description

This hook is called when PSF enter into a critical section. It must provide an implementation to disable the interrupts globally. This hook implementation must be very short, otherwise response time to the interrupt will take longer time. Define relevant function that has no arguments without return type.

Preconditions

None.

Returns

None.

Remarks

User definition of this Hook function is mandatory

Example

```
#define MCHP_PSF_HOOK_DISABLE_GLOBAL_INTERRUPT()   CRITICAL_SECTION_ENTER()
void CRITICAL_SECTION_ENTER(void);
void CRITICAL_SECTION_ENTER()
{
    //Disable SOC's global interrupts
}
```

10.2.6 Memory Compare and Copy

Macros

	Name	Description
⊸	MCHP_PSF_HOOK_MEMCMP	Compare two memory regions
⊸	MCHP_PSF_HOOK_MEMCPY	Copies one memory area to another memory area

10.2.6.1 MCHP_PSF_HOOK_MEMCMP

C

```
#define MCHP_PSF_HOOK_MEMCMP(pObj1, pObj2, iLength) 0
```

10

Description

This function is called to compare two memory regions pau8Obj1, pau8Obj2 with specified length u8Length. User must define this hook based on compiler of SOC.

Preconditions

None.

Parameters

Parameters	Description
pObj1	This is the pointer to block of Memory region 1
pObj2	This is the pointer to block of Memory region 2
iLength	This is the number of bytes to be compared.

Returns

Return 0 if two memory regions are same else return number of bytes did not match.

Remarks

User definition of this Hook function is mandatory

Example

```
#define MCHP_PSF_HOOK_MEMCMP(pObj1, pObj2, iLength) memcmp(pObj1, pObj2, iLength)
//This hook definition can be compiler defined or user defined.
```

10.2.6.2 MCHP_PSF_HOOK_MEMCPY

C

```
#define MCHP_PSF_HOOK_MEMCPY(pDest, pSrc, iLen) 0
```

Description

This function is called to copy iLen bytes from pSrc memory area to pDest memory area. User must define this function based on compiler of SOC. The memory areas must not overlap.

Preconditions

None.

Parameters

Parameters	Description
pDest	This is the pointer to block of destination memory region
pSrc	This is the pointer to block of source memory region
iLen	This is the number of bytes to be copied.

Returns

Returns a pointer to pDest.

Remarks

User definition of this Hook function is mandatory

Example

```
#define MCHP_PSF_HOOK_MEMCPY(pDest, pSrc, iLen) memcpy(pDest, pSrc, iLen)
//This hook definition can be compiler defined or user defined.
```

10

10.2.7 Structure Packing

Macros

	Name	Description
⊸	MCHP_PSF_STRUCT_PACKED_START	Structure packing to align the bytes in data memory based on the compiler.
⊸	MCHP_PSF_STRUCT_PACKED_END	Structure packing to align the bytes in data memory based on the compiler.

10.2.7.1 MCHP_PSF_STRUCT_PACKED_START

C

```
#define MCHP_PSF_STRUCT_PACKED_START
```

Description

Generally packed structures will be used to save space & align the bytes in data memory based on the compiler. If this pre-processor is defined, then all the PSF's "C" structures will be replaced with this keyword for compilation. If this pre-processor is not defined, then it will be default compilation rules based on the compiler.

Remarks

Need to be packed always based on type of SOC.

Example

```
#define MCHP_PSF_STRUCT_PACKED_START   __attribute__((__packed__))
```

10.2.7.2 MCHP_PSF_STRUCT_PACKED_END

C

```
#define MCHP_PSF_STRUCT_PACKED_END
```

Description

Generally packed structures will be used to save space & align the bytes in data memory based on the compiler. If this pre-processor is defined, then all the PSF's "C" structures will be replaced with this keyword for compilation. If this pre-processor is not defined, then it will be default compilation rules based on the compiler.

Remarks

Need to be packed always based on type of SOC.

Example

```
#define CONFIG_STRUCT_PACKED_END   _Pragma("pack()")
```

10.2.8 Port Power Control

Macros

	Name	Description
⊸	MCHP_PSF_HOOK_HW_PORTPWR_INIT	Initializes all the hardware modules related to port power functionality especially DC-DC buck boost controller and load switch.
⊸	MCHP_PSF_HOOK_PORTPWR_DRIVE_VBUS	Drives the VBUS for given voltage, current

10

	MCHP_PSF_HOOK_PORTPWR_ENDIS_VBUSDISCH	Enables or disables VBUS Discharge functionality for a given port.
	MCHP_PSF_HOOK_PORTPWR_CONFIG_SINK_HW	Enables or disables sink hardware circuitry and configures it to sinks the VBUS voltage for a given port based on the sink requested voltage and current.

10.2.8.1 MCHP_PSF_HOOK_HW_PORTPWR_INIT

C

```
#define MCHP_PSF_HOOK_HW_PORTPWR_INIT(u8PortNum)
```

Description

PSF provides default DC-DC Buck booster control configuration via CONFIG_DCDC_CTRL define. User can chose to implement their own DC-DC buck booster control or modify the default using this hook. This hook is to initialize the hardware modules related to port power functionality. Implementation of this function depends on the type of DC-DC buck boost controller and load switch used. Define relevant function that has no argument without return type.

