System level Integration of USB Power Delivery Software Framework(PSF)

Version 0.92

Table of Contents

1 Introduction

USB Power Delivery Software Framework(PSF) is MCU-agnostic which is designed to run on different target SoCs. It provides USB-PD functionality using UPD350-USB power delivery controller. It runs on companion SoC which uses SPI/I2C interface to communicate with one or more UPD350s.

This document serves as a guide on how to integrate PSF to any SoC platform.

It also provides details on

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

1.1 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

1.2 Terms and Abbreviations

Term	Definition
USB-PD	USB Power Delivery
SPI	Serial Peripheral Interface bus - Full-duplex bus utilizing a single-master/multi-slave architecture. SPI is a 4-wire bus consisting of SPI CLK, SPI MOSI, SPI MISO and SPI SS
UPD350	Microchip UPD350 Power Delivery Port Controller
Sink Role	USB Type-C Port which sinks power from its Port Partner
Source Role	USB Type-C Port which sources power to its Port Partner
USER_APPLICATION	The software platform that runs on SoC where the PSF is integrated
SoC	system-on-chip

2 SW License Agreement

Software License Agreement

3 Hardware Integration

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

3.1 UPD350

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

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

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

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

PSF requires following hardware support from SoC to control UPD350:

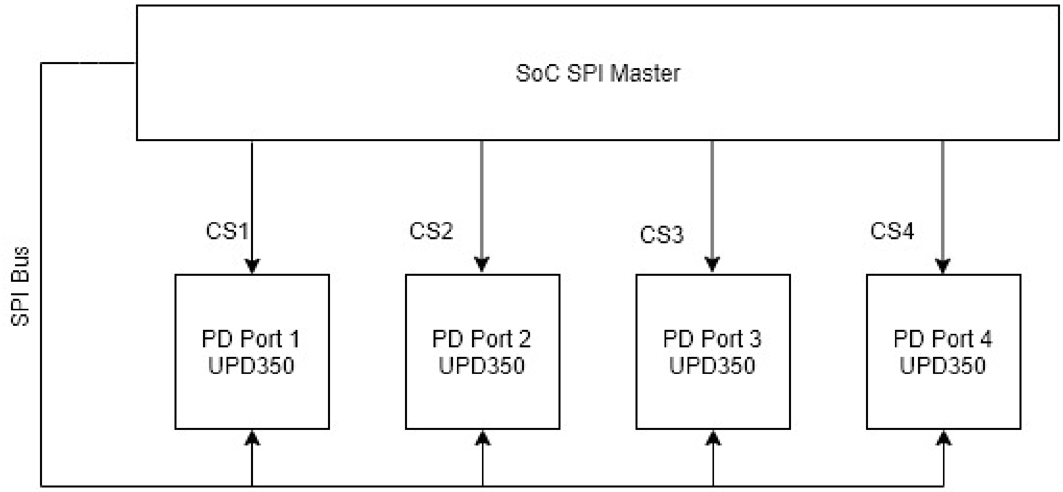
- Hardware communication interface either I2C or SPI to interact with UPD350
- PIOs per UPD350 to detect the UPD350 IRQ alert
- PIO to reset the UPD350s

3.1.1 Hardware Communication Interface

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

SPI Master:

