UM2552
User manual

Managing USB power delivery systems with STM32 microcontrollers

Introduction

This document details how to use the X-CUBE-USB-PD software for managing USB power delivery systems based on STM32 microcontrollers.

It describes:

- how to create a software architecture that fulfills the specified requirements
- how to evaluate the requirements generated by the hardware architecture
- how to select a suitable set of run-time system interfaces, user interfaces, and data formats and representations.

.

Contents

List of tables

List of figures

1 Generalities

Table 1. List of acronyms

Acronym	Definition
AMS	Atomic message sequence
APDO	Augmented power data object
BSP	Board support package
CAD	Cable detection module
DFP	Downstream facing port
DPM	Device policy manager
DRP	Dual role power (ability for a product to either source or sink power)
DRS	Data role swap
FWUP	Firmware update
GUI	Graphical user interface
HAL	Hardware abstraction layer
HW	Hardware
LL	Low layer
MSC	Message sequence chart
PDO	Power data object
PE	Policy engine
PHY	Physical layer
PPS	Programmable power supply
PRL	Physical protocol layer
PRS	Power role swap
SNK	Power sink capability
SRC	Power source capability
TCPM	Type C port manager
TCPC	Type C port controller
UCPD	USB Type-C™ power delivery
UFP	Upstream facing port
USB-PD	Universal serial bus - Power delivery
VDM	Vendor defined messages

Table 2. Standards

Name		Title	Version
[1]	USB 3.0 Promoter Group	Universal Serial Bus Type-C Cable and Connector Specification	Edition 1.3 July 14, 2017
[2]		Universal Serial Bus Power Delivery Specification	Edition 1.2 June 21, 2018
[3]	VESA® DISPLAYPORT™	VESA DisplayPort Alt Mode on USB Type-C standard	Version 1.0a August 5, 2015

STM32 microcontrollers are based on Arm®[a] cores.

a. Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

2 General design aspects

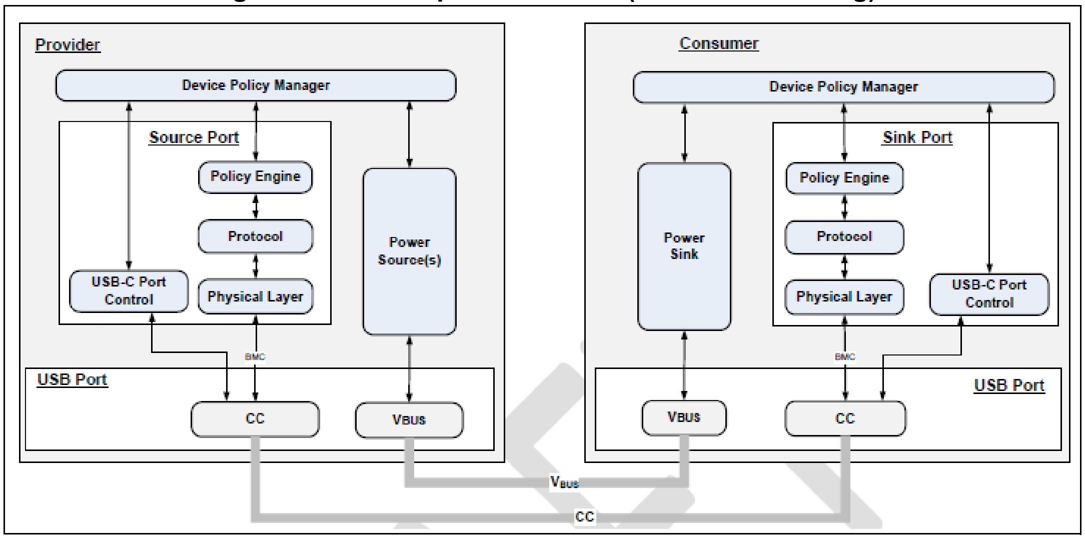
2.1 Context

The USB power delivery specification defines a power delivery system covering all elements of an USB system including hosts, devices, hubs, chargers and cable assemblies. This specification describes the architecture, protocols, power supply behavior, connectors and cabling necessary for managing power delivery over USB at up to 100 W.