Preconditions

API implementation must make sure the Port Power(VBUS) of all ports must be set to 0V.

Parameters

Parameters	Description
u8PortNum	Port number of the device. It takes value between 0 to (CONFIG_PD_PORT_COUNT-1).

Returns

None.

Remarks

User definition of this Hook function is mandatory if CONFIG_DCDC_CTRL is undefined

Example

```
#define MCHP_PSF_HOOK_HW_PORTPWR_INIT(u8PortNum)        hw_portpower_init(u8PortNum)
void hw_portpower_init(void);
void hw_portpower_init(void)
{
    //Initializes the hardware modules related to port power functionality
}
```

10.2.8.2 MCHP_PSF_HOOK_PORTPWR_DRIVE_VBUS

C

```
#define MCHP_PSF_HOOK_PORTPWR_DRIVE_VBUS(u8PortNum,u16VBUSVolatge,u16Current)
```

Description

PSF provides default DC-DC Buck booster control configuration via CONFIG_DCDC_CTRL define. If user chose to implement their own DC-DC buck booster control, this hook must be implemented to drive VBUS as per the parameter passed based on voltage and current. It can also be used to modify the default option. Implementation of this function depends on the type of DC-DC buck boost controller and load switch used. Define relevant function that has UINT8,UINT16, UINT16 arguments without return type.

Preconditions

It is applicable only for Source operation.

10

Parameters

Parameters	Description
u8PortNum	Port number of the device. It takes value between 0 to (CONFIG_PD_PORT_COUNT-1).
u16VBUSVolatge	VBUS Voltage level to be driven in VBUS expressed in terms of milliVolts.
u16Current	VBUS current level in terms of mA.

Returns

None.

Remarks

User definition of this Hook function is mandatory if CONFIG_DCDC_CTRL is undefined.

Example

```
#define MCHP_PSF_HOOK_PORTPWR_DRIVE_VBUS(u8PortNum, u16VBUSVolatge, u16Current)
        hw_portpower_driveVBUS(u8PortNum, u16VBUSVolatge, u16Current)
void hw_portpower_driveVBUS(UINT8 u8PortNum, UINT16 u16VBUSVolatge, UINT16 u16Current);
void hw_portpower_driveVBUS(UINT8 u8PortNum, UINT16 u16VBUSVolatge, UINT16 u16Current)
{
    // Configure DC-DC buck boost control to drive u16VBUSVolatge & u16Current in VBUS
}
```

10.2.8.3 MCHP_PSF_HOOK_PORTPWR_ENDIS_VBUSDISCH

C

```
#define MCHP_PSF_HOOK_PORTPWR_ENDIS_VBUSDISCH(u8PortNum, u8EnableDisable)
```

Description

VBUS Discharge mechanism is required to enable quick of discharge VBUS when VBUS transition from higher to lower voltage. PSF provides default DC-DC Buck booster control configuration via CONFIG_DCDC_CTRL define. If user chose to implement their own DC-DC buck booster control, this hook must be implemented to enable or disable the VBUS Discharge functionality for a given Port. Implementation of this function depends on the type of DC-DC buck boost controller and load switch used. Define relevant function that has UINT8,UINT8 arguments without return type.

Preconditions

MCHP_PSF_HOOK_PORTPWR_ENDIS_VBUSDISCH is called in ISR handler. Its implementation shall be very short, otherwise response time to the interrupt will take longer time. Passing of the Compliance test "TD.4.2.1" (Source Connect Sink) in "USB_Type_C_Functional_Test_Specification" depends on the VBUS Discharge circuitry used. Typical VBUS Discharge time from any higher voltage to 0V should be around 10ms.

Parameters

Parameters	Description
u8PortNum	Port number of the device. It takes value between 0 to (CONFIG_PD_PORT_COUNT-1).
u8EnableDisable	Flag indicating whether to enable/disable VBUS Discharge mechanism.

Returns

None.

Remarks

User definition of this Hook function is mandatory if CONFIG_DCDC_CTRL is undefined.

Example

```
#define MCHP_PSF_HOOK_PORTPWR_ENDIS_VBUSDISCH(u8PortNum, u8EnableDisable)
            hw_portpower_enab_dis_VBUSDischarge(u8PortNum, u8EnableDisable)
void hw_portpower_enab_dis_VBUSDischarge(UINT8 u8PortNum,UINT8 u8EnableDisable);
void hw_portpower_enab_dis_VBUSDischarge(UINT8 u8PortNum,UINT8 u8EnableDisable)
```

```
{
    if (TRUE == u8EnableDisable)
    {
        //Enable the VBUS Discharge for "u8PortNum" Port
    }
    else
    {
        //Disable the VBUS Discharge for "u8PortNum" Port
    }
}
```

10.2.8.4 MCHP_PSF_HOOK_PORTPWR_CONFIG_SINK_HW

C

```
#define MCHP_PSF_HOOK_PORTPWR_CONFIG_SINK_HW(u8PortNum,u16Voltage,u16Current)
```

Description

This hook is to enable or disable sink hardware circuitry and configure it for Sink requested requested current and voltage.Implementation of this function depends on the type of Sink circuitry used. Define relevant function that has UINT8,UINT16,UINT16 arguments without return type.