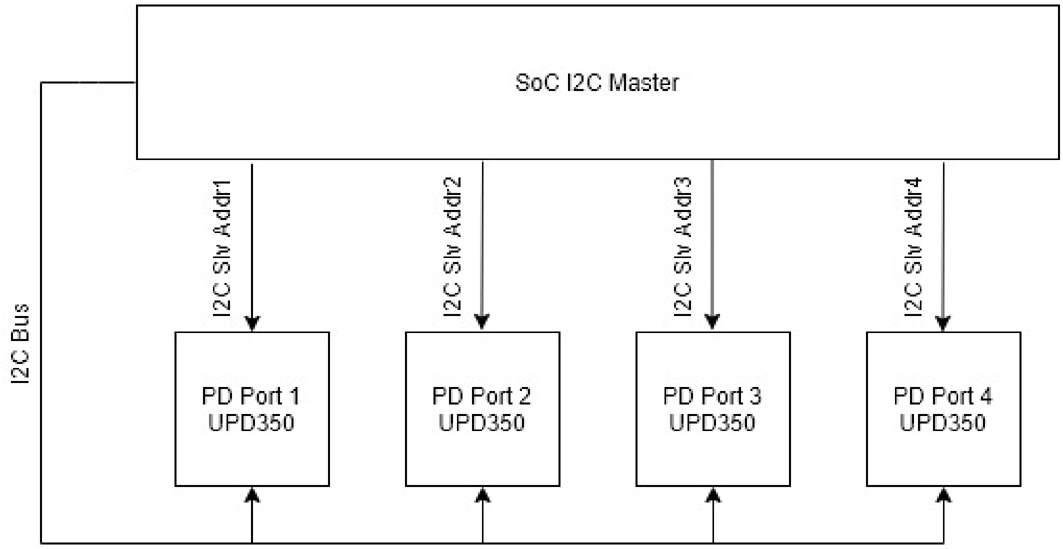
External SoC shall have SPI master support for the PSF to interact with UPD350 through SPI interface. UPD350 SPI slave supports read and write instructions at a maximum SPI Clock Frequency of 25MHz. Each PD port having a UPD350 shall have a separate SPI Chip Select(CS) line for the SoC’s SPI master to uniquely identify the SPI slave.



Only 2 Port Source and Sink solution has been tested at 8MHz SPI clock speed.

I2C Master:

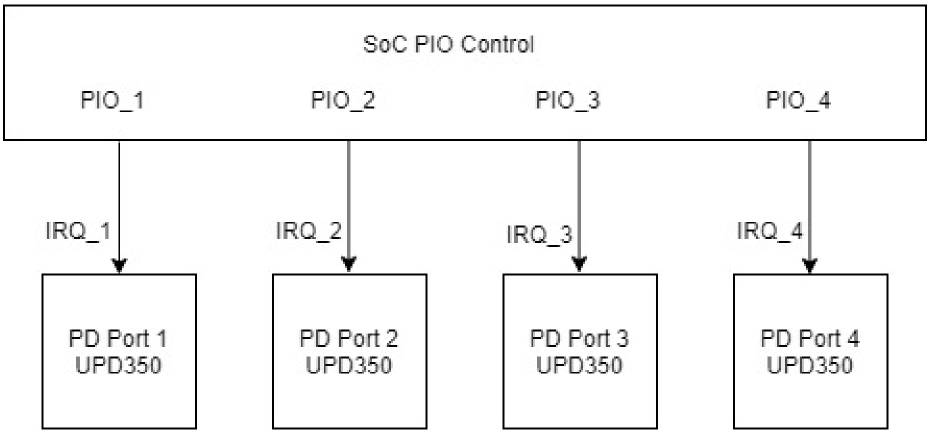
External SoC shall have I2C master support for the PSF to interact with UPD350 through I2C interface. UPD350 I2C slave supports Standard mode(100 kbps), Fast Mode(400 kbps) and Fast Mode Plus(1 Mbps) speeds. Each PD port having a UPD350 is identified by unique I2C slave address.



System level PD communication between PSF and UPD350 through I2C interface is untested, whereas basic I2C read/writes are tested.

