MICROCHIP

USB Power Delivery Software Framework(PSF) User Guide

Version 1.03

Table of Contents

1 Revision History

REV	DATE	DESCRIPTION OF CHANGE
0.92	11-Dec-19	Initial Revision
0.95	1-Jan-20	Revised multiple sections to improve document flow
1.00	26-Feb-20	Updated document version to align with v1.00 release
1.01	19-Mar-20	Updated the document to align with recent PSF_Config header file updates where various PSF Configuration Options are moved to global configuration structure and Type C and Policy Engine Timeouts are moved inside PSF stack.
1.02	7-Apr-20	Sink functionality added. GotoMin support, Sink related configuration registers updated; MCHP_PSF_HOOK_DRIVE_DAC_I() added.
1.03	21-Apr-20	Added EN_SINK functionality. Updated comments for elements of PortCfgStatus Structure accordingly.

1

2 Introduction

USB Power Delivery Software Framework (PSF) with USB-PD Port Controller UPD350 is a full-featured, adaptive USB-PD compliant to the USB-PD 3.0 Specification. PSF is MCU-agnostic; a system designer may choose from a wide catalog Microchip MCU/SoCs to meet specific system requirements while optimizing cost and PCB space. Versatility is achieved through flexible configuration and hardware abstraction. Most major PD functions, like power roles and capabilities, data role and capabilities, product/vendor IDs, and control I/O configuration are achieved through a simple configuration header file..

This documents serves a user guide covering:

- PSF overview and high level architecture
- PSF stack directory structure
- Supported/Not supported features & messages
- It provides a high level configurable options
- Requirements for new platform integration
- 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 and USB Type-C Specification Revision 1.3
- Multi-port support upto 4 ports.
- USB-PD Source-only or Sink-only port specific configurability
- FW update through CC support compliant to PD FW Update Specification Revision 1.0

Check Supported/Non Supported PD Features for full details on features supported by PSF.

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 USB Type-C™ Power Delivery 3.0 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
DFP	Downstream Facing Port
UFP	Upstream Facing Port
PDFU	Power Delivery Firmware update
PDO	Power Data Object
FRS	Fast Role Swap
PPS	Programmable Power Supply
PB	Power Balancing
PT	Power Throttling

3 Legal Information

Software License Agreement

Copyright © [2019-2020] 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.

4 PSF Architecture Overview

PSF is a layered Software framework that enables Power delivery support with any SoC interfaced to UPD350. PSF also provides support for multiple ports, where each port could be configured in a manner independent of other ports. PSF also supports multiple compile time and run time configuration options to tailor the stack to the needs of USER_APPLICATION
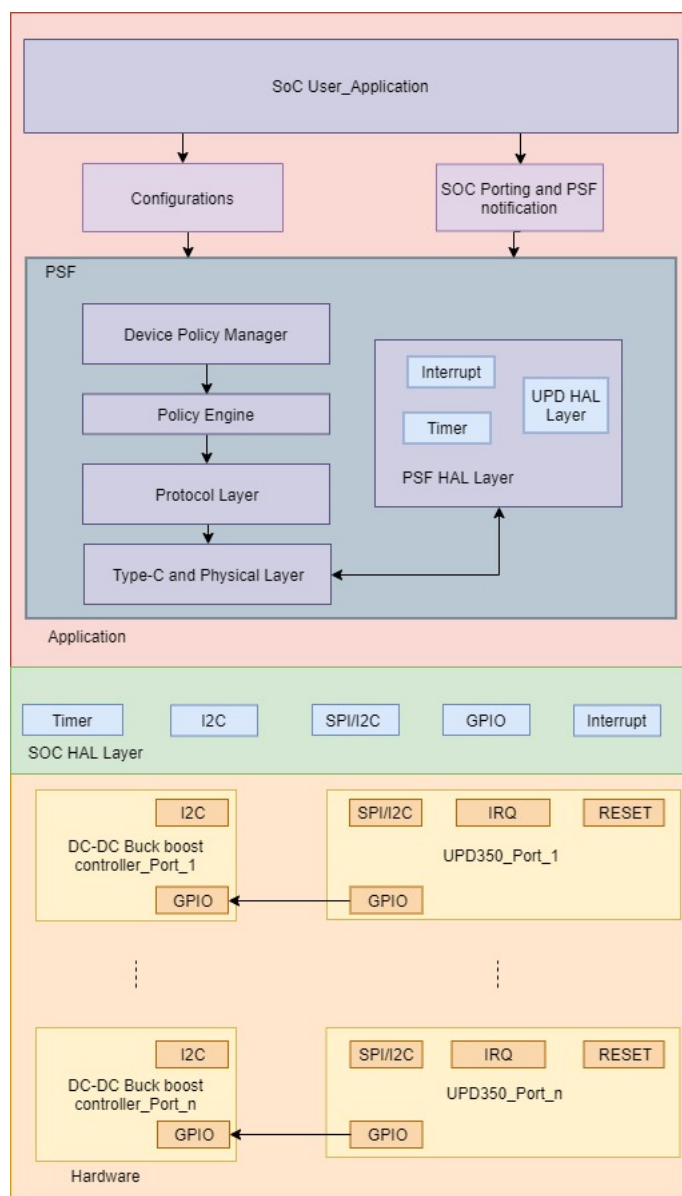
SoC User_Application:

- 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 variable VBUS voltages (i.e.: 5V, 9V, 15V, or 20V), an external DC-DC buck-boost controller is required. It can either be driven by UPD350 PIOs or SoC's I2C controller. UPD350 requires the 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 this release of PSF. Future releases may extend PSF feature set and the supported messages.

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
Dynamic Power Balancing	Supported
Power Throttling	Not Supported
PPS	Not Supported
FRS support	Not Supported
UVDM & USB Type-C Bridging	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 Messages

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	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 Messages

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 Messages

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, PSF configuration files of Source Lite, Source Pro and Sink sample applications.
  - Source Lite - PSF Source only sample application with no add-on features
  - Source Pro - PSF Source sample application with value-add features such as Power Balancing, PPS, etc.
  - Sink - PSF Sink sample application
- ../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_Sink
    PSF_EVB_Source_Lite
    PSF_EVB_Source_Pro
  SOC_Portable
    New_SOC
    SAMD20
  Source
    include
    PB
    PD3_0
```
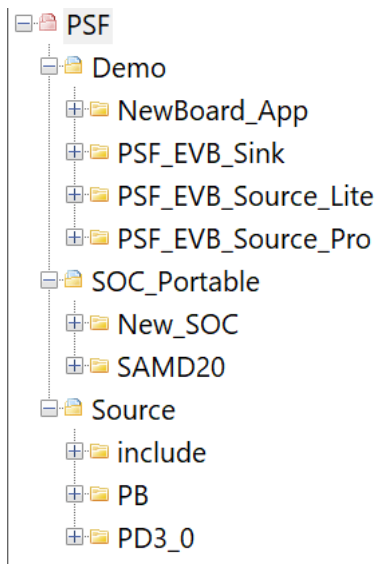
7.1 Source folder

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
- ..\PSF\Source\PB - contains all the source and header files for Power Balancing funtionality

File descriptions of all the files present under include and PD3_0

File Name	Application	File description
policy_manager.c policy_manager.h	Device Policy Manager (DPM)	Maintains DPM functionality
policy_engine.h policy_engine.c policy_engine_src.c policy_engine_snk.c	Policy Engine	• 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.c protocol_layer.h	Protocol Layer	APIs for Protocol layer functionality
typeC_control.c typeC_control.h	Type-C Connector Management	Type-C control functionality & connector management
pd_timer.c pd_timer.h	Timer management	Contains APIs to manage multiple Software Timers
upd_interrupts.c upd_interrupts.h	Interrupts Management	UPD350 Alert Interrupt management (port specific)
debug.h	Debug support	Debug interface support APIs file
upd_hw.c upd_hw.h	UPD350 Hardware	APIs to access for UPD350 Hardware GPIO, Registers, PIO override
int_globals.c int_globals.h	Internal globals	Maintains globals internal to PSF.
generic_defs.h	Generic Defines	contains generic defines for data types
timer_defs.h	Timer Defines	contains Type C and Policy Engine Timeout configurations
psf_stdinc.h	Standard include file	Standard include file for PSF
pd_main.c	PSF main source file	contains APIs to be called by User_Application
policy_engine_fwup.c policy_engine_fwup.h	Power Deliver Firmware update	Maintains PDFU functionality
portpower_control.c portpower_control.h	Port Power control	Maintains PSF default port power control support
pb_helper.c pb_mngr.c pb_mngr.h	Power Balancing	Maintains Power Balancing related APIs that are used to handle PPM events and helper functions needed for the same.
ProjectVersion.h	PSF version	Maintains PSF release version

7.2 SOC_Portable folder

SOC_Portable contains all the drivers porting files required for PSF functionality from SoC layer and I2C DC-DC files. It varies depending on the SoC and user application. The 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 PSF sample applications specific to the boards. It contains:

- project files

- Application and board specific files and documents

- PSF_Config header file.

- Boot time configuration file specific to sample application.

With these files the user can build and exercise an application on the PSF evaluation platform or other custom-designed hardware platform. PD functionality can be configured with the "PSF_Config.h" header file located at "..\Demo\NewBoard_App\PSF_Config.h". This file is used to configure the PD application to meet specific system requirements and feature-set. In Boot time configuration file, user must write a function to initialize all the needed configuration parameters.

8 PSF Configuration Options

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

This release of PSF allows configuration of the following:

- Number of ports
- Power Role - Source only or Sink only
- Data Role - DFP or UFP
- Port Power control management
- Source & Sink PDOs
- Power Balancing parameters
- 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 Handling 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.
⊷	INCLUDE_POWER_BALANCING	Power Balancing support code inclusion.
⊷	INCLUDE_POWER_THROTTLING	Power Throttling support code inclusion.
⊷	INCLUDE_PD_SOURCE_PPS	PPS support 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

8.1.1 INCLUDE_PD_3_0

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 and 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.

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. For Source Operation, it is mandatory to define this macro as '1'. When INCLUDE_PD_SOURCE is defined as '1', define this macro as '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 uses this feature to disable EN_VBUS(in case of source operation) or EN_SINK(in case of sink operation) 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 or EN_SINK 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, EN_SINK 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)
```

8.1.7 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 if the PSF application doesnot use Firmware update feature.