Preconditions

It is applicable only for Sink operation.

Parameters

Parameters	Description
u8PortNum	Port number of the device. It takes value between 0 to (CONFIG_PD_PORT_COUNT-1).
u16voltage	Enable Sink HW Circuitry if the u16voltage is not Vsafe0V to drain power. Disable sink HW circuitry if the u16voltage is VSafe0V. Configure the HW to requested u16voltage in mV.
u16Current	Configure the HW for the requested current passed in terms of mA.

Returns

None.

Remarks

User definition of this Hook function is mandatory if PSF is configured for Sink functionality.

Example

```
#define MCHP_PSF_HOOK_PORTPWR_CONFIG_SINK_HW(u8PortNum, u16Voltage, u16Current)
    hw_SinkCircuitary_enab_dis_(u8PortNum, u16Voltage, u16Current)
void hw_SinkCircuitary_enab_dis_(UINT8 u8PortNum,UINT16 u16Votlage,UINT16 u16Current);
void hw_SinkCircuitary_enab_dis_(UINT8 u8PortNum,UINT16 u16Votlage,UINT16 u16Current)
{
    if(u16Voltage == Vsafe0V)
    {
        //Disable the Sink circuitary for "u8PortNum" Port
    }
    else
    {
        //Enable the Sink circuitary for "u8PortNum" Port and configure it to drain
u16Voltage
    }
    //Conifgure Sink circuitary for u16Current current rating
}
```

10

10.2.9 Boot time Configuration

Macros

	Name	Description
⊸	MCHP_PSF_HOOK_BOOT_TIME_CONFIG	Updates the Type-C Configurable parameters(Power Role and Rp Current), number of PDO and PDOs parameter at boot time.

Structures

	Name	Description
🔶	_PortData_cfg	User configurable boot time structure.

10.2.9.1 _PortData_cfg Structure

C

```c
struct _PortData_cfg {
  UINT32 u32CfgData;
  UINT32 u32PDO[7];
  UINT8 u8PDOCnt;
};
```

Description

User Configurable Structure at boot time: This structure contains the Type-C configure parameters PDO count and PDOs parameters

Members

Members	Description
UINT32 u32CfgData;	Bits 2:0 - Port Role Bits 4:3 - Type-C Current Bits 5 - Port Enable/Disable
UINT32 u32PDO[7];	Source/Sink Capabilities PDOs
UINT8 u8PDOCnt;	PDO count of Source/Sink Capabilities

Remarks

None

10.2.9.2 MCHP_PSF_HOOK_BOOT_TIME_CONFIG

C

```c
#define MCHP_PSF_HOOK_BOOT_TIME_CONFIG(CfgGlobals)
```

Description

This function is called to update the configuration parameters of Type-C (Power Role and Rp Current), number of PDO and PDOs parameter at boot time(Stack initialization). This API must have a input parameter of PORT_CONFIG_DATA prototype (Structure Pointer to PORT_CONFIG_DATA).

Preconditions

None.

Parameters

Parameters	Description
CfgGlobals	Holds the structure pointer of the structure PORT_CONFIG_DATA

10

Returns

None.

Remarks

User definition of this Hook function is optional

Example

```
#define  MCHP_PSF_HOOK_BOOT_TIME_CONFIG(_CfgGlobals_)  STRAPS_PowerRole_Set(_CfgGlobals_)
void STRAPS_PowerRole_Set(PORT_CONFIG_DATA *PortConfigData);
void STRAPS_PowerRole_Set(PORT_CONFIG_DATA *PortConfigData)
{
    // Configure Cfg variables for Source or Sink
}
```

10.2.10 Hooks for Policy Manager

Macros

	Name	Description
–o	MCHP_PSF_HOOK_DPM_PRE_PROCESS	This hook is called before entering into the DPM state machine.

10.2.10.1 MCHP_PSF_HOOK_DPM_PRE_PROCESS

C

```
#define MCHP_PSF_HOOK_DPM_PRE_PROCESS(u8PortNum)
```

Description

This hook is called at the entry of DPM_RunStateMachine() API before executing the Type C state machine and policy engine state machine. USER_APPLICATION can define this function if a change is required in default device policy manager functionality or add a user defined state machine. Define relevant function that has one UINT8 argument without return type.

Preconditions

None.

Parameters

Parameters	Description
u8PortNum	Port number of the device. It takes value between 0 to (CONFIG_PD_PORT_COUNT-1).

Returns

None.