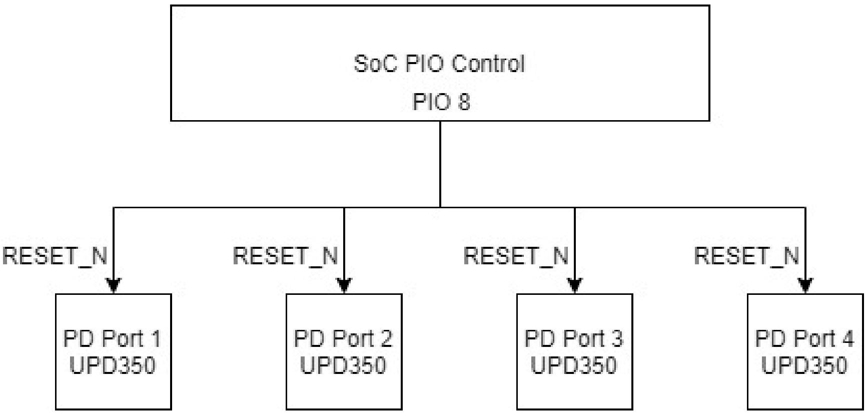
3.1.2 PIOs for UPD350 IRQs

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.



3.1.3 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.



3.2 Hardware Timer

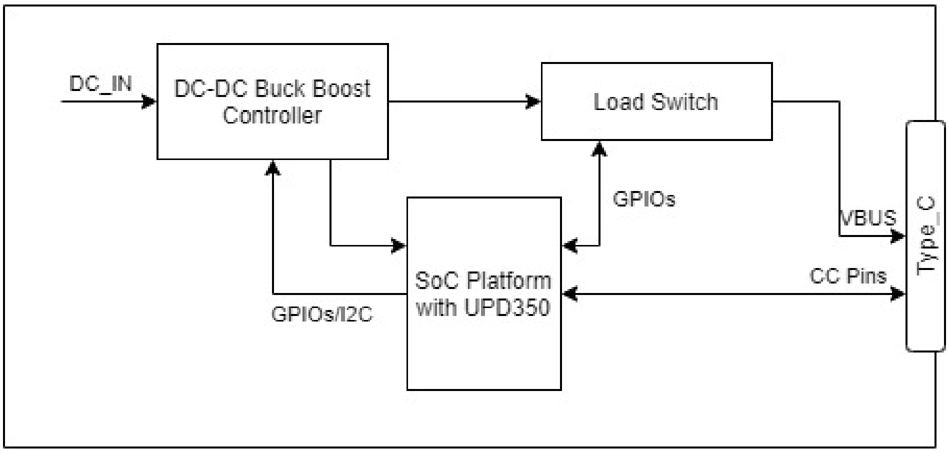
A Hardware timer with minimum resolution of 1ms is required for PSF functionality. PSF creates multiple software instance of this hardware timer. SoC User_Application shall inform the PSF the occurrence of every Timer interrupt through APIs provided.

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

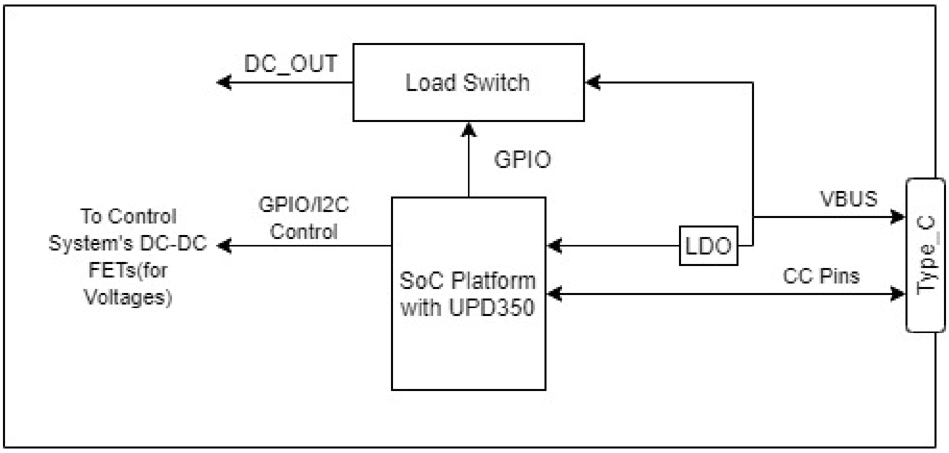
3.3 DC-DC Buck Boost Controller

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

For Source-only port, the setup would be as follows,



For Sink-Only port, the set up would be as follows,



By default, PSF supports GPIO based Microchip Power Buck controller MCP19119.