The communication stack can be broken down as shown in *Figure 1*:

- a device policy manager that exists in all devices and manages USB power delivery resources within the device across one or more ports based on the device local policy.
- a policy engine that exists in each USB power delivery port implements the local policy for that Port.
- a protocol layer that enables messages to be exchanged between a source port and a sink port.
- a physical layer that handles transmission and reception of bits on the wire and handles data transmission.
- the USB-C port control that handles the type C detection state machines.

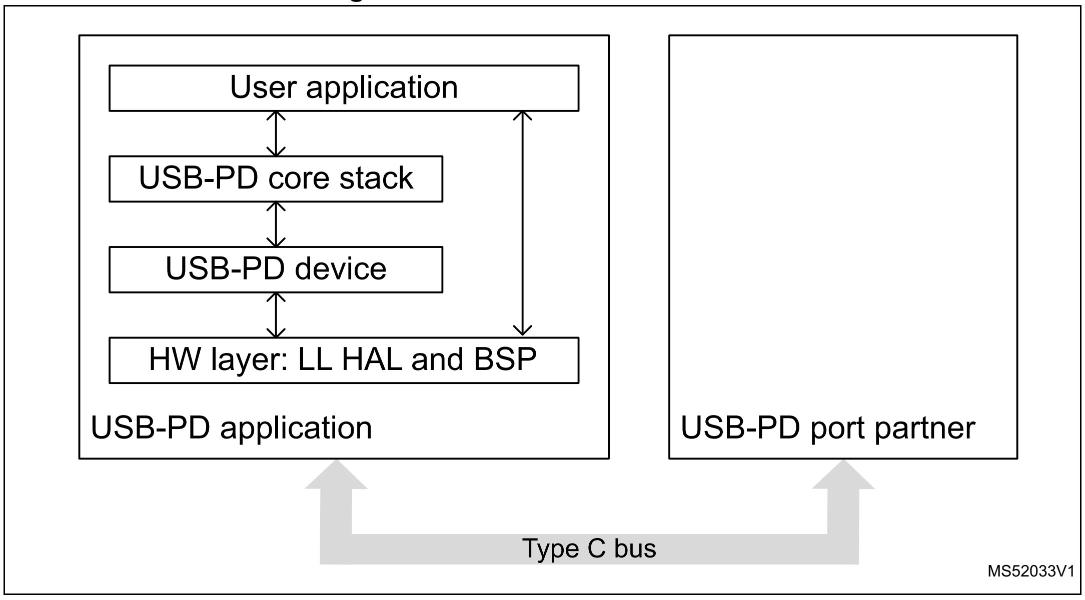
Figure 1. USB-PD protocol stack (from www.usb.org)



The different layers used to build an USB-PD application are shown in *Figure 2*:

- User application, corresponding to the Device policy manager + Power management
- USB-PD core stack, corresponding to Policy engine + Protocol layer
- USB-PD devices is equivalent to Physical layer + USB-C port control
- HW layer: the different resources used to interact with the hardware HAL, LL and BSP

Figure 2. STM32 USB-PD overview



2.2 Memory budget

Different libraries can be selected, depending upon the supported features in the stack, see *Section 3.6.3: Library description* for more details.

Table 3. Maximum memory budget for the USB-PD stack, PD3 configuration

Block	Flash memory	RAM
PE	26192 bytes	1620 bytes
PRL	3052 bytes	212 bytes
CAD	2476 bytes	44 bytes

Note: This budget has been estimated with EWARM V8.20 with High size optimization. It is an indication and may vary upon library versions.