Remarks

User definition of this Hook function is optional

Example

```
    #define MCHP_PSF_HOOK_DPM_PRE_PROCESS(u8PortNum)
HookDevicePolicyManagerPreProcess(u8PortNum)
    void HookDevicePolicyManagerPreProcess(UINT8 u8PortNum);
    void HookDevicePolicyManagerPreProcess(UINT8 u8PortNum)
    {
        //any application related change or enhancement or user defined state machine
    }
```

10

10.2.11 Debug Hook Functions

Macros

	Name	Description
⊸	MCHP_PSF_HOOK_DEBUG_INIT	Initialization of debug module interface
⊸	MCHP_PSF_HOOK_DEBUG_INT32	Send INT32 to Debug interface
⊸	MCHP_PSF_HOOK_DEBUG_STRING	To pass a string to Debug interface
⊸	MCHP_PSF_HOOK_DEBUG_UINT16	Send UINT16 to Debug interface
⊸	MCHP_PSF_HOOK_DEBUG_UINT32	Send UINT32 to Debug interface
⊸	MCHP_PSF_HOOK_DEBUG_UINT8	Send a UINT8 to Debug interface

10.2.11.1 MCHP_PSF_HOOK_DEBUG_INIT

C

```
#define MCHP_PSF_HOOK_DEBUG_INIT
```

Description

This hook is called during initialization of PSF if CONFIG_HOOK_DEBUG_MSG is set to 1. Define relevant function to initialize the Debug interface used with no arguments without return type.

Preconditions

None.

Returns

None.

Remarks

User definition of this Hook function is mandatory when CONFIG_HOOK_DEBUG_MSG is declared as '1'.

Example

```
#define MCHP_PSF_HOOK_DEBUG_INIT()          uart_init()
void uart_init();
void uart_init()
{
    //Initialzes the uart module to send and receive a character
}
```

10.2.11.2 MCHP_PSF_HOOK_DEBUG_INT32

C

```
#define MCHP_PSF_HOOK_DEBUG_INT32(i32Val)
```

Description

This hook is called by stack to send a INT32 data to Debug interface. This API will be called if CONFIG_HOOK_DEBUG_MSG is set to 1. Define relevant function that has INT32 argument without return type.

Preconditions

None.

Parameters

Parameters	Description
i32Val	INT32 data to be sent to DEBUG_MODULE

Returns

None.

Remarks

User definition of this Hook function is mandatory when CONFIG_HOOK_DEBUG_MSG is declared as '1'.

Example

```
#define MCHP_PSF_HOOK_DEBUG_INT32(_i32Val_)        uart_write(i32Val)
void uart_write(INT32 i32Val);
void uart_write(INT32 i32Val)
{
    //Convert INT32 to character string and write to UART
}
```

10.2.11.3 MCHP_PSF_HOOK_DEBUG_STRING

C

```
#define MCHP_PSF_HOOK_DEBUG_STRING(pcharBuf)
```

Description

This hook is called by PSF to send a character string to DEBUG_MODULE. It will be called if CONFIG_HOOK_DEBUG_MSG is set to 1. Define relevant function that has CHAR pointer argument without return type.

Preconditions

None.

Parameters

Parameters	Description
pcharBuf	Pointer to the character buffer

Returns

None.

Remarks

User definition of this Hook function is mandatory when CONFIG_HOOK_DEBUG_MSG is declared as '1'.

Example

```
#define MCHP_PSF_HOOK_DEBUG_STRING(pcharBuf)        uart_write(pcharBuf)
void uart_write(char *chBuffer);
void uart_write(char *chBuffer)
{
    //Write character string to UART
}
```

10.2.11.4 MCHP_PSF_HOOK_DEBUG_UINT16

C

```
#define MCHP_PSF_HOOK_DEBUG_UINT16(u16Val)
```

Description

This hook is called by stack to send a UINT16 data to DEBUG_MODULE This API will be called if CONFIG_HOOK_DEBUG_MSG is set to 1. Define relevant function that has UINT16 argument without return type.

Preconditions

None.

10

Parameters

Parameters	Description
u16Val	UINT16 data to be sent to Debug interface

Returns

None.

Remarks

User definition of this Hook function is mandatory when CONFIG_HOOK_DEBUG_MSG is declared as '1'.

Example

```
#define MCHP_PSF_HOOK_DEBUG_UINT16(u16Val)     uart_write(u16Val)
void uart_write(UINT16 u16Val);
void uart_write(UINT16 u16Val)
{
    //Convert UINT16 to character string and write to UART
}
```

10.2.11.5 MCHP_PSF_HOOK_DEBUG_UINT32

C

```
#define MCHP_PSF_HOOK_DEBUG_UINT32(u32Val)
```

Description

This hook is called by stack to send a UINT32 data to DEBUG_MODULE. This API will be called if CONFIG_HOOK_DEBUG_MSG is set to 1. Define relevant function that has a UINT32 argument without return type.

Preconditions

None.

Parameters

Parameters	Description
u32Val	UINT32 data to be sent to Debug interface

Returns

None.

Remarks

User definition of this Hook function is mandatory when CONFIG_HOOK_DEBUG_MSG is declared as '1'.

Example

```
#define MCHP_PSF_HOOK_DEBUG_UINT32(u32Val)     uart_write(u32Val)
void uart_write(UINT32 u32Val);
void uart_write(UINT32 u32Val)
{
    //Convert UINT32 to character string and write to UART
}
```

10.2.11.6 MCHP_PSF_HOOK_DEBUG_UINT8

C

```
#define MCHP_PSF_HOOK_DEBUG_UINT8(u8Val)
```

Description

This hook is called by stack to send a UINT8 data to debug interface. This API will be called if CONFIG_HOOK_DEBUG_MSG is set to 1. Define relevant function that has UINT8 argument without return type.