Remarks

Recommended default value is 0 unless Firmware update feature is used. It is mandatory to have INCLUDE_PD_3_0 is defined as '1' when INCLUDE_PDFU is '1'.

Example

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

8.1.9 INCLUDE_POWER_BALANCING

C

```
#define INCLUDE_POWER_BALANCING 1
```

Description

Setting the INCLUDE_POWER_BALANCING as 1 enables PSF to include the PD Power Balancing functionality at compile time. User can set this define to 0 to reduce code size if none of the PD enabled Source ports in the system require Power

Balancing functionality.

Remarks

Recommended default value is 1. For INCLUDE_POWER_BALANCING to be 1, INCLUDE_PD_SOURCE shall be set to 1.

Example

```
#define INCLUDE_POWER_BALANCING 1(Include Power Balancing functionality in PSF)
#define INCLUDE_POWER_BALANCING 0(Exclude Power Balancing functionality from PSF)
```

8.1.10 INCLUDE_POWER_THROTTLING

C

```
#define INCLUDE_POWER_THROTTLING 1
```

Description

Setting the INCLUDE_POWER_THROTTLING as 1 enables PSF to include the Power Throttling(PT) feature at compile time. User can set this define to 0 to reduce code size if none of the Source ports in the system require PT functionality.

Remarks

Recommended default value is 1. For INCLUDE_POWER_THROTTLING to be 1, INCLUDE_PD_SOURCE shall be set to 1.

Example

```
#define INCLUDE_POWER_THROTTLING    1(Include PT functionality in PSF)
#define INCLUDE_POWER_THROTTLING    0(Exclude PT functionality from PSF)
```

8.1.11 INCLUDE_PD_SOURCE_PPS

C

```
#define INCLUDE_PD_SOURCE_PPS 1
```

Description

Setting the INCLUDE_PD_SOURCE_PPS as 1 enables PSF to include the Programmable Power Supply(PPS) feature at compile time. User can set this define to 0 to reduce code size if none of the Source ports in the system require PPS functionality.

Remarks

Recommended default value is 1. For INCLUDE_PD_SOURCE_PPS to be 1, INCLUDE_PD_SOURCE and INCLUDE_PD_3_0 shall be set to 1.

Example

```
#define INCLUDE_PD_SOURCE_PPS   1(Include PPS functionality in PSF)
#define INCLUDE_PD_SOURCE_PPS   0(Exclude PPS functionality from PSF)
```

8.2 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.

8.2.1 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 support 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. For each port defined by this macro approximately 500Bytes of Data RAM is consumed.

Example

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

8.2.2 CONFIG_DEFINE_UPD350_HW_INTF_SEL

C

```
#define CONFIG_DEFINE_UPD350_HW_INTF_SEL CONFIG_UPD350_SPI
```

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 input value.

CONFIG_UPD350_SPI - SPI is the communication interface between SOC and UPD350. SPI interface is supported by UPD350 B and D parts alone.

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. Using mixed interfaces for individual ports is not supported (i.e.: SPI for Port 1 and I2C for Port 2).

If the target for PSF is a UPD301C device, SPI must always be selected. I2C is not an option for UPD301C due to the physical bonding of the UPD301C.

Example

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

8

8.3 Configuration and Status Registers

Structures

	Name	Description
◆	_GlobalCfgStatusData	This structure contains the system level, Port specific configurations and Status parameters of PSF for Type C, PD, PB, PT and PPS parameters. gasCfgStatusData is the defined variable of this structure.
◆	_PortCfgStatus	This structure contains port specific Type C and PD configuration and status parameters. sPerPortData is referred from _GlobalCfgStatusData.
◆	_PBPortCfgStatus	This structure contains port specific Power Balancing Configuration parameters. sPBPerPortData is referred from _GlobalCfgStatusData.
◆	_PPSPortCfgStatus	This structure contains port specific PPS configuration parameters. sPPSPerPortData is referred from _GlobalCfgStatusData.

Variables

	Name	Description
◆	gasCfgStatusData	This is an instance of _GlobalCfgStatusData.

Description

Configuration and Status registers help users to control the operation of the PSF ports and to know the run time status information of the ports defined in CONFIG_PD_PORT_COUNT.

8.3.1 _GlobalCfgStatusData Structure

C

```
struct _GlobalCfgStatusData {
  UINT8 u8MinorVersion;
  UINT8 u8MajorVersion;
  UINT8 u8HWVersion;
  UINT8 u8SiVersion;
  UINT8 u8aManfString[8];
  UINT8 u8PSFMajorVersion;
  UINT8 u8PSFMinorVersion;
  UINT8 u8aReserved6[2];
  UINT16 u16ProducdID;
  UINT16 u16VendorID;
  UINT16 u16ProductTypeVDO;
  UINT16 u16ProductVDO;
  UINT16 u16CertStatVDO;
  UINT16 u16IDHeaderVDO;
  PORT_CFG_STATUS sPerPortData[CONFIG_PD_PORT_COUNT];
  UINT8 u8PBEnableSelect;
  UINT8 u8aReserved7[3];
  UINT8 u8PwrThrottleCfg;
  UINT8 u8aReserved8[3];
  UINT16 u16SystemPowerBankA;
  UINT16 u16MinPowerBankA;
  UINT16 u16SystemPowerBankB;
  UINT16 u16MinPowerBankB;
  UINT16 u16SystemPowerBankC;
  UINT16 u16MinPowerBankC;
  UINT16 u16SharedPowerCapacity;
  UINT16 u16Reserved2;
  PB_PORT_CFG_STATUS sPBPerPortData[CONFIG_PD_PORT_COUNT];
  PPS_PORT_CFG_STATUS sPPSPerPortData[CONFIG_PD_PORT_COUNT];
};
```

8

Description

This structure contains global configuration and status parameters that are either Integer Datatypes, Bit-Mapped bytes or another structure.

1. Members that are Integer data types:

Name	Size in Bytes	R/W Config time	R/W Run time	Description
u8MinorVersion	1	R/W	R	Defines Structure Minor Version
u8MajorVersion	1	R/W	R	Defines Structure Major Version
u8HWVersion	1	R/W	R	Defines Hardware Version
u8SiVersion	1	R/W	R	Defines Silicon Version
u8aManfString[8]	1	R/W	R	Defines Manufacturer String
u8PSFMajorVersion	1	R/W	R	Defines PSF Stack Major Version
u8PSFMinorVersion	1	R/W	R	Defines PSF Stack Minor Version
u16ProducdID	2	R/W	R	• Defines 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. • User definition of this macro is mandatory when INCLUDE_PDFU is defined as 1. It should always be two byte wide.
u16VendorID	2	R/W	R	• Defines the 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. • User definition of this macro is mandatory when INCLUDE_PDFU is defined as 1. It should always be two byte wide.
u16ProductTypeVDO	2	R/W	R	Defines Product Type VDO
u16ProductVDO	2	R/W	R	Defines Product VDO
u16CertStatVDO	2	R/W	R	Defines Cert Stat VDO
u16IDHeaderVDO	2	R/W	R	Defines ID Header VDO
u8PwrThrottleCfg	1	R/W	R/W	• Defines the currently selected Power Bank. • Possible values are, 1. 0x00 = Bank A 2. 0x01 = Bank B 3. 0x10 = Bank C 4. 0x11 = Shutdown mode • This variable is used only when either of INCLUDE_POWER_BALANCING or INCLUDE_POWER_THROTTLING is set to '1'.

8

u16SystemPowerBankA	2	R/W	R	• Defines the Total System Power of Bank A. Each unit is 0.25W
				• Sample values are,
				  1. 0x0001 = 0.25W
				  2. 0x01E0 = 120W
				  3. 0x0320 = 200W
				• A setting of 0x0000 and 0x0321-0xFFFF is invalid.
				• This variable is used only when either of INCLUDE_POWER_BALANCING or INCLUDE_POWER_THROTTLING is set to '1'.

u16MinPowerBankA	2	R/W	R	• Defines the Guaranteed minimum Power of Bank A. Each unit is 0.25W
				• Sample values are,
				  1. 0x0001 = 0.25W
				  2. 0x003C = 15W
				  3. 0x0190 = 100W
				• A setting of 0x0000 and 0x0191-0xFFFF is invalid.
				• This variable is used only when either of INCLUDE_POWER_BALANCING or INCLUDE_POWER_THROTTLING is set to '1'.

u16SystemPowerBankB	2	R/W	R	• Defines the Total System Power of Bank B. Each unit is 0.25W
				• Sample values are,
				  1. 0x0001 = 0.25W
				  2. 0x01E0 = 120W
				  3. 0x0320 = 200W
				• A setting of 0x0000 and 0x0321-0xFFFF is invalid.
				• This variable is used only when either of INCLUDE_POWER_BALANCING or INCLUDE_POWER_THROTTLING is set to '1'.

u16MinPowerBankB	2	R/W	R	• Defines the Guaranteed minimum Power of Bank B. Each unit is 0.25W
				• Sample values are,
				  1. 0x0001 = 0.25W
				  2. 0x003C = 15W
				  3. 0x0190 = 100W
				• A setting of 0x0000 and 0x0191-0xFFFF is invalid.
				• This variable is used only when either of INCLUDE_POWER_BALANCING or INCLUDE_POWER_THROTTLING is set to '1'.