3 Architecture description

Figure 3. Modules overview

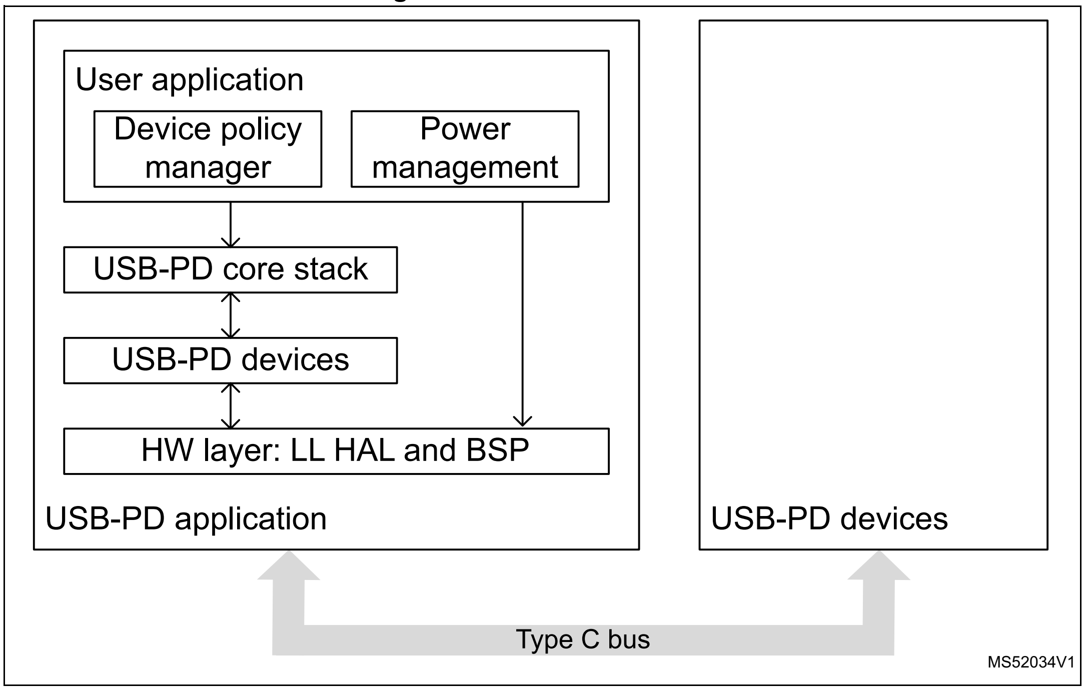
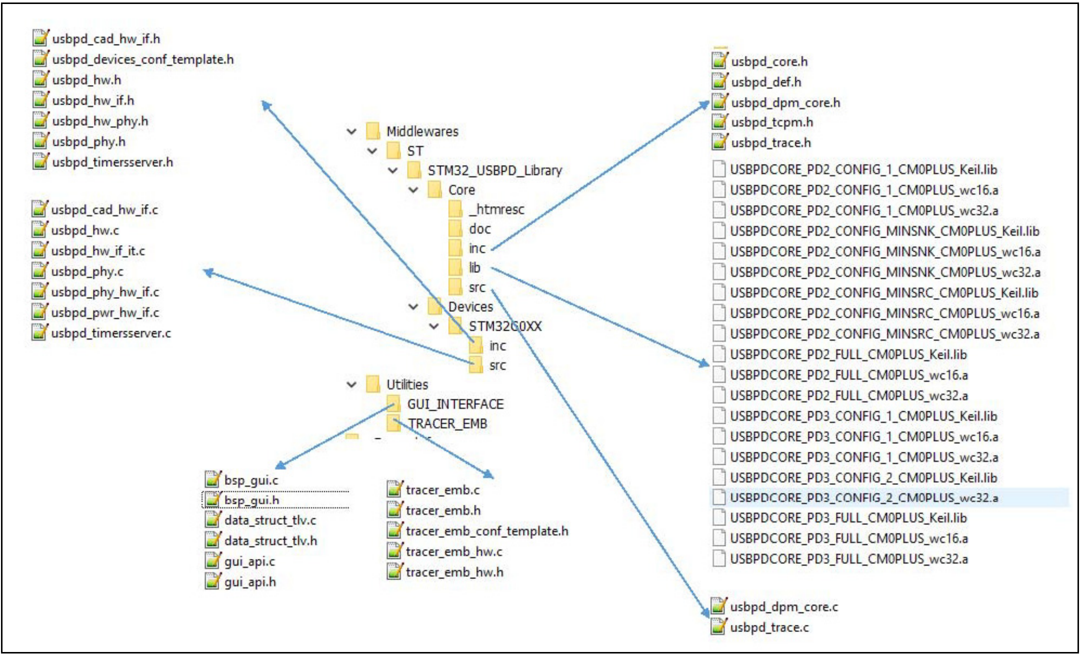


Figure 3 shows the different layers used to build an USB-PD application.