10

Preconditions

None.

Parameters

Parameters	Description
u8Val	UINT8 data to be sent to Debug interface

Returns

None.

Remarks

User definition of this Hook function is mandatory when CONFIG_HOOK_DEBUG_MSG is declared as '1'.

Example

```
#define MCHP_PSF_HOOK_DEBUG_UINT8(u8Val)     uart_write(u8Val)
void uart_write(UINT8 u8Val);
void uart_write(UINT8 u8Val)
{
    //Convert UINT8 to character string and write to UART
}
```

10.2.12 PD Firmware Upgrade

Macros

	Name	Description
⊸	MCHP_PSF_HOOK_BOOT_FIXED_APP	MCHP_PSF_HOOK_BOOT_FIXED_APP shall perform necessary operation to switch from Updatable application to Fixed application.
⊸	MCHP_PSF_HOOK_BOOT_UPDATABLE_APP	MCHP_PSF_HOOK_BOOT_UPDATABLE_APP shall perform necessary operation to boot from the updatable application after a PDFU is successfully completed.
⊸	MCHP_PSF_HOOK_GETCURRENT_IMAGEBANK	To Return the Index of the Image Bank which is currently executing.
⊸	MCHP_PSF_HOOK_IS_PDFU_ALLOWED_NOW	
⊸	MCHP_PSF_HOOK_PROGRAM_FWBLOCK	Validate the Data block and program the data to the Memory, and return the status of the Programming Operation.
⊸	MCHP_PSF_HOOK_VALIDATE_FIRMWARE	To validate the Flashed Binary using a user defined validation method and return the status of the Firmware Validation Operation.

10.2.12.1 MCHP_PSF_HOOK_BOOT_FIXED_APP

C

```
#define MCHP_PSF_HOOK_BOOT_FIXED_APP
```

Description

Re-flash of the Updatable_Application image bank while currently executing in the Updatable Application image bank, requires switch to Fixed application for performing the upgrade. The application switching may include invalidating the Updatable_Application signatures (and/or) jump/reset for fresh boot from Fixed application.

Preconditions

This hook is invoked by the PD Firmware Update State-machine during the Reconfiguration phase(On reception PDFU_INITIATE Command), when the Updatable application is currently running.

Returns

No Return Value. During execution of this function the control shall be switched to the Fixed application.

Remarks

User definition of this Hook function is mandatory in the Updatable application when INCLUDE_PDFU is TRUE

Example

```c
#define MCHP_PSF_HOOK_BOOT_FIXED_APP()   Boot_Fixed_Application()
void Boot_Fixed_Application(void)
{
    EraseUpdatableAppSignature(); //Invalidate the Updatable app sign
    Reset(); //Reset to boot from Fixed app
}
```

10.2.12.2 MCHP_PSF_HOOK_BOOT_UPDATABLE_APP

C

```c
#define MCHP_PSF_HOOK_BOOT_UPDATABLE_APP
```

Description

As the flashing operation is executed from the Fixed application, once the PDFU process is complete it is necessary to switch to the newly upgraded updatable application. This hook definition shall implement necessary operations to safely exit the fixed application and boot from the updatable application. The application switching may include setting the valid Updatable_Application signatures (and) jump/reset for fresh boot from Updatable application.

Preconditions

This function is invoked by the PD Firmware Update State-machine during the Manifestation phase (On reception PDFU_INITIATE Command), when the Fixed application is currently running.

Returns

No Return Value. During execution of this function the control shall be switched to the Updatable application.

Remarks

User definition of this Hook function is mandatory in the Fixed application when INCLUDE_PDFU is TRUE.

Example

```c
#define MCHP_PSF_HOOK_BOOT_UPDATABLE_APP()   Boot_Updatable_Application()
void Boot_Updatable_Application(void)
{
    Reset(); //Reset to boot from Updatable app
}
```

10.2.12.3 MCHP_PSF_HOOK_GETCURRENT_IMAGEBANK

C

```c
#define MCHP_PSF_HOOK_GETCURRENT_IMAGEBANK 0x0
```

Description

This hook is called by PSF to get the Index of the image bank which is currently executing in the application. PSF follows "Architecture 2 - Fixed base application with updatable application image". In which the Fixed Application is Image Bank 1 and updatable Application is Image Bank 2.

Preconditions

This function is invoked by the PD Firmware Update State-machine during the Enumeration Phase (On reception PDFU_GET_FW_ID Command).

Returns

Returns UINT8 - the index of the Image Bank. It can take following values: 0x01 - IMAGE_BANK_BOOTLOADER 0x02 - IMAGE_BANK_FIXED_APP 0x03 - IMAGE_BANK_UPDATABLE_APP

Remarks

The User definition of the function is mandatory in both Fixed and Updatable application when INCLUDE_PDFU is TRUE.