8

u16SystemPowerBankC	2	R/W	R	• Defines the Total System Power of Bank C. Each unit is 0.25W
				• Sample values are,
				  1. 0x0001 = 0.25W
				  2. 0x01E0 = 120W
				  3. 0x0320 = 200W
				• A setting of 0x0000 and 0x0321-0xFFFF is invalid.
				• This variable is used only when either of INCLUDE_POWER_BALANCING or INCLUDE_POWER_THROTTLING is set to '1'.

u16MinPowerBankC	2	R/W	R	• Defines the Guaranteed minimum Power of Bank C. Each unit is 0.25W
				• Sample values are,
				  1. 0x0001 = 0.25W
				  2. 0x003C = 15W
				  3. 0x0190 = 100W
				• A setting of 0x0000 and 0x0191-0xFFFF is invalid.
				• This variable is used only when either of INCLUDE_POWER_BALANCING or INCLUDE_POWER_THROTTLING is set to '1'.

u16SharedPowerCapacity	2	R/W	R	• Defines the currently available shared power capacity from which power is allocated to ports that are not yet in a valid negotiated contract.
				• The shared power capacity is dynamically adjusted as ports are connected and disconnected. Each unit is 0.25W
				• Sample values are,
				  1. 0x0001 = 0.25W
				  2. 0x003C = 15W
				  3. 0x0190 = 100W
				• This variable is used only when either of INCLUDE_POWER_BALANCING or INCLUDE_POWER_THROTTLING is set to '1'.

u8aReserved6[2]	2			Reserved
u8aReserved7[3]	3			Reserved
u8aReserved8[3]	3			Reserved
u16Reserved2	2			Reserved

2. Members that are Bit-Mapped bytes:

a. u8PBEnableSelect:

u8PBEnableSelect defines PB Enable/Disable for the system and also the Power Balancing algorithm. It's size is 1 byte.

8

Bit	R/W Config time	R/W Run time	Description
3:0	R/W	R	Selection of Power Balancing Algorithm • 0000 = First Come First Serve • 0001 = Last Plugged Gets Priority • 0010-1111 = Reserved
4	R/W	R	PD Power balancing Enable/Disable for the system • 0 = PD Balancing is disabled • 1 = PD Balancing is enabled
7:5			Reserved

3. Members that are another structures:

Structure	Description
sPerPortData	Includes Type C and PD parameters of a port, say default Source PDOs, default Sink PDOs, currently negotiated voltage and current values, under voltage and over voltage threshold values, etc., Tag for this structure is _PortCfgStatus.
sPPSPerPortData	Includes PPS parameters of a port, say PPS Enable/Disable option and list of Augmented PDOs supported. Tag for this structure is _PPSPortCfgStatus.
sPBPerPortData	Includes Power Balancing parameters of a port, say maximum power and maximum current. Tag for this structure is _PBPortCfgStatus.

Remarks

None

8.3.2 _PortCfgStatus Structure

C

```
struct _PortCfgStatus {
  UINT32 u32CfgData;
  UINT32 u32aSourcePDO[7];
  UINT32 u32aSinkPDO[7];
  UINT32 u32aNewPDO[7];
  UINT32 u32aAdvertisedPDO[7];
  UINT32 u32aPartnerPDO[7];
  UINT32 u32RDO;
  UINT32 u32PortConnectStatus;
  UINT16 u16AllocatedPowerIn250mW;
  UINT16 u16NegoVoltageIn50mV;
  UINT16 u16NegoCurrentIn10mA;
  UINT16 u16MaximumOperatingCurInmA;
  UINT16 u16PortIOStatus;
  UINT16 u16PortStatusChange;
  UINT16 u16PortIntrMask;
  UINT16 u16PowerGoodTimerInms;
  UINT16 u16aMinPDOPreferredCurInmA[7];
  UINT16 u16MinimumOperatingCurInmA;
  UINT16 u16DAC_I_MaxOutVoltInmV;
  UINT16 u16DAC_I_MinOutVoltInmV;
  UINT16 u16DAC_I_CurrentInd_MaxInA;
  UINT16 u16Reserved1;
  UINT8 u8SourcePDOCnt;
  UINT8 u8SinkPDOCnt;
```

8

```
    UINT8 u8NewPDOCnt;
    UINT8 u8AdvertisedPDOCnt;
    UINT8 u8PartnerPDOCnt;
    UINT8 u8NewPDOSelect;
    UINT8 u8SinkConfigSel;
    UINT8 u8FaultInDebounceInms;
    UINT8 u8OCSThresholdPercentage;
    UINT8 u8OVThresholdPercentage;
    UINT8 u8UVThresholdPercentage;
    UINT8 u8VCONNOCSDebounceInms;
    UINT8 u8VBUSMaxFaultCnt;
    UINT8 u8VCONNMaxFaultCnt;
    UINT8 u8Pio_FAULT_IN;
    UINT8 u8Mode_FAULT_IN;
    UINT8 u8Pio_VBUS_DIS;
    UINT8 u8mode_VBUS_DIS;
    UINT8 u8aReserved1[2];
    UINT8 u8Pio_EN_VBUS;
    UINT8 u8Mode_EN_VBUS;
    UINT8 u8Pio_DC_DC_EN;
    UINT8 u8Mode_DC_DC_EN;
    UINT8 u8aPio_VSEL[3];
    UINT8 u8aMode_VSEL[3];
    UINT8 u8aVSELTruthTable[8];
    UINT8 u8aReserved2[2];
    UINT8 u8Pio_EN_SINK;
    UINT8 u8Mode_EN_SINK;
    UINT8 u8DAC_I_Direction;
    UINT8 u8Reserved3;
};
```

Description

This structure contains global configuration and status parameters that are either Integer Datatypes, Bit-Mapped bytes or another structure.

1. Members that are Integer data types:

Name	Size in Bytes	R/W Config time	R/W Run time	Description
u32aSourcePDO[7]	28	R/W	R	• Upto 7 fixed Source PDOs where Voltage is specified in mV and Current is specified in mA. • This array should be used only for Source Operation.
u32aSinkPDO[7]	28	R/W	R	• Upto 7 fixed Sink PDOs where Voltage is specified in mV and Current is specified in mA. • This array should be used only when the port is configured as Sink.
u32aNewPDO[7]	28	R/W	R/W	• Upto 7 fixed New PDOs where Voltage is specified in mV and Current in mA. • This array is common for Source and Sink. It is valid only when u8NewPDOSelect is set to 1.
u32aAdvertisedPDO[7]	28	R	R	• Upto 7 PDOs that are advertised to Port Partner. • During run time, this array holds the value of current u32aNewPDO[7] if u8NewPDOSelect is enabled else holds the value of current u32aSourcePDO[7]

u32aPartnerPDO[7]	28	R	R	• Upto 7 fixed Source PDOs where Voltage is specified in mV and Current is specified in mA • This array is common for Source and Sink.
u32RDO	4	R	R	• Complete raw RDO Data as Sent/Requested by connected port partner, Will be blank of no RDO has been received. • This variable is common for Source and Sink.
u16AllocatedPowerIn250mW	2	R	R	• Allocated Power for the Port PD contract in 0.25W steps

u16NegoVoltageIn50mV	2	R	R	• Negotiated Voltage from the Port Bits 19:10 from the RDO. Voltage is in 50mV steps. Possible values are,
				  1. 0x00 = No Contract
				  2. 0x064 = 5V
				  3. 0x0B4 = 9 V
				  4. 0x12C = 15V
				  5. 0x190= 20V
				  6. 0x3FF = 51.15V

u16NegoCurrentIn10mA	2	R	R	• Negotiated Current from the Port Bits 9: 0 from the RDO. Ampere is in 10mA steps. Sample values are,
				  1. 0x0000 = No Contract
				  2. 0x012C = 3A
				  3. 0x01F4 = 5A
				  4. 0x03FF = 10.24A

u16MaximumOperatingCurInmA	2	R/W	R	• Maximum allowable current or system's maximum operating current in terms of mA
u16aMinPDOPreferredCurInmA[7]	14	R/W	R	• Preferred minimum current range for the PDO by which the Sink may select without setting Capability Mismatch Bit with highest current preferred. • This array is applicable only for Sink Operation.
u16MinimumOperatingCurInmA	2	R/W	R	• Minimum current required by the sink hardware to be operational. • This variable is applicable only for Sink Operation. • When a Gotomin message is issued by source, sink reduces its operating current to the value provided in this variable.
u16DAC_I_MaxOutVoltInmV	2	R/W	R	• Defines the maximum voltage on DAC_I with a maximum of 2.5V in terms of mV • This is applicable only for Sink operation.
u16DAC_I_MinOutVoltInmV	2	R/W	R	• Defines the minimum voltage on DAC_I with a minimum of 0V in terms of mV • This is applicable only for Sink operation.

u16DAC_I_CurrentInd_MaxInA	2	R/W	R	• Defines which current in terms of mA corresponding to maximum output voltage
				• It can take either 3A or 5A value.
				• If it is 5A and maximum output voltage is 2.5V and if direction mentioned in u8DAC_I_Direction is High Amperage - Max Voltage, then
				  1. 0.5A > DAC_I = 0.25V
				  2. 1.5A > DAC_I = 0.75V
				  3. 2.0A > DAC_I = 1V
				  3. 2.0A > DAC_I = 1V
				  4. 3.0A > DAC_I = 1.5V
				  5. 4.0A > DAC_I = 2.0V
				  6. 5.0A > DAC_I = 2.5V
				• If it is 3A and maximum output voltage is 2.5V, then
				  1. 0.5A > DAC_I = 0.42V
				  2. 1.5A > DAC_I = 1.25V
				  3. 2.0A > DAC_I = 1.67V
				  4. 3.0A > DAC_I = 2.5V
				  5. 4.0A > DAC_I = 2.5V
				  6. 5.0A > DAC_I = 2.5V
				• This is applicable only for Sink operation.

u16PowerGoodTimerInms	2	R/W	R	• After an automatic fault recovery, u16PowerGoodTimerInms is run 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 fault 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.
u8SourcePDOCnt	1	R/W	R	• Number of Default Source PDOs supported. • This variable is applicable only when the port is configured as Source.
u8SinkPDOCnt	1	R/W	R	• Number of Default Sink PDOs supported. • This variable is applicable only when the port is configured as Sink.
u8NewPDOCnt	1	R/W	R/W	• Number of New PDOs Supported. • This variable is common for both Source and Sink. It is valid only when u8NewPDOSelect is set to 1.
u8AdvertisedPDOCnt	1	R	R	• Number of PDOs advertised to port partner.

u8PartnerPDOCnt	1	R	R	• Number of PDOs received from port partner. • This variable is common for Source and Sink.

u8NewPDOSelect	1	R/W	R/W	• 0 = Clear
				• 1 = Set, when Set the below steps are done by PSF.
				  1. Load New Source PDO
				  2. Advertise New PDOs to Port partner
				  3. Clear all New PDO registers
				  4. Clear this bit

u8SinkConfigSel	1	R/W	R	• BIT[1:0] - Sink Selection mode for operation.
				  1. '0x00' Mode A: Prefer Higher Voltage and Wattage
				  2. '0x01' Mode B: Prefer Lower Voltage and Wattage
				• BIT2 - No USB Suspend Flag
				  1. '1':Set the flag to '1' in RDO request
				  2. '0':Set the flag to '0' in RDO request
				• BIT3 - GiveBackFlag
				  1. '1':Set the flag to '1' in RDO request enabling GotoMin feature
				  2. Set the flag to '0' in RDO request disabling GotoMin Feature

u8FaultInDebounceInms	1	R/W	R	• 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.
u8OCSThresholdPercentage	1	R/W	R	• OCS Threshold. Reserved for future use. This variable is not currently used by PSF.
u8OVThresholdPercentage	1	R/W	R	• 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 u8OVThresholdPercentage should be greater than the desired range. • If 115% of the PDO voltage has to be considered as overvoltage for that PDO voltage, then define u8OVThresholdPercentage as 115. • It is mandatory to define u8OVThresholdPercentage when INCLUDE_POWER_FAULT_HANDLING is defined as '1'.