- User application:
  - Device policy manager: user application interacting with the USB-PD core stack and make the link with Power management
  - Power management: user application parts in charge of the power management VBUS and VCONN
- USB-PD core stack: USB-PD protocol stack
- USB-PD devices (depend upon the STM32 Series and the required services needed by the stack): embedded the physical layer (i.e. low level of the type C state machine plus low level of message transport)
- HW layer: the different LL and HAL drivers and BSP package used to control the hardware behavior
- Utilities:
  - TRACER_EMB: utility to trace the system behavior
  - GUI_INTERFACE: embedded interface for UCPD monitor

Figure 4. USB-PD files organization

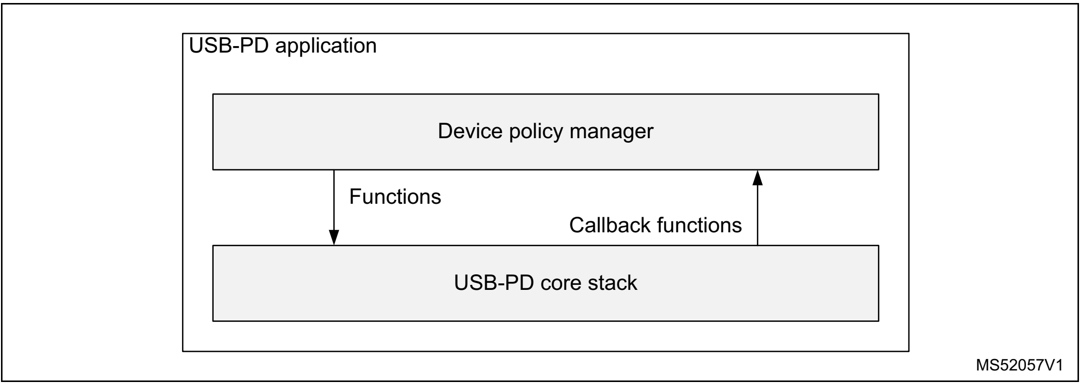

usbpd_cad_hw_if.h
usbpd_devices_conf_template.h
usbpd_hw.h
usbpd_hw_if.h
usbpd_hw_phy.h
usbpd_phy.h
usbpd_timersserver.h
usbpd_cad_hw_if.c
usbpd_hw.c
usbpd_hw_if_it.c
usbpd_phy.c
usbpd_phy_hw_if.c
usbpd_pwr_hw_if.c
usbpd_timersserver.c
Middlewares
ST
STM32_USBPD_Library
Core
_htmresc
doc
inc
lib
src
Devices
STM32G0XX
inc
src
Utilities
GUI_INTERFACE
TRACER_EMB
bsp_gui.c
bsp_gui.h
data_struct_tlv.c
data_struct_tlv.h
gui_api.c
gui_api.h
tracer_emb.c
tracer_emb.h
tracer_emb_conf_template.h
tracer_emb_hw.c
tracer_emb_hw.h
usbpd_core.h
usbpd_def.h
usbpd_dpm_core.h
usbpd_tcpm.h
usbpd_trace.h
USBPDCORE_PD2_CONFIG_1_CM0PLUS_Keil.lib
USBPDCORE_PD2_CONFIG_1_CM0PLUS_wc16.a
USBPDCORE_PD2_CONFIG_1_CM0PLUS_wc32.a
USBPDCORE_PD2_CONFIG_MINSNK_CM0PLUS_Keil.lib
USBPDCORE_PD2_CONFIG_MINSNK_CM0PLUS_wc16.a
USBPDCORE_PD2_CONFIG_MINSNK_CM0PLUS_wc32.a
USBPDCORE_PD2_CONFIG_MINSRC_CM0PLUS_Keil.lib
USBPDCORE_PD2_CONFIG_MINSRC_CM0PLUS_wc16.a
USBPDCORE_PD2_CONFIG_MINSRC_CM0PLUS_wc32.a
USBPDCORE_PD2_FULL_CM0PLUS_Keil.lib
USBPDCORE_PD2_FULL_CM0PLUS_wc16.a
USBPDCORE_PD2_FULL_CM0PLUS_wc32.a
USBPDCORE_PD3_CONFIG_1_CM0PLUS_Keil.lib
USBPDCORE_PD3_CONFIG_1_CM0PLUS_wc16.a
USBPDCORE_PD3_CONFIG_1_CM0PLUS_wc32.a
USBPDCORE_PD3_CONFIG_2_CM0PLUS_Keil.lib
USBPDCORE_PD3_CONFIG_2_CM0PLUS_wc32.a
USBPDCORE_PD3_FULL_CM0PLUS_Keil.lib
USBPDCORE_PD3_FULL_CM0PLUS_wc16.a
USBPDCORE_PD3_FULL_CM0PLUS_wc32.a
usbpd_dpm_core.c
usbpd_trace.c

3.1 User application / USB-PD core stack interface

All the interfaces used between the user application and the USB-PD core stack are described within the folder Firmware\Middlewares\ST\STM32_USBPD_Library\Core\doc. They are divided in two parts:

- Callbacks used by USB-PD core stack to interact with the application
- USB-PD core stack APIs.