4 Software Integration

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

4.1 Memory Requirement

- **32-bit**

In a 32-bit SoC environment,

Estimated code size of PSF is **TBD**

Estimated Data RAM size is **TBD**

- **16-bit**

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

- **8-bit**

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

4.2 Multi-Port Support

PSF supports maximum of 4 ports. The tested PSF configuration is 2 port source configuration with SPI interface.

4.3 Endianness

PSF supports only Little-Endian format.

5 APIs Implementation required for SW integration

5.1 Block Diagram

5.2 Data types

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

5.3 APIs to be implemented by the User Application

5.3.1 UPD350 Hardware Interface Configurations

5.3.1.1 MCHP_PSF_HOOK_UPDHW_INTF_INIT

C

```
#define MCHP_PSF_HOOK_UPDHW_INTF_INIT 0
```

Description

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

Preconditions

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

Returns

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

Remarks

User definition of this Hook function is mandatory.

Example

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

5.3.1.2 MCHP_PSF_HOOK_UPD_COMM_ENDIS

C

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

Description

This hook is called before and after the read/write operations of 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).
u8EnDis	Parameter indicating whether to enable or disable the communication for the port.

Returns

None.

Remarks

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

Example

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

5.3.1.3 MCHP_PSF_HOOK_UPD_READ

C

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

Description

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

Preconditions

None.

Parameters

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

u8ReadLen	PSF will pass the number of bytes to be read on the SPI/I2C bus. Data type of this parameter must be UINT8.

Returns

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

Remarks

User definition of this Hook function is mandatory.

Example

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

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

5.3.1.4 MCHP_PSF_HOOK_UPD_WRITE

C

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

Description

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

Preconditions

None.

Parameters

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

Returns

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

Remarks

User definition of this Hook function is mandatory

Example

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

5.3.2 PD Timer Configuration

5.3.2.1 MCHP_PSF_HOOK_HW_PDTIMER_INIT

C

```
#define MCHP_PSF_HOOK_HW_PDTIMER_INIT 0
```

Description

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

Preconditions

None.

Returns

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

Remarks

User definition of this Hook function is mandatory

Example

```
    #define MCHP_PSF_HOOK_HW_PDTIMER_INIT()       Timer_Init()
    UINT8 Timer_Init(void);
    UINT8 Timer_Init(void)
    {
        //Initialize and start the SOC timer module for MCHP_PSF_PDTIMER_INTERRUPT_RATE
interrupt
        // frequency & register MchpPSF_PDTimerHandler as callback for timer interrupt
        //Return TRUE if Timer initialisation is successful else return FALSE
    }
```

5.3.2.2 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)
```

5.3.2.3 CONFIG_16_BIT_PDTIMEOUT

C

```
#define CONFIG_16_BIT_PDTIMEOUT 0
```

Description

CONFIG_16_BIT_PDTIMEOUT 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 CONFIG_16_BIT_PDTIMEOUT 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 CONFIG_16_BIT_PDTIMEOUT   1 (Sets timeout variable inside the PSF as unsigned
16bit)
  #define CONFIG_16_BIT_PDTIMEOUT   0 (Sets timeout variable inside the PSF as unsigned
32bit)
```

5.3.3 UPD350 IRQ Control

5.3.3.1 MCHP_PSF_HOOK_UPD_IRQ_GPIO_INIT

C

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

Description

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

Preconditions

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

Parameters

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

Returns

None.

Remarks

User definition of this Hook function is mandatory

Example

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

5.3.4 UPD350 Reset Control

5.3.4.1 MCHP_PSF_HOOK_UPD_RESET_GPIO_INIT

C

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

Description

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

Preconditions

None.

Parameters

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

Returns

None.