8

u8UVThresholdPercentage	1	R/W	R	• 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 u8UVThresholdPercentage should be less than the desired range. • If 85% of the PDO voltage has to be considered as under voltage for that PDO voltage, then define u8UVThresholdPercentage as 85. • u8UVThresholdPercentage must be defined when INCLUDE_POWER_FAULT_HANDLING is defined as '1'. As an exceptional case this factor is not considered for VSafe5V.
u8VCONNOCSDebounceInms	1	R/W	R	• Debounce timer value in terms of milliseconds for VCONN overcurrent fault conditions before reacting and entering fault recovery routine.
u8VBUSMaxFaultCnt	1	R/W	R	• The maximum number of back-to-back VBUS faults allowed before permanent shut down of the port. A back-to-back fault is a second fault which occurs within the u16PowerGoodTimerInms after a port is automatically reenabled from a previous fault condition. • During port shutdown due to over current fault, the device removes its CC termination and wait for port partner to get detached physically from the port to resume its normal operation.
u8VCONNMaxFaultCnt	1	R/W	R	• The maximum number of back-to-back VCONN faults allowed before it permanently disables the VCONN. A back-to-back fault is a second fault which occurs within the u16PowerGoodTimerInms after a port is automatically reenabled from a previous fault condition. • If VCONN is disabled due to over current VCONN power fault, VCONN will be enabled only after a physical detach and reattach.
u8Pio_EN_VBUS	1	R/W	R	• Defines the UPD350 PIO number used for EN_VBUS pin functionality for the Port. • This variable is applicable only for source operation. • 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 u8Mode_EN_VBUS configuration mode whenever stack requires VBUS to driven high as well as low. • The range of valid values is 0 to 15 which correspond to UPD350 PIO0 to PIO15. • To disable the pin functionality from the stack, the user can define a value of 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.

8

u8Mode_EN_VBUS	1	R/W	R	• Defines the PIO mode of the UPD350 PIO EN_VBUS defined in u8Pio_EN_VBUS. • This variable is applicable only for source operation. • It takes values only from enum eUPD_OUTPUT_PIN_MODES_TYPE.
u8Pio_FAULT_IN	1	R/W	R	• Defines the UPD PIO used as FAULT_IN pin. • FAULT_IN pin detects over current fault or under/over voltage fault from external sensing device based on its configuration u8Mode_FAULT_IN. • It can take values only from 0 to 15 and to disable the pin funtionality 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'.
u8Mode_FAULT_IN	1	R/W	R	• Defines the PIO mode of the UPD350 PIO FAULT_IN defined in u8Pio_FAULT_IN. • It takes only values from enum eFAULT_IN_MODE_TYPE.
u8Pio_VBUS_DIS	1	R/W	R	• Defines the UPD350 PIO for VBUS Discharge functionality. • It is a control for discharging VBUS (connecting VBUS to GND). It asserts as per u8mode_VBUS_DIS whenever VBUS voltage must transition from a high voltage to a lower voltage state and when VBUS is disabled. • The range of valid values are 0 to 15 which correspond to UPD350 PIO0 to PIO15. • To disable the pin functionality from the stack, user can define a value of 0cFF. • It is applicable only when CONFIG_DCDC_CTRL is defined as PWRCTRL_DEFAULT_PSF_GPIO_CONFIG
u8mode_VBUS_DIS	1	R/W	R	• Defines the PIO mode of the UPD350 PIO VBUS_DIS defined in u8Pio_VBUS_DIS. • It takes values only from enum eUPD_OUTPUT_PIN_MODES_TYPE
u8Pio_DC_DC_EN	1	R/W	R	• Defines the UPD350 PIO to enable DC-DC controller. • It is asserted as per u8Mode_DC_DC_EN during initialization and deasserted during error condition to reset the DC-DC controller. • It can take values from 0 to 15 and to disable the funtionality from stack, user can define it as 0xFF. • This variable is applicable only for source operation. • It is applicable only when CONFIG_DCDC_CTRL is defined as PWRCTRL_DEFAULT_PSF_GPIO_CONFIG
u8Mode_DC_DC_EN	1	R/W	R	• Defines the PIO mode of the UPD350 PIO DC_DC_EN defined in u8Pio_DC_DC_EN. • It takes values only from enum eUPD_OUTPUT_PIN_MODES_TYPE • This variable is applicable only for source operation.

8

u8aPio_VSEL[3]	3	R/W	R	• Defines the UPD350 PIO as voltage selector pins (VSEL[2:0]). • PSF Stack provides provision for three Voltage selector pins 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. • The range of valid values is 0 to 15 which correspond to UPD350 PIO0 to PIO15. • To disable the pin funtionality from the stack, user can define a value of 0xFF. Index 0 to 2 of this array correponds to VSEL0 to VSEL2. • This variable is applicable only for source operation. • It is applicable only when CONFIG_DCDC_CTRL is defined as PWRCTRL_DEFAULT_PSF_GPIO_CONFIG
u8aMode_VSEL[3]	3	R/W	R	• Defines the PIO mode of the UPD350 PIO VSEL pins VSEL0 to VSEL2. • It takes values only from enum eUPD_OUTPUT_PIN_MODES_TYPE. • This variable is applicable only for source operation. • It is applicable only when CONFIG_DCDC_CTRL is defined as PWRCTRL_DEFAULT_PSF_GPIO_CONFIG

u8aVSELTruthTable[8]	8	R/W	R	• Index 0 defines the assertion and deassertion to be driven on VSEL[2:0] pins(defined in u8aPio_VSEL[3]) by the PSF as per u8aMode_VSEL[3] to have an output voltage of VSafe0V out of DC/DC controller.
				• Index 1 to 7 defines the assertion and deassertion to be driven on VSEL[2:0] pins (defined in u8aPio_VSEL[3]) by the PSF stack as per u8aMode_VSEL[3] to have an output voltage of PDO voltage defined in u32aSourcePDO[7] out of DC/DC controller.
				• It is applicable only for Source.
				• For a 1 pin per voltage implementation, corresponding VSEL mapping would be,
				  1. '000' 5V (No pins asserted)
				  2. '001' 9V (VSEL0 asserted)
				  3. '010' 15V (VSEL1 asserted)
				  4. '100' 20V (VSEL2 asserted)

u8Pio_EN_SINK	1	R/W	R	• Defines the UPD350 PIO number used for EN_SINK pin. • This is applicable only for Sink operation. • This pin is asserted in the following conditions: 1. If the source supports Power delivery, the PD negotiated current should be greater than or equal to the current mentioned under u16MinimumOperatingCurInmA variable. 2. If the source does not support Power delivery and is Type-C only, the Rp value in source partner should be greater than or equal to the current mentioned under u16MinimumOperatingCurInmA variable. • This pin is de-asserted during a hard reset, a power fault recovery or a detach. • The range of valid values is 0 to 15 which correspond to UPD350 PIO0 to PIO15. • 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.
u8Mode_EN_SINK	1	R/W	R	• Defines the PIO mode for EN_SINK pin • This is applicable only for Sink operation. • It takes values only from enum eUPD_OUTPUT_PIN_MODES_TYPE.
u8DAC_I_Direction	1	R/W	R	• Specifies the direction of DAC_I to allow user invert direction of DAC_I if required 1. 0 - High Amperage - Max Voltage 2. 1- High Amperage - Min Voltage • This is applicable only for Sink operation.
u16Reserved1	2			Reserved
u8aReserved1[2]	2			Reserved
u8aReserved2[2]	2			Reserved
u8Reserved3	1			Reserved

2. Members that are Bit-Mapped bytes:

a. u32CfgData:

u32CfgData variable holds the Port Configuration Data. It's size is 4 bytes.

Bit	R/W Config time	R/W Run time	Description
0	RW	R	Port Enable/Disable • '0' Disabled • '1' Enabled

3:1	RW	R	Port Power Role • '000' Sink • '001' Source
5:4	RW	R	Rp Selection • '00' Disabled • '01' USB Power • '10' 1.5A • '11' 3.0A
8:6	RW	R	USB Data • '000' No Data • '001' USB2 • '010' USB3.1 Gen1 • '011' USB3.1 Gen2
9	RW	R	VCONN OCS Enable • '0' Disable • '1' Enable
32:10			Reserved

b. u32PortConnectStatus: u32PortConnectStatus variable holds the connection status of the port. It's size is 4 bytes.