Figure 5. Interfaces DPM / core stack



3.1.1 Callbacks API

- USBPD_PE_Callbacks Struct Reference: lists all the callback functions used by PE to exchange information with DPM.
- USBPD_CAD_Callbacks Struct Reference: lists all the callback functions used by Cable accessory detection to exchange information with the DPM.
- USBPD_VDM_Callbacks Struct Reference: list all the callback functions used by PE Engine to exchange information with DPM.

3.1.2 Core stack API list

- USBPD CORE PE Exported Functions to DPM CORE: lists all the functions used by DPM CORE to start a USB-PD application.
- USBPD CORE PE Exported Functions to DPM USER: lists all the functions to initiate a PD request.
- USBPD CORE CAD Exported Functions: lists all the functions to control the detection state machine.
- USBPD CORE PE Exported Functions to VDM USER: lists all the functions to control the VDM exchanges.

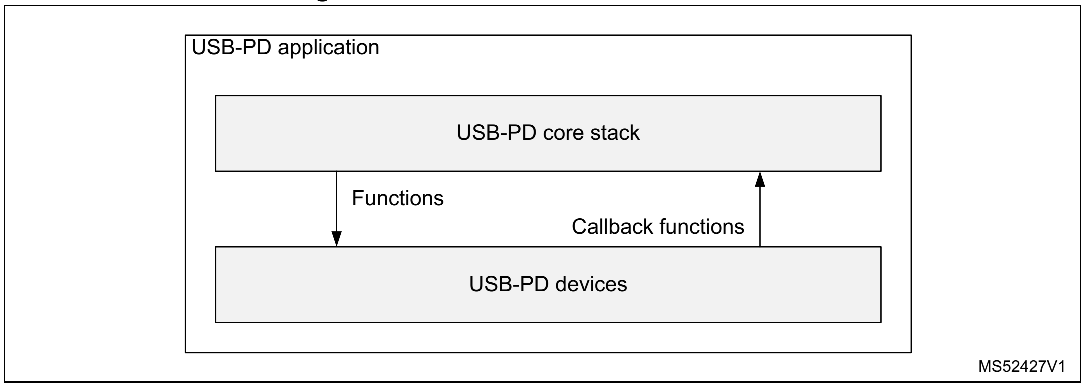
3.2 USB-PD device / HW layer, and user application / HW layer interfaces

All the interfaces used between the USB-PD core stack and the USB-PD device, are described within the folder Firmware\Middlewares\ST\STM32_USBPD_Library\Devices\STM32YYXXX\Doc callbacks used by USB-PD device to interact with the core stack USB-PD devices API.

The HW layer interfaces are linked with the usage of HAL and LL driver inside the device. There are two important rules about these interfaces:

- USB-PD behavior is handled through the LL UCPD driver
- power management is performed through the BSP power.

Figure 6. Interfaces devices / core stack



3.2.1 Callbacks API

- USBPD_PHY_Callbacks Struct Reference

3.2.2 Device interfaces

- USBPD DEVICE_CAD HW IF Exported Functions
- USBPD DEVICE PHY Exported Functions
- USBPD_DEVICE_TIMESERVER Exported Functions

3.2.3 LL UCPD interfaces

See the HAL/LL documentation for the interfaces available in Firmware\Drivers\STM32XXxx_HAL_Driver

3.2.4 BSP power interfaces

The BSP power interfaces are described inside the BSP component provided for each board supporting USBPD feature. The full description of the available APIs can be found in the STM32XXYYY_EVAL_POWER section of the BSP component.

3.3 Application initialization

For application initialization, it is recommend the use of “usbpd_dpm_core.c” file, provided inside the USB-PD Core stack module. This file contains the functions *USB-PD_DPM_InitCore()* and *USB-PD_DPM_InitOS()* in charge of initialization of structures and modules. The provided functions can be used as templates if adaptations are needed.