Example 1

1. 0x01 - Corresponds to Bootloader Application

2. 0x02 - Corresponds to Fixed Application

3. 0x03 - Corresponds to Updatable Application

Example 2

```c
#define MCHP_PSF_HOOK_GETCURRENT_IMAGEBANK()  getCurrentImageBank()
UINT8 getCurrentImageBank(void)
{
    return u8ImageBankIndex;
}
```

10.2.12.4 MCHP_PSF_HOOK_IS_PDFU_ALLOWED_NOW

C

```c
#define MCHP_PSF_HOOK_IS_PDFU_ALLOWED_NOW 0
```

Description

MCHP_PSF_HOOK_IS_PDFU_ALLOWED_NOW specifies if PD Firmware Update can be currently allowed, based on the priority of the application tasks currently executing.

1. When the PD Firmware Update is allowed, the PDFU Initiator can perform firmware upgrade by the PDFU Sequence

2. When the PD Firmware Update is not allowed, the PDFU Initiator is responded with the error code during the Reconfiguration phase.

Example scenario of When PDFU cannot be allowed: Assuming a product with firmware update capability through CC and I2C as well. In an example, if the firmware upgrade through I2C is already in progress, then PDFU through CC interface shall not be allowed. To handle such situations, this macro shall return the current status of allow-ability of firmware upgrade.

Preconditions

This function is invoked by the PD Firmware Update State-machine during the Reconfiguration Phase (On reception PDFU_INITIATE Command).

Returns

UINT8 value - Shall return the run time status whether PDFU via CC is allowed now or not. 0x00 - PDFU Not Allowed. 0x01 - PDFU Allowed.

Remarks

User definition of this Hook function is mandatory in fixed as well as updatable when INCLUDE_PDFU is TRUE.

Example

```c
#define MCHP_PSF_HOOK_IS_PDFU_ALLOWED_NOW   isPdfuAllowedNow()
UINT8 isPdfuAllowedNow(void)
{
    return u8PdfuAllow;
}
```

10

10.2.12.5 MCHP_PSF_HOOK_PROGRAM_FWBLOCK

C

```
#define MCHP_PSF_HOOK_PROGRAM_FWBLOCK(u8pObj,u16Len) 0
```

Description

This hook is invoked during the Transfer Phase on the successful reception event of every PDFU_DATA packet. It is responsible for updating the Firmware data to the memory and identifying any errors during the Firmware flash.

Preconditions

Only during the Policy Engine State-Reconfigure Phase or Transfer phase this function hook will be invoked.

Parameters

Parameters	Description
u8pObj	UINT8 Pointer to PDFU_DATA packet payload Buffer.
u8pObj[0]	Reserved field Contains PD FW Version.
u8pObj[1]	Reserved field Contains Msg Type which is PDFU_DATA 0x83.
u8pObj[2]	LSB of Data Block Index.
u8pObj[3]	MSB of Data Block
Index u8pObj[4..260]	Firmware Image data upto 256 bytes where the Data block index is used to calculate the Physical memory address to which the current data block corresponds to 16 bit parameter.
u16Len	Indicates the length of the Firmware data contained in the packet.

Returns

Returns ePE_PDFU_RESPONSE_CODE Type Return Value - The Status of the Program Operation.

1. ePE_FWUP_OK - Upon successful flash operation.

2. ePE_FWUP_errVERIFY - When verification of the flash operation failed.

3. ePE_FWUP_errADDRESS - When data block index is out of range.

Remarks

User definition of this Hook function is mandatory when INCLUDE_PDFU is TRUE.

Example

```
#define MCHP_PSF_HOOK_PROGRAM_FWBLOCK(u8pObj, u16Len)
            PDFW_ProcessPDFUDataRequest(u8pObj, u16Len)
ePE_PDFU_RESPONSE_CODE PDFW_ProcessPDFUDataRequest( UINT8 u8RequestBuffer,
                                    UINT16 u16RequestLength)
{
    UINT16 u16DataBlockIndex = *((UINT16*)&u8RequestBuffer[2]);
    u32ProgAddr = CalculateAddress(u16DataBlockIndex);
    if( u32ProgAddr < 0xFFFFu )
    {
        ProgramMemoryCB(u32ProgAddr, &u8RequestBuffer[4u],u16RequestLength);
        ReadMemoryCB(u32ProgAddr, &u8ResponseBuffer[0u],u16RequestLength);
        //Compare data written and read
        if (0 == MCHP_PSF_HOOK_MEMCMP(&u8ResponseBuffer[0],
                            &u8RequestBuffer[4], u16RequestLength))
        {
            //Set the status as OK
            u8Status = ePE_FWUP_OK;
        }
        else
        {
            //Verification Stage failure
            u8Status = ePE_FWUP_errVERIFY;
        }
    }
    else
```

```
        {
            u8Status = ePE_FWUP_errADDRESS;
        }

        return u8Status;
    }
```

10.2.12.6 MCHP_PSF_HOOK_VALIDATE_FIRMWARE

C

```
#define MCHP_PSF_HOOK_VALIDATE_FIRMWARE 0
```

Description

This hook is invoked during the validation phase on reception of every PDFU Validation Request. It is responsible for validating the Firmware data in the memory. It shall return the progress status of the Validation on every invocation. If the Status indicates "On going" then the Validation command will be responded with the configured Wait time CONFIG_VALIDATION_PHASE_WAITTIME. Validation Method can be any custom method as required by the User.