Bit	R/W Config time	R/W Run time	Description
0	R	R	Attached • '0' Detached • '1' Attached
1	R	R	Orientation • '0' Unflipped • '1' Flipped
2	R	R	Data Role • '0' UFP • '1' DFP
3	R	R	Power Role • '0' Sink • '1' Source
4	R	R	VCONN Status • '0' Disabled • '1' Enabled
5	R	R	Cable Reduced Source Capabilities • '0' Attached USB-C cable supports the locally-defined Source PDOs • '1' Attached USB-C cable does not support the locally defined Source PDOs

6	R	R	PD Bal Reduced Source Capabilities • '0' The advertised PDOs are equivalent to the default configured values • '1' The advertised PDOs have been reduced from default configured values
7	R	R	Source Capability Mismatch • '0' De-asserted by Source port when there is capability mismatch with sink partner • '1' Asserted by Source port when sink port indicates capability mismatch in RDO
8	R	R	As Source PD Contract Good • '0' As Source: USB-C Connection Only (No Request Made Yet) • '1' As Source; USB PD connection established, Power request has been made, accepted and PS_RDY message sent. • This bit will always remain 0 when acting as sink.
9	R	R	As Source RDO Accepted • '0' As Source: No RDO Accept message has been sent to last Request made by attached Sink or no Request has yet been made during connection. • '1' As Source: RDO Accept message has been sent to last Request made by attached Sink • This bit will always remain 0 when acting as sink
10	R	R	As Source RDO Rejected • '0' As source; No RDO reject message has been sent to last request made by attached Sink or no Request has yet been made during connection • '1' As Source: RDO Reject message has been sent to last Request made by attached Sink • This bit will always remain 0 when acting as Sink
11	R	R	As Sink Last Request Accept • '0' As Sink: Last RDO Request was not Accepted or no request has yet been made. • '1' As Sink: Last RDO Request was Accepted • This bit will always remain 0 when acting as a source.
12	R	R	As Sink Last Request Reject • '0' As Sink: Last RDO Request was not Rejected or no request has yet been made. • '1' As Sink: Last RDO Request was Rejected • This bit will always remain 0 when acting as a source.
13	R	R	As Sink Last Request PS_RDY • '0' As Sink: PS_RDY not yet received for last RDO request • '1' As Sink: PS_RDY received for last RDO request • This bit will always remain 0 when acting as a source.
14	R	R	Sink Capability Mismatch • '0' De-asserted by the Sink Port when there is no capability mismatch • '1' Asserted by Sink Port when no Source capability was found
16:15	R	R	Rp Value detected by Sink • '00' Disabled • '01' USB Power • '10' 1.5A • '11' 3.0A

31:17			Reserved

c. u16PortIOStatus: u16PortIOStatus variable holds the IO status of the port. It's size is 2 bytes.

Bit	R/W Config time	R/W Run time	Description
0	R	R	EN_DC_DC Status • '1' Asserted • '0' De-asserted
1	R	R	VSEL0 Status • '1' Asserted • '0' De-asserted
2	R	R	VSEL1 Status • '1' Asserted • '0' De-asserted
3	R	R	VSEL2 Status • '1' Asserted • '0' De-asserted
4	R	R	EN_VBUS Status • '1' Asserted • '0' De-asserted
5	R	R	VBUS_DIS Status • '1' Asserted • '0' De-asserted
6	R	R	EN_SINK Status • '1' Asserted • '0' De-asserted
7	R	R	1.5_IND Status • '1' Asserted • '0' De-asserted
8	R	R	3.0_IND Status • '1' Asserted • '0' De-asserted
9	R	R	PS_RDY Received • '1' Asserted • '0' De-asserted
10	R	R	Capability Mismatch • '1' Asserted • '0' De-asserted
15:11			Reserved

8

d. u16PortStatusChange: u16PortStatusChange variable defines the port connection status change bits. It's size is 2 bytes.

Bit	R/W Config time	R/W Run time	Description
0	R	R	Attach Event • '0' Since the last read of this register, PSF has not experienced a USB-C attach • '1' Since the last read of this register, PSF has experienced a USB-C attach
1	R	R	Detach Event • '0' Since the last read of this register, PSF has not experienced a USB-C detach • '1' Since the last read of this register, PSF has experienced a USB-C detach
2	R	R	As Source New Request • '0' As Source, since the last read of this register, PSF has not received any new PDO request from attached port partner • '1' As Source: Received a new PDO request the attached Sink port partner, As Sink: Received an updated set of Source capabilities form the attached Source port partner
3	R	R	As Sink New PDOs Received • '0' As Sink: Since the last read of this register, PSF has not received any changed source capabilities • '1' As Sink: Received an updated set of Source capabilities form the attached Source port partner
4	R	R	As Sink New Request Sent • '0' As Sink: Since the last read of this register, PSF has not sent any additional Sink RDOs • '1' As Sink: Since the last read of this register, PSF has issued a new Sink RDO to the attached Source, • This bit will always remain 0 when acting as a source
5	R	R	As Sink Last Request Accept • '0' As Sink: Since the last read of this register, PSF has not received any new Request Accept messages from the attached Source • '1' As Sink: Since the last read of this register, PSF has received a new Request Accept from the attached Source
6	R	R	As Sink Last Request Reject • '0' As Sink: Since the last read of this register, PSF has not received any new Request Reject messages from the attached Source • '1' As Sink: Since the last read of this register, PSF has received a new Request Reject message from the attached Source
7	R	R	As Sink Last Sink PS Rdy • '0' As Sink: Since the last read of this register, PSF has not received any PS_RDY messages from the attached Source • '1' As Sink: Since the last read of this register, PSF has received a PS_RDY message from the attached Source
8	R	R	Hard Reset Event • '0' Since the last read of this register, PSF has not experienced a USB PD Hard Reset • '1' Since the last read of this register, PSF has experienced a USB PD Hard Reset

9	R	R	Pin Reset Event • '0' Since the last read of this register, PSF has not been reset via POR or pin • '1' Since the last read of this register, PSF has been reset via POR or pin
10	R	R	VBUS Fault • '0' Since the last read of this register, no VBUS faults have been detected • '1' Since the last read of this register, 1 or more VBUS faults have been detected
11	R	R	VCONN Fault • '0' Since the last read of this register, no VCONN faults have been detected • '1' Since the last read of this register, 1 or more VCONN faults have been detected
15:12			Reserved

e. u16PortIntrMask: u16PortIntrMask variable defines the port interrupt mask bits. It's size is 2 bytes.

Bit	R/W Config time	R/W Run time	Description
0	RC	RC	Attach Event Mask • '0' Do not mask interrupt pin toggle on changes to this event. • '1' Mask this event from generating interrupt pin toggle.
1	RC	RC	Detach Event Mask • '0' Do not mask interrupt pin toggle on changes to this event. • '1' Mask this event from generating interrupt pin toggle.
2	RC	RC	As Source New Request Mask • '0' Do not mask interrupt pin toggle on changes to this event. • '1' Mask this event from generating interrupt pin toggle.
3	RC	RC	As Sink New PDOs Received Mask • '0' Do not mask interrupt pin toggle on changes to this event. • '1' Mask this event from generating interrupt pin toggle.
4	RC	RC	As Sink New Request Sent Mask • '0' Do not mask interrupt pin toggle on changes to this event. • '1' Mask this event from generating interrupt pin toggle.
5	RC	RC	As Sink Last Request Accept Mask • '0' Do not mask interrupt pin toggle on changes to this event. • '1' Mask this event from generating interrupt pin toggle.
6	RC	RC	As Sink Last Request Reject Mask • '0' Do not mask interrupt pin toggle on changes to this event. • '1' Mask this event from generating interrupt pin toggle.
7	RC	RC	As Sink Last Sink PS RDY Mask • '0' Do not mask interrupt pin toggle on changes to this event. • '1' Mask this event from generating interrupt pin toggle.

8

8	RC	RC	Hard Reset Event Mask
			• '0' Do not mask interrupt pin toggle on changes to this event.
			• '1' Mask this event from generating interrupt pin toggle.
9	RC	RC	Pin Reset Mask
			• '0' Do not mask interrupt pin toggle on changes to this event.
			• '1' Mask this event from generating interrupt pin toggle.
10	RC	RC	VBUS Fault Mask
			• '0' Do not mask interrupt pin toggle on changes to this event.
			• '1' Mask this event from generating interrupt pin toggle.
11	RC	RC	VCONN Fault Mask
			• '0' Do not mask interrupt pin toggle on changes to this event.
			• '1' Mask this event from generating interrupt pin toggle.
15:12			Reserved

Remarks

None

8.3.3 _PBPortCfgStatus Structure

C

```
struct _PBPortCfgStatus {
  UINT16 u16MaxPrtPwrBankA;
  UINT16 u16MaxPrtPwrBankB;
  UINT16 u16MaxPrtPwrBankC;
  UINT16 u16MaxPrtCurrent;
  UINT8 u8PBEnablePriority;
  UINT8 u8aReserved4[3];
};
```

Description

This structure contains global configuration and status parameters that are either Integer Datatypes, Bit-Mapped bytes or another structure. This structure is used only when either of the macros INCLUDE_POWER_BALANCING or INCLUDE_POWER_THROTTLING is set to '1'.

1. Members that are Integer data types:

Name	Size in Bytes	R/W Config time	R/W Run time	Description
u16MaxPrtPwrBankA	2	R/W	R	• Maximum Port Power Bank A in 0.25W steps.
				• Unit/LSB = 0.25W
				• Sample values this variable can take are,
				1. 0x0001 = 0.25W
				2. 0x00F0 = 60W
				3. 0x0190 = 100W
				• Note : A setting of 0x0000 and 0x191-0xFFF is invalid.

u16MaxPrtPwrBankB	2	R/W	R	- Maximum Port Power Bank B in 0.25W steps. - Unit/LSB = 0.25W - Sample values this variable can take are, 0x0001 = 0.25W 0x00F0 = 60W 0x0190 = 100W - Note : A setting of 0x0000 and 0x191-0xFFF is invalid.
u16MaxPrtPwrBankC	2	R/W	R	- Maximum Port Power Bank A in 0.25W steps. - Unit/LSB = 0.25W - Sample values this variable can take are, 0x0001 = 0.25W 0x00F0 = 60W 0x0190 = 100W - Note : A setting of 0x0000 and 0x191-0xFFF is invalid.
u16MaxPrtCurrent	2	R/W	R	- Maximum allowable current for ports in 10mA steps - Sample values this variable can take are, 0x0032 = 0.5A 0x012C = 3A 0x01F4 = 5A - Note : Values above 5A (0x01F5 - 0x0FFF) are not supported
u8aReserved4[3]	3			Reserved

2. Members that are Bit-Mapped bytes:

u8PBEnablePriority:

u8PBEnablePriority defines PB Enable/Disable per port and also the priority for the port. It's size is 1 byte.