Remarks

User definition of this Hook function is mandatory

Example

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

5.3.4.2 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
    }
}
```

5.3.5 SOC Interrupt Enable/Disable

5.3.5.1 MCHP_PSF_HOOK_ENABLE_GLOBAL_INTERRUPT

C

```
#define MCHP_PSF_HOOK_ENABLE_GLOBAL_INTERRUPT
```

Description

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

Preconditions

None.

Returns

None.

Remarks

User definition of this Hook function is mandatory

Example

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

5.3.5.2 MCHP_PSF_HOOK_DISABLE_GLOBAL_INTERRUPT

C

```
#define MCHP_PSF_HOOK_DISABLE_GLOBAL_INTERRUPT
```

Description

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

Preconditions

None.

Returns

None.

Remarks

User definition of this Hook function is mandatory

Example

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

5.3.6 Memory Compare and Copy

5.3.6.1 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.
```

5.3.6.2 MCHP_PSF_HOOK_MEMCPY

C

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

Description

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

Preconditions

None.

Parameters

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

Returns

Returns a pointer to pDest.

Remarks

User definition of this Hook function is mandatory

Example

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

5.3.7 Structure Packing

5.3.7.1 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__))
```

5.3.7.2 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()")
```

5.3.8 Port Power Control

5.3.8.1 MCHP_PSF_HOOK_HW_PORTPWR_INIT

C

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

Description

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

Preconditions

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

Parameters

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

Returns

None.

Remarks

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

Example

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

5.3.8.2 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
  }
```

5.3.8.3 MCHP_PSF_HOOK_PORTPWR_ENDIS_VBUSDISCH

C

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

Description

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

Preconditions

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

Parameters

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

Returns

None.

Remarks

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

Example

```
#define MCHP_PSF_HOOK_PORTPWR_ENDIS_VBUSDISCH(u8PortNum, u8EnableDisable)
              hw_portpower_enab_dis_VBUSDischarge(u8PortNum, u8EnableDisable)
void hw_portpower_enab_dis_VBUSDischarge(UINT8 u8PortNum,UINT8 u8EnableDisable);
void hw_portpower_enab_dis_VBUSDischarge(UINT8 u8PortNum,UINT8 u8EnableDisable)
{
    if (TRUE == u8EnableDisable)
    {
        //Enable the VBUS Discharge for "u8PortNum" Port
    }
    else
    {
        //Disable the VBUS Discharge for "u8PortNum" Port
    }
}
```

5.3.8.4 MCHP_PSF_HOOK_PORTPWR_CONFIG_SINK_HW

C

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

Description

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

Preconditions

It is applicable only for Sink operation.

Parameters

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

Returns

None.

Remarks

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

Example

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

5.3.9 Boot time Configuration

5.3.9.1 _PortData_cfg Structure

C

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

Description

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

Members

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

Remarks

None

5.3.9.2 MCHP_PSF_HOOK_BOOT_TIME_CONFIG

C

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

Description

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

Preconditions

None.

Parameters

Parameters	Description
CfgGlobals	Holds the structure pointer of the structure PORT_CONFIG_DATA

Returns

None.

Remarks

User definition of this Hook function is optional

Example

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

5.3.10 Hooks for Policy Manager

5.3.10.1 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
    }
```

5.3.11 Debug Hook Functions

Debug Hook Functions will be enabled if CONFIG_HOOK_DEBUG_MSG is set to 1. Debug hook functions must be very short to make device complaint to USB-PD. USER_APPLICATION can choose any DEBUG_MODULE by providing appropriate debug hook functions.