Preconditions

Multiple invocations of the function hook is possible from PDFU Validation phase. Until the Validation process is complete, for every request of PDFU_VALIDATION command this function will be invoked.The definition of this function shall include 1) Starting the Validation process on the First call, 2) Returning the Status of the Validation process during subsequent invocation of the function.

Returns

Returns the UINT8 Status of the Validation Operation. It take following values

0x00u - PE_FWUP_VALIDATION_SUCCESSFUL

0x01u - PE_FWUP_VALIDATION_INPROGRESS

0x02u - PE_FWUP_VALIDATION_FAILURE

Remarks

User definition of this Hook function is mandatory when INCLUDE_PDFU is TRUE

Example

```
#define MCHP_PSF_HOOK_VALIDATE_FIRMWARE()   PDFW_ProcessPDFUDataRequest()
UINT8 PDFW_ProcessPDFUValidateRequest(void)
{
    The definition of this function shall include
    1) Starting the Validation process on the First call,
    2) Returning the Status of the Validation process during subsequent invocation of
the function.
}
```

10.3 APIs to be called by the User application

Functions

	Name	Description
⇛♦	MchpPSF_Init	PSF initialization API
⇛♦	MchpPSF_PDTimerHandler	PD Timer Interrupt Handler
⇛♦	MchpPSF_UPDIrqHandler	UPD350 IRQ Interrupt Handler
⇛♦	MchpPSF_RUN	PSF state machine run API

Description

PSF APIs have to be called by the SoC User_Application for PSF to run as well as to get to know about the Timer and

IRQ_N interrupt from the UPD350. This section explains them in detail

10.3.1 MchpPSF_Init

C

```
UINT8 MchpPSF_Init();
```

Description

This API should be called by the SOC layer to initialize the PSF stack and UPD350 Hardware.

Preconditions

API should be called before MchpPSF_RUN().

Returns

TRUE - Stack and UPD350 HW successfully initialized. FALSE - Stack and UPD350 HW initialization failed.

Remarks

For the current PSF implementation, return value is not used. API called with void. With SAMD20 environment configured for CPU frequency 48MHZ, this API took maximum of 3.488ms and 6.182ms execution time for 1 and 2 port solution respectively.

10.3.2 MchpPSF_PDTimerHandler

C

```
void MchpPSF_PDTimerHandler();
```

Description

This API is used to handle the PD Timer (Software timer) Interrupt, User should call this API whenever the hardware timer interrupt triggered.

Preconditions

This API should be called inside the Hardware timer ISR.

Returns

None

Remarks

With SAMD20 environment configured for CPU frequency 48MHZ, this API took maximum of 262us execution time for both 1 and 2 port solution.

10.3.3 MchpPSF_UPDIrqHandler

C

```
void MchpPSF_UPDIrqHandler(
    UINT8 u8PortNum
);
```

Description

This API handles the UPD350 IRQ_N Interrupt, User should call this API when the IRQ line interrupt triggered to the SOC. This API will services and then clear the Alert interrupt for corresponding port.

Preconditions

This API should be called inside the GPIO ISR for IRQ interrupt

Parameters

Parameters	Description
u8PortNum	Port number of the device. It takes value between 0 to (CONFIG_PD_PORT_COUNT-1).

Returns

None

Remarks

With SAMD20 environment configured for CPU frequency 48MHZ, this API took maximum of 3.98us and 5.8us execution time for 1 and 2 port solution respectively.

10.3.4 MchpPSF_RUN

C

```
void MchpPSF_RUN();
```

Description

This API is to run the PSF state machine. For single threaded environment, it should be called repeatedly within a while(1).

Preconditions

API should be called only after MchpPSF_Init().

Returns

None

Remarks

In SAMD20 environment where CPU frequency is configured for 48MHZ and single threaded environment where MchpPSF_RUN is called in a while(1) loop, it took maximum of 74.57us and 0.1439 ms execution time for 1 port and 2 source port solution respectively. In Multi threaded SAMD20 configured for 48MHz environment, MchpPSF_RUN has to be called for every 2ms for Successful 2-Port Source only operation.

10

11 Notification callback from PSF

Enumerations

	Name	Description
📛	MCHP_PSF_NOTIFICATION	PSF notification enum

Macros

	Name	Description
⊸	MCHP_PSF_NOTIFY_CALL_BACK	Notifies the USER_APPLICATION about various PD events from PSF.

Description

This sections gives details on the all the notifications given by PSF through its callback.