Bit	R/W Config time	R/W Run time	Description
0	R/W	R/W	PB Enable/Disable '0' Disable '1' Enable
3:1	R/W	R/W	Selects the port Priority 000b is the highest priority
7:4			Reserved

Remarks

None

8.3.4 _PPSPortCfgStatus Structure

C

```
struct _PPSPortCfgStatus {
  UINT8 u8PPSEnable;
  UINT8 u8aReserved5[3];
  UINT32 u32aPPSApdo[3];
};
```

Description

This structure contains the following PPS configuration parameters. This structure is used only when INCLUDE_PD_SOURCE_PPS is set to '1'.

Name	Size in Bytes	R/W Config time	R/W Run time	Description
u8PPSEnable	1	R/W	R	PPS Enable/Disable. • '0' = Disable • '1' = Enable
u32aPPSApdo[3]	12	R/W	R	Defines the PPS APDOs.
u8aReserved5[3]	3			Reserved

Remarks

None

8.3.5 gasCfgStatusData Variable

C

```
GLOBAL_CFG_STATUS_DATA gasCfgStatusData;
```

Description

gasCfgStatusData is the global structure which defines the overall system level configuration and status parameters of PSF. This structure contains the system level and port specific Configuration and Status parameters of PSF including Type C, PD, PB, PT and PPS parameters.

It is mandatory that the user has to initialize the configuration parameters for the PSF stack to funtion properly. This can be done through MCHP_PSF_HOOK_BOOT_TIME_CONFIG which initializes the parameters defined in gasCfgStatusData during compile time. For accessing the configuration registers and reading the status registers at run time, an I2C slave interface shall be used by the user application.

Remarks

None

8

8.4 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.

8.4.1 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 CONFIG_DCDC_CTRL is defined as I2C_DC_DC_CONTROL_CONFIG, default I2C based DC-DC Controller is used. If left undefined, default stack's DC-DC control option is not used and the user must control power 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     I2C_DC_DC_CONTROL_CONFIG (Uses default I2C based DC-DC contol)
#define CONFIG_DCDC_CTRL     (If undefined, Default DC DC control provided by stack is not
used)
```

Macros

	Name	Description
⊸	PWRCTRL_DEFAULT_PSF_GPIO_CONFIG	Macro to indicate GPIO based DC-DC Controller.
⊸	I2C_DC_DC_CONTROL_CONFIG	Macro to indicate I2C based DC-DC Controller.

8.4.1.1 PWRCTRL_DEFAULT_PSF_GPIO_CONFIG

C

```
#define PWRCTRL_DEFAULT_PSF_GPIO_CONFIG 1
```

Description

PWRCTRL_DEFAULT_PSF_GPIO_CONFIG defines the default GPIO based DC-DC Controller used by PSF.

Remarks

None.

8

8.4.1.2 I2C_DC_DC_CONTROL_CONFIG

C

```
#define I2C_DC_DC_CONTROL_CONFIG 2
```

Description

I2C_DC_DC_CONTROL_CONFIG defines the default GPIO based DC-DC Controller used by PSF.

Remarks

None.

8.4.2 Power Control GPIO Enumeration constants

Enumerations

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

Description

This section describes the enumeration constants provided by PSF for configuring the various Port Power Control parameters such as Mode and PIO number of EN_VBUS, DC_DC_EN pins.

8.4.2.1 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

8.4.2.2 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

8.5 MCU Idle Timeout Configuration

Macros

	Name	Description
⊸	CONFIG_PORT_UPD_IDLE_TIMEOUT_MS	UPD350 Idle Timeout value in milliseconds.

8.5.1 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)
```

8.6 Debug Hook Configuration

Macros

	Name	Description
⊸	CONFIG_HOOK_DEBUG_MSG	Print status messages from PSF stack through UART interface

8.6.1 CONFIG_HOOK_DEBUG_MSG

C

```
#define CONFIG_HOOK_DEBUG_MSG 0
```

Description

Setting CONFIG_HOOK_DEBUG_MSG to 1, prints status messages from PSF stack through UART interface.

Remarks

The following hook APIs should be defined with appropriate UART functions to view status messages from PSF stack.

1. MCHP_PSF_HOOK_DEBUG_INIT()

2. MCHP_PSF_HOOK_PRINT_CHAR(byData)

3. MCHP_PSF_HOOK_PRINT_INTEGER(dwWriteInt, byWidth)

4. MCHP_PSF_HOOK_PRINT_TRACE(pbyMessage)

None.

Example

```
#define CONFIG_HOOK_DEBUG_MSG    0
#define CONFIG_HOOK_DEBUG_MSG    1
```

8.7 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.
⊸	CONFIG_HWMAJOR_VERSION	Hardware Major Version.
⊸	CONFIG_HWMINOR_VERSION	Hardware Minor Version.

Description

This section explains the configurable options available for PDFU.

8.7.1 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
```

8.7.2 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
```

8.7.3 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(34KB).

Example

```
#define CONFIG_MAX_FIRMWARE_IMAGESIZE 0x9800UL (38*1024 bytes for 38KB Updatable
application).
```

8.7.4 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 '1'. 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
```

8.7.5 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 '1'. 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
```

8.7.6 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)
```

8.7.7 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 '1'. It can have values from 0x00 to 0xFF. By default, it is defined as '0x03'.

Example

```
#define CONFIG_VALIDATION_PHASE_WAITTIME   0x03u
```

8.7.8 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
```

8.7.9 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

```
#define CONFIG_HWMINOR_VERSION    0x0
```

9 System level integration of PSF

As PSF is hardware agnostic, it can be integrated to many Microchip SoC platforms. 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 supports the following versions of UPD350 Rev A Silicon.

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

PSF requires the 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.
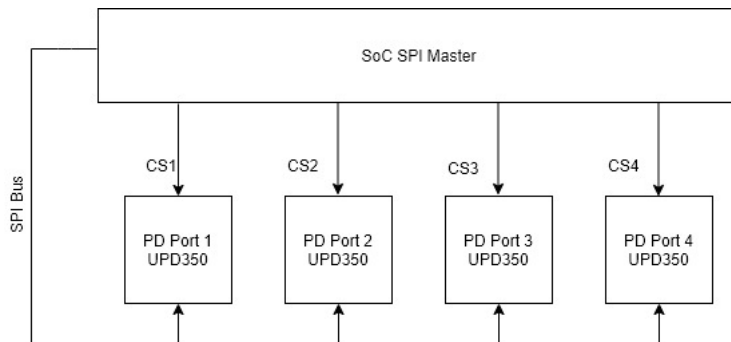
I2C Master:

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.
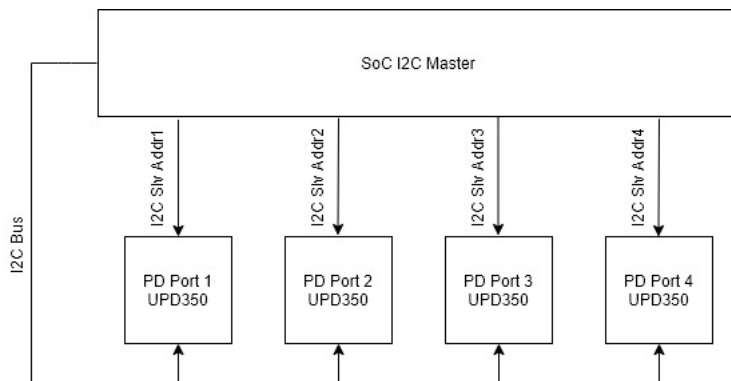
9.1.1.2 PIOs for UPD350 IRQs

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.
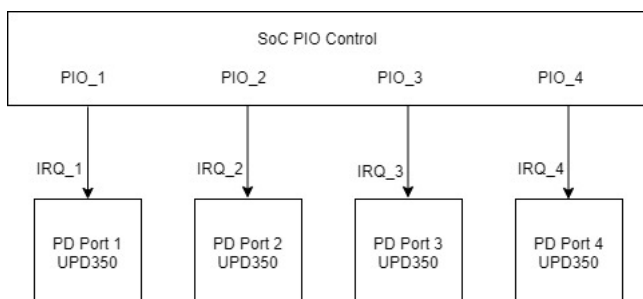
9.1.1.3 PIO for UPD350 Reset

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.
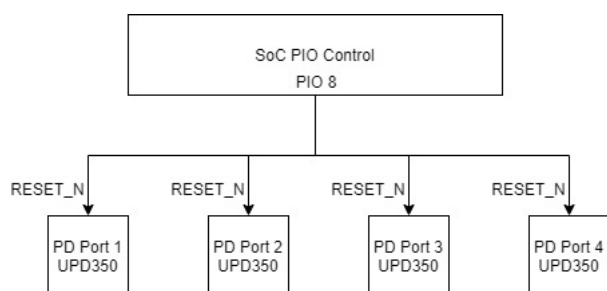
9.1.2 Hardware Timer

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,
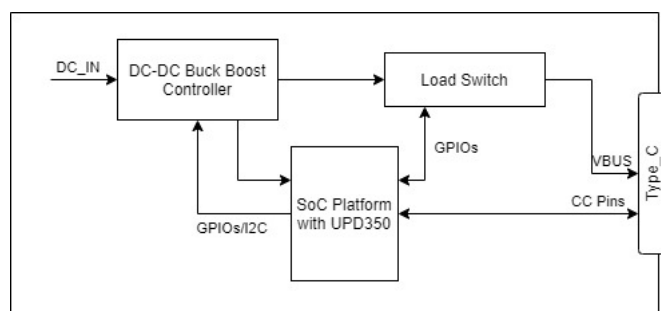
For Sink-Only port, the set up would be as follows,
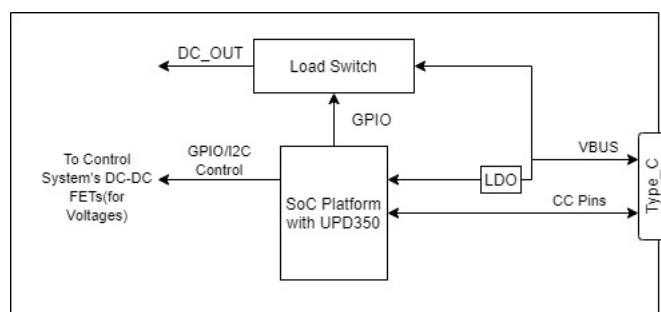
Note: The PSF evaluation and development platform includes a Microchip MCP19119 Buck converter. This Buck converter is digitally enhanced and capable of supporting a number of control schemes through firmware modification. By default, it is configured for a basic GPIO based control scheme.