5.3.11.1 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
    }
```

5.3.11.2 MCHP_PSF_HOOK_DEBUG_STRING

C

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

Description

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

Preconditions

None.

Parameters

Parameters	Description
pcharBuf	Pointer to the character buffer

Returns

None.

Remarks

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

Example

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

5.3.11.3 MCHP_PSF_HOOK_DEBUG_UINT8

C

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

Description

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

Preconditions

None.

Parameters

Parameters	Description
u8Val	UINT8 data to be sent to Debug interface

Returns

None.

Remarks

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

Example

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

5.3.11.4 MCHP_PSF_HOOK_DEBUG_UINT16

C

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

Description

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

Preconditions

None.

Parameters

Parameters	Description
u32Val	UINT32 data to be sent to Debug interface

Returns

None.

Remarks

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

Example

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

5.3.11.5 MCHP_PSF_HOOK_DEBUG_UINT32

C

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

Description

This is macro MCHP_PSF_HOOK_DEBUG_UINT32.

5.3.11.6 MCHP_PSF_HOOK_DEBUG_INT32

C

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

Description

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

Preconditions

None.

Parameters

Parameters	Description
i32Val	INT32 data to be sent to DEBUG_MODULE

Returns

None.

Remarks

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

Example

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

5.3.12 PD Firmware Upgrade

5.3.12.1 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
}
```

5.3.12.2 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.

Returns

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

Remarks

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

Example

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

5.3.12.3 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 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;
}
```

5.3.12.4 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

5

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 (MCHP_PSF_HOOK_MEMCMP(&u8ResponseBuffer[0],
&u8RequestBuffer[4],u16RequestLength) == 0)
      {
        //Set the status as OK
        u8Status = ePE_FWUP_OK;
      }
      else
      {
        //Verification Stage failure
        u8Status = ePE_FWUP_errVERIFY;
      }
    }
    else
    {
      u8Status = ePE_FWUP_errADDRESS;
    }

    return u8Status;
  }
```

5.3.12.5 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.
    }
```

5.3.12.6 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 INCLUDE_PDFU is TRUE.

Example 1

1. 0x01 - Corresponds to Bootloader Application
2. 0x02 - Corresponds to Fixed Application
3. 0x03 - Corresponds to Updatable Application

Example 2

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

5.4 APIs to be called by the User Application

For PSF integration, few of PSF APIs shall be called by the user application

5.4.1 MchpPSF_Init Function

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.

5.4.2 MchpPSF_PDTimerHandler Function

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

None

5.4.3 MchpPSF_RUN Function

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

Multi threaded Free RTOS environment is untested and latency of API call is to identified yet.

5.4.4 MchpPSF_UPDIrqHandler Function

C

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

Description

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

Preconditions

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

Parameters

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

Returns

None

Remarks

None

6 Notification callback from PSF

6.1 MCHP_PSF_NOTIFICATION Enumeration

C

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

Description

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

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

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

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

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

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

eMCHP_PSF_VBUS_PWR_FAULT : PSF notifies all VBUS power fault VBUS Over voltage, VBUS under voltage, VBUS OCS via this notification. For this notification, PSF expects a return value to decide whether to handle the fault occurred.When user returns TRUE for power fault, Incase of explicit contract, if power fault count is less than CONFIG_MAX_VBUS_POWER_FAULT_COUNT,PSF DPM power fault manager handles it by sending Hard Reset. When the power fault count exceeds the max fault count,CC termination on the port is removed until the physical detach of the port partner. Incase of implicit contract, PSF handles by entering TypeC Error Recovery. This notification occurs only when INCLUDE_POWER_FAULT_HANDLING is defined as 1.

Members

Members	Description
eMCHP_PSF_TYPEC_DETACH_EVENT = 1	Detach event has occurred
eMCHP_PSF_TYPEC_CC1_ATTACH	Port partner attached at CC1 orientation

eMCHP_PSF_TYPEC_CC2_ATTACH	Port partner attached at CC2 orientation
eMCHP_PSF_UPDS_IN_IDLE	All the UPD350s are in Idle
eMCHP_PSF_VCONN_PWR_FAULT	VCONN Power Fault has occurred
eMCHP_PSF_VBUS_PWR_FAULT	VBUS Power Fault has occurred

Remarks

None

6.2 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
}
```