11.1 MCHP_PSF_NOTIFICATION Enumeration

C

```
enum MCHP_PSF_NOTIFICATION {
  eMCHP_PSF_TYPEC_DETACH_EVENT = 1,
  eMCHP_PSF_TYPEC_CC1_ATTACH,
  eMCHP_PSF_TYPEC_CC2_ATTACH,
  eMCHP_PSF_UPDS_IN_IDLE,
  eMCHP_PSF_VCONN_PWR_FAULT,
  eMCHP_PSF_VBUS_PWR_FAULT
};
```

Description

eMCHP_PSF_NOTIFICATION enum defines the all the notifications PSF can notify via MCHP_PSF_NOTIFY_CALL_BACK.

eMCHP_PSF_TYPEC_DETACH_EVENT: This event is posted by PSF Type C state machine when port partner Detach event is detected.

eMCHP_PSF_TYPEC_CC1_ATTACH: This event is posted by PSF Type C state machine when port partner Type C attach is detected in CC1 pin.

eMCHP_PSF_TYPEC_CC2_ATTACH: This event is posted by PSF Type C state machine when port partner Type C attach is detected in CC2 pin.

eMCHP_PSF_UPDS_IN_IDLE: This event is posted by Power management control. PSF runs an algorithm backend for Power management control. If there is no activity in UPD350 for CONFIG_PORT_UPD_IDLE_TIMEOUT_MS corresponding UPD350 is put to low power mode. When all the UPD350 present in the system enters low mode, eMCHP_PSF_UPDS_IN_IDLE is posted. User can put SOC in low power mode as required on this notification. This notification occurs only when INCLUDE_POWER_MANAGEMENT_CTRL defined as 1.

eMCHP_PSF_VCONN_PWR_FAULT: UPD350 has VCONN comparators to detect VCONN OCS faults. This event is notified when VCONN OCS fault is detected by UPD350. For this notification, PSF expects a return value to decide whether to handle the fault occurred. When user returns TRUE for VCONN power fault, Incase of explicit contract, if VCONN power fault count is less than CONFIG_MAX_VCONN_POWER_FAULT_COUNT, PSF DPM power fault manager handles it by sending Hard Reset. If the count exceeds max fault count,VCONN is powered off until physical detach of port partner. Incase of implicit contract, PSF handles by entering TypeC Error Recovery. This notification occurs only when INCLUDE_POWER_FAULT_HANDLING is defined as 1.

eMCHP_PSF_VBUS_PWR_FAULT : PSF notifies all VBUS power fault VBUS Over voltage, VBUS under voltage, VBUS OCS via this notification. For this notification, PSF expects a return value to decide whether to handle the fault

11

occurred.When user returns TRUE for power fault, Incase of explicit contract, if power fault count is less than CONFIG_MAX_VBUS_POWER_FAULT_COUNT, PSF DPM power fault manager handles it by sending Hard Reset. When the power fault count exceeds the max fault count,CC termination on the port is removed until the physical detach of the port partner. Incase of implicit contract, PSF handles by entering TypeC Error Recovery. This notification occurs only when INCLUDE_POWER_FAULT_HANDLING is defined as 1.

Members

Members	Description
eMCHP_PSF_TYPEC_DETACH_EVENT = 1	Detach event has occurred
eMCHP_PSF_TYPEC_CC1_ATTACH	Port partner attached at CC1 orientation
eMCHP_PSF_TYPEC_CC2_ATTACH	Port partner attached at CC2 orientation
eMCHP_PSF_UPDS_IN_IDLE	All the UPD350s are in Idle
eMCHP_PSF_VCONN_PWR_FAULT	VCONN Power Fault has occurred
eMCHP_PSF_VBUS_PWR_FAULT	VBUS Power Fault has occurred

Remarks

None

11.2 MCHP_PSF_NOTIFY_CALL_BACK

C

```
#define MCHP_PSF_NOTIFY_CALL_BACK(u8PortNum, ePSFNotification)
```

Description

This hook is called by the various modules of PSF to notify the USER_APPLICATION about different PD events such as Type-C Attach and Detach , Type-C Orientation. USER_APPLICATION can define this hook function if it wants external handling of the PD events. Define relevant function that has UINT8, eMCHP_PSF_NOTIFICATION argument without return type.

Preconditions

None.

Parameters

Parameters	Description
u8PortNum	Port number of the device. It takes value between 0 to (CONFIG_PD_PORT_COUNT-1).
ePSFNotification	Type of Notification occurred inside the stack. This argument can take one of the values from enum eMCHP_PSF_NOTIFICATION.

Returns

UINT8 - Except for eMCHP_PSF_VCONN_PWR_FAULT and eMCHP_PSF_VBUS_PWR_FAULT the return value is ignored by PSF. For eMCHP_PSF_VCONN_PWR_FAULT and eMCHP_PSF_VBUS_PWR_FAULT event, user can return TRUE - if the Power fault shall be handled by PSF FALSE - if the Power fault occurrence is ignored.

Remarks

User definition of this Hook function is mandatory

Example

```
#define MCHP_PSF_NOTIFY_CALL_BACK(u8PortNum, ePSFNotification)
            HookNotifyPDEvents(u8PortNum, ePSFNotification)
void HookNotifyPDEvents(UINT8 u8PortNum, eMCHP_PSF_NOTIFICATION ePSFNotification);
void HookNotifyPDEvents(UINT8 u8PortNum, eMCHP_PSF_NOTIFICATION ePSFNotification)
```

11

```
{
    // Return for Power fault notification
    // Implement user specific application as required
}
```