9.2 Software Requirements

This section lists the software requirements for PSF integration. This guide helps to determine 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

1. 2-Port Source only solution is 21.26KB
2. 2-Port Source with Power Balancing enabled is 23.77KB
3. 1-Port Sink Solution is 19KB.

Estimated Data RAM size of PSF for

1. 2-Port Source only solution is 1580 bytes
2. 2-Port Source with Power Balancing enabled is 1626 bytes .
3. 1-Port Sink is 936 bytes.
4. Any additional port will increase total Data RAM size requirement by approximately 500 Bytes.

- **16-bit**

TBD

- **8-bit**

TBD

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

Only Little-Endian supported SoC is tested with PSF. Big-Endian SoC is yet to be tested.

9

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. This APIs will also be available in PSF_APIHooks header file.

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 PSF Configuration Options.

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

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_READ	Initiates a read transfer to UPD350 via I2C/SPI
⊸	MCHP_PSF_HOOK_UPD_WRITE	Initiates a write transfer to UPD350 via I2C/SPI
⊸	MCHP_PSF_HOOK_UPD_COMM_ENABLE	Enable or disables hardware interface(SPI/I2C) communication to the port's UPD350

10.2.1.1 MCHP_PSF_HOOK_UPDHW_INTF_INIT

C

```
#define MCHP_PSF_HOOK_UPDHW_INTF_INIT 0
```

Description

PSF requires 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_READ

C

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

10

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
        }
```

10.2.1.3 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
    }
```

10.2.1.4 MCHP_PSF_HOOK_UPD_COMM_ENABLE

C

```
#define MCHP_PSF_HOOK_UPD_COMM_ENABLE(u8PortNum,u8Enable)
```

10

Description

This hook is called before and after the read/write operations of UPD350 registers with port specifically enable/disable hardware interface(I2C/SPI) communication to the port's UPD350. 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 UPD350. 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 UPD350. 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 (CONFIG_PD_PORT_COUNT-1).
u8Enable	Parameter indicating whether to enable or disable the communication for the port.
When u8Enable is 1	Enable interface, 0 - Disable interface.

Returns

None.

Remarks

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

Example

```c
#define MCHP_PSF_HOOK_UPD_COMM_ENABLE (u8PortNum,u8Enable)
                hw_spi_drive_cs (u8PortNum, u8Enable)
void hw_spi_drive_cs (UINT8 u8PortNum,UINT8 u8Enable);
void hw_spi_drive_cs (UINT8 u8PortNum,UINT8 u8Enable)
{
    if (u8Enable == TRUE)
    {
        //Set pin level low for SOC's GPIO that is connected to the u8PortNum port's
        //UPD350 SPI CS pin
    }
    else if(u8Enable == 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_ENABLE(u8PortNum,u8Enable)
                hw_port_i2cmux_routing(u8PortNum,u8Enable)
void hw_port_i2cmux_routing(u8PortNum,u8Enable);
void hw_port_i2cmux_routing(u8PortNum,u8Enable)
{
    if (u8Enable == TRUE)
    {
        //Route the I2C mux to the u8PortNum UPD350 Port
    }
    else if(u8Enable == FALSE)
    {
        //disable the I2C mux routing to the u8PortNum UPD350 Port
    }
}
```

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.
⊸	MCHP_PSF_CONFIG_16BIT_PDTIMER_COUNTER	Config 16-bit PD Timer Counter

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
```

10

```
        //interrupt frequency & register MchpPSF_PDTimerHandler as callback
        //Return TRUE if Timer initialisation is successful else return FALSE
    }
```

10.2.2.3 MCHP_PSF_CONFIG_16BIT_PDTIMER_COUNTER

C

```
#define MCHP_PSF_CONFIG_16BIT_PDTIMER_COUNTER 0
```

Description

MCHP_PSF_CONFIG_16BIT_PDTIMER_COUNTER can be defined as either 1 or 0 to set the timeout counter in PSF to unsigned 16bit or unsigned 32bit correspondingly. When set as 1, maximum timeout that can be set will be 65535 ticks.(Ticks = Resolution of the Hardware timer used). When set as 0 , maximum timeout that can be set will be 4294967296 ticks. Default value of MCHP_PSF_CONFIG_16BIT_PDTIMER_COUNTER is set as 1. With Hardware timer resolution set as 1ms , PSF will be capable of handling timeouts upto 65.535 Seconds.

Remarks

None

Example

```
    #define MCHP_PSF_CONFIG_16BIT_PDTIMER_COUNTER 1 (Sets timeout variable inside the PSF as
unsigned 16bit)
    #define MCHP_PSF_CONFIG_16BIT_PDTIMER_COUNTER 0 (Sets timeout variable inside the PSF as
unsigned 32bit)
```

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 a 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).

10

Returns

None.

Remarks

User definition of this Hook function is mandatory

Example

```c
#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
    }
}
```

10.2.4 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

```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 define a 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.

10

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)
```

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

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 when PSF 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 may be delayed and cause timing issues/conflicts. 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 enters into a critical section. It must provide an implementation to disable the interrupts globally. This hook implementation must be very short, otherwise resposne time may be delayed and cause timing issues/conflicts. Define relevant function that has no arguments without return type.

Preconditions

None.

Returns

None.

Remarks

User definition of this Hook function is mandatory

10

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

Description

Memory compare and copy functionalities are given as hooks for user implementation because some compilers have inbuilt library functions that does this operation in minimum machine cycle. This hooks provide options to use those functions for better performance.

10.2.6.1 MCHP_PSF_HOOK_MEMCMP

C

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

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
```

10

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.
```

10.2.7 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. Additionally, in case of sink functionality, this hook may be defined with APIs to initialize a DAC.
⚬	MCHP_PSF_HOOK_PORTPWR_DRIVE_VBUS	Drives the VBUS for given voltage, current
⚬	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.
⚬	MCHP_PSF_HOOK_PORTPWR_ENDIS_DCDCEN	Enables or disables DC_DC_EN functionality for a given port.
⚬	MCHP_PSF_HOOK_DRIVE_DAC_I	Indicates the implicit/explicit current capability of attached source partner.

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, load switch or DAC 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. A DAC may initialized under this hook if PSF is configured as SINK.

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.

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 discharge of VBUS when VBUS transitions from higher to lower voltage. PSF provides default DC-DC Buck booster control configuration via CONFIG_DCDC_CTRL define. If user chose to

10

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 may be delayed and cause timing issues/conflicts. 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 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.

10

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_Configure_SinkCircuitary(u8PortNum, u16Voltage, u16Current)
void hw_Configure_SinkCircuitary(UINT8 u8PortNum,UINT16 u16Votlage,UINT16 u16Current);
void hw_Configure_SinkCircuitary(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
}
```

10.2.8.5 MCHP_PSF_HOOK_PORTPWR_ENDIS_DCDCEN

C

```
#define MCHP_PSF_HOOK_PORTPWR_ENDIS_DCDCEN(u8PortNum, u8EnaDisDCDCEn)
```

Description

DC_DC_EN functionality is required to enable DC-DC Controller. It asserts high once PSF is initialized, ready to operate and CC pins are functional. It will be toggled low during error condition say, on occurrence of fault to reset the DC-DC. 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 DC_DC_EN 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

None.

Parameters

Parameters	Description
u8PortNum	Port number of the device. It takes value between 0 to (CONFIG_PD_PORT_COUNT-1).
u8EnaDisDCDCEn	Flag indicating whether to enable/disable DC_DC_EN.

Returns

None.

Remarks

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

Example

```
#define MCHP_PSF_HOOK_PORTPWR_ENDIS_DCDCEN(u8PortNum, u8EnaDisDCDCEn)
                hw_portpower_enab_dis_DCDCEn(u8PortNum, u8EnableDisable)
void hw_portpower_enab_dis_DCDCEn(UINT8 u8PortNum,UINT8 u8EnableDisable);
void hw_portpower_enab_dis_DCDCEn(UINT8 u8PortNum,UINT8 u8EnableDisable)
{
    if (TRUE == u8EnableDisable)
    {
```

10

```
            //Enable the DC_DC_EN for "u8PortNum" Port
    }
    else
    {
            //Disable the DC_DC_EN for "u8PortNum" Port
    }
}
```

10.2.8.6 MCHP_PSF_HOOK_DRIVE_DAC_I

C

```
#define MCHP_PSF_HOOK_DRIVE_DAC_I(u16DACData)
```

Description

This hook is called to indicate the sink hardware of the implicit/explicit current capability of attached source partner. The current capability is indicated thorough a voltage level on Digital to Analog Converter(DAC)'s output pin. The voltage level on DAC's output pin is calculated based on per port Configuration parameters, which were configured using MCHP_PSF_HOOK_BOOT_TIME_CONFIG(gasCfgStatusData) hook.

In gasCfgStatusData structure, if u16DAC_I_CurrentInd_MaxInA is 5000mA, u16DAC_I_MaxOutVoltInmV is 2500mV, u16DAC_I_MinOutVoltInmV is 0V and direction mentioned in u8DAC_I_Direction is High Amperage - Max Voltage, then

1. 0.5A > DAC_I = 0.25V

2. 1.5A > DAC_I = 0.75V

3. 2.0A > DAC_I = 1V

4. 3.0A > DAC_I = 1.5V

5. 4.0A > DAC_I = 2.0V

6. 5.0A > DAC_I = 2.5V

In gasCfgStatusData structure, if u16DAC_I_CurrentInd_MaxInA is 3000mA, u16DAC_I_MaxOutVoltInmV is 2500mV, u16DAC_I_MinOutVoltInmV is 0V and direction mentioned in u8DAC_I_Direction is High Amperage - Max Voltage, then * If it is 3A and maximum

1. 0.5A > DAC_I = 0.42V

2. 1.5A > DAC_I = 1.25V

3. 2.0A > DAC_I = 1.67V

4. 3.0A > DAC_I = 2.5V

5. 4.0A > DAC_I = 2.5V

6. 5.0A > DAC_I = 2.5V

This is applicable only for Sink operation.

A suitable function that initializes DAC from SoC may be implemented in this hook.

Preconditions

SoC should support a DAC. And the DAC should be initialized under MCHP_PSF_HOOK_HW_PORTPWR_INIT() hook.

Returns

None.

Remarks

This hook is applicable only if INCLUDE_PD_SINK macro is 1. Definition of this hook is not mandatory.

Example

```
#define MCHP_PSF_HOOK_DRIVE_DAC_I(u16DACData)    SAMD20_Drive_DAC_I(u16DACData)
void SAMD20_Drive_DAC_I(UINT16 u16DACData);
void SAMD20_Drive_DAC_I(UINT16 u16DACData)
{
```

10

```
        //Implement user specific application to output volatge provided under
        //u16DACData argument in DAC's output pin
    }
```

10.2.9 Boot time Configuration

Macros

	Name	Description
⊸○	MCHP_PSF_HOOK_BOOT_TIME_CONFIG	Updates the global and per port Configuration parameters.

10.2.9.1 MCHP_PSF_HOOK_BOOT_TIME_CONFIG

C

```
#define MCHP_PSF_HOOK_BOOT_TIME_CONFIG(gasCfgStatusData)
```

Description

This function is called to update the configuration parameters of Type-C, PD, Power Balancing, Power throttling and PPS contained in gasCfgStatusData structure. This API must have an input parameter of gasCfgStatusData.

Preconditions

None.

Parameters

Parameters	Description
gasCfgStatusData	Holds the structure pointer of the structure _GlobalCfgStatusData

Returns

None.

Remarks

User definition of this Hook function is mandatory

Example

```
    #define MCHP_PSF_HOOK_BOOT_TIME_CONFIG(gasCfgStatusData)
PSF_LoadConfig(gasCfgStatusData)
    void PSF_LoadConfig(GLOBAL_CFG_STATUS_DATA gasCfgStatusData);
    void PSF_LoadConfig(gasCfgStatusData)
    {
    // Configure the global parameters
    // Select Throttling Bank B
    gasCfgStatusData.u8PwrThrottleCfg = 0x01;
    // Set 120W as Total system Power of Bank A
    gasCfgStatusData.u16SystemPowerBankA = 0x01E0U;
    // Configure per port parameters
    // Set Port 1's VBUS Maximum Fault Count as 3
    gasCfgStatusData.sPerPortData[0].u8VBUSMaxFaultCnt = 0x03; // 0 is the port number
    // Configure PIO6 as the DC_DC_EN pin for Port 1
    gasCfgStatusData.sPerPortData[0].u8Pio_DC_DC_EN = 0x06; // 0 is the port number
    // Configure per port PB parameters
    // Set Port 2's maximum port power for Bank C as 60W
    gasCfgStatusData.sPBPerPortData[1].u16MaxPrtPwrBankC = 0x00F0U; // 1 is the port number
    }
```

10

10.2.10 Hooks for Policy Manager

Macros

	Name	Description
⊸	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
    }
```

10.2.11 Debug Hook Functions

Macros

	Name	Description
⊸	MCHP_PSF_HOOK_DEBUG_INIT	Initialization of debug module interface
⊸	MCHP_PSF_HOOK_PRINT_CHAR	Outputs a character via UART interface
⊸	MCHP_PSF_HOOK_PRINT_INTEGER	Outputs an integer via UART interface
⊸	MCHP_PSF_HOOK_PRINT_TRACE	Outputs a string via UART interface

10

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_PRINT_CHAR

C

```
#define MCHP_PSF_HOOK_PRINT_CHAR(byData)
```

Description

This hook is can be called to output a character via UART interface to help the user in debugging. byData is of type char. Define relevant function to print a character via UART terminal with argument of type char and no return type.

Preconditions

MCHP_PSF_HOOK_DEBUG_INIT() should have been called before using this API.

Returns

None.

Remarks

This hook API can be used only if CONFIG_HOOK_DEBUG_MSG is 1.

Example

```
#define MCHP_PSF_HOOK_PRINT_CHAR(byData)          uart_print_char(byData)
void uart_print_char(char);
void uart_print_char(char byData)
{
    //Print a character through UART terminal
}
```

10.2.11.3 MCHP_PSF_HOOK_PRINT_INTEGER

C

```
#define MCHP_PSF_HOOK_PRINT_INTEGER(dwData, byWidth)
```

Description

This hook is can be called to output an integer via UART interface to help the user in debugging. The size of integer is specified in byWidth. dwWriteInt is of type unsigned long. byWidth is of type unsigned char. Define relevant function to print an integer via UART terminal with arguments of type unsigned long and unsigned char and no return type.

Preconditions

MCHP_PSF_HOOK_DEBUG_INIT() should have been called before using this API.

Returns

None.

Remarks

This hook API can be used only if CONFIG_HOOK_DEBUG_MSG is 1.

Example

```
#define MCHP_PSF_HOOK_PRINT_INTEGER(dwWriteInt, byWidth)
uart_print_char(dwWriteInt, byWidth)
    void uart_print_int(UINT32 dwWriteInt, UINT8 byWidth);
    void uart_print_int(UINT32 dwWriteInt, UINT8 byWidth)
    {
        //Print byWidth no of bytes from dwWriteInt through UART terminal.
    }
```

10.2.11.4 MCHP_PSF_HOOK_PRINT_TRACE

C

```
#define MCHP_PSF_HOOK_PRINT_TRACE(pbyMessage)
```

Description

This hook is can be called to output a string via UART interface to help the user in debugging. pbyMessage is of type char *. Define relevant function to print a string via UART terminal with argument of type char* and no return type.

Preconditions

MCHP_PSF_HOOK_DEBUG_INIT() should have been called before using this API.

Returns

None.

Remarks

This hook API can be used only if CONFIG_HOOK_DEBUG_MSG is 1.

Example

```
#define MCHP_PSF_HOOK_PRINT_TRACE(pbyMessage)          uart_print_string(pbyMessage)
    void uart_print_string(char *);
    void uart_print_string(char * pbyMessage)
    {
        //Print a string in pbyMessage through UART terminal
    }
```

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

```
#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

```
#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.

10

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 <u>INCLUDE_PDFU</u> is TRUE.

Example

```
#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

```
#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 <u>INCLUDE_PDFU</u> is TRUE.

Example 1

1. 0x01 - Corresponds to Bootloader Application

2. 0x02 - Corresponds to Fixed Application

3. 0x03 - Corresponds to Updatable Application

Example 2

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

10.2.12.4 MCHP_PSF_HOOK_IS_PDFU_ALLOWED_NOW

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

10

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

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

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.
}
```

10.2.13 MCU Idle Configuration

Macros

	Name	Description
⚬	MCHP_PSF_HOOK_SET_MCU_IDLE	Configures the SoC to enter Low power mode.

10.2.13.1 MCHP_PSF_HOOK_SET_MCU_IDLE

C

```
#define MCHP_PSF_HOOK_SET_MCU_IDLE
```

Description

This hook is called when eMCHP_PSF_UPDS_IN_IDLE event notification is posted by PSF. A suitable function that puts SoC to low power mode need to be implemented in this hook.

Preconditions

None.

Returns

None.

Remarks

User definition of this Hook function is optional.

Example

```
#define MCHP_PSF_HOOK_SET_MCU_IDLE()   SAMD20_SetMCUIdle()
```

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

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.

10

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 can be called for every 1ms to 5ms 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

```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,
  eMCHP_PSF_PD_CONTRACT_NEGOTIATED,
  eMCHP_PSF_GET_SINK_CAPS_RCVD,
  eMCHP_PSF_GET_SINK_CAPS_NOT_RCVD,
  eMCHP_PSF_CAPS_MISMATCH,
  eMCHP_PSF_NEW_SRC_CAPS_RCVD
};
```

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

11

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 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.

eMCHP_PSF_PD_CONTRACT_NEGOTIATED : PSF notifies when PD contract is established with the Port partner.

eMCHP_PSF_GET_SINK_CAPS_RCVD : This event is used by PSF to notify DPM when Sink capabilities has been received from Port Partner in response to the Get_Sink_Caps message initiated by PSF on request from DPM.

eMCHP_PSF_GET_SINK_CAPS_NOT_RCVD : This event is used by PSF to notify DPM when Sink capabilities has not been received from Port Partner within tSenderResponseTimer as a response to the Get_Sink_Caps message initiated by PSF on request from DPM.

eMCHP_PSF_CAPS_MISMATCH : It is notified by PSF when there is a capability mismatch with Source partner PDOs in a PD negotiation.

eMCHP_PSF_NEW_SRC_CAPS_RCVD : It is notified by PSF when new source capability message is received from the Source Partner.

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
eMCHP_PSF_PD_CONTRACT_NEGOTIATED	PD Contract established with port partner
eMCHP_PSF_GET_SINK_CAPS_RCVD	Sink Caps received from Port Partner
eMCHP_PSF_GET_SINK_CAPS_NOT_RCVD	Sink Caps not received from Port Partner
eMCHP_PSF_CAPS_MISMATCH	Capability mismatch with Source Port Partner
eMCHP_PSF_NEW_SRC_CAPS_RCVD	New source capability message is received from Source Partner

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