As detailed in the code below, the initialization of the USB-PD application is carried out in five steps (step 4 must be performed only in RTOS context).

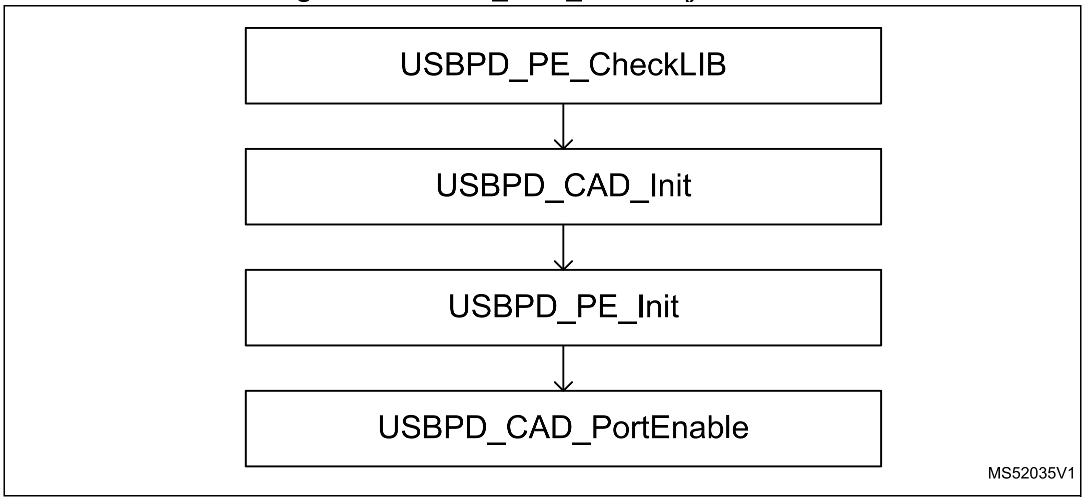
```
int main(void)
{
  …
  /* Step 1 : Global Init of USBPD HW */
  USBPD_HW_IF_GlobalHwInit();

  /* Step 2 : Initialize the Device Policy Manager */
  if( USBPD_ERROR == USBPD_DPM_InitCore())
  {
    /* error on core init  */
    while(1);
  }

  /* Step 3 : Initialise the DPM application */
  if (USBPD_OK != USBPD_DPM_UserInit())
  {
    while(1);
  }
#ifdef _RTOS
  /* Step 4 : Initialize the Device Policy Manager */
  if( USBPD_ERROR == USBPD_DPM_InitOS())
  {
    /* error the OS init  */
    while(1);
  }
#endif
  /* Step 5 : Run the application */
  USBPD_DPM_Run();
}
```

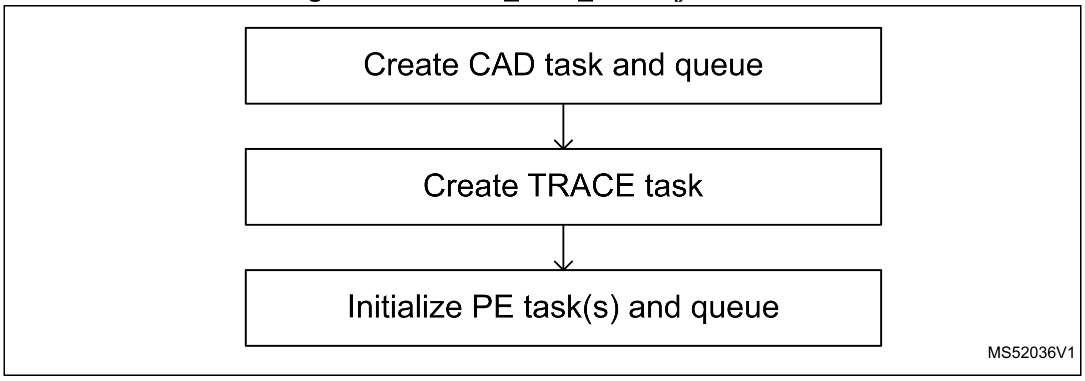
The second step corresponds to the USB-PD stack initialization. It is achieved by *USB-PD_DPM_InitCore()* function (see *Figure 7*).

Figure 7. *USB-PD_DPM_InitCore()* function



The fourth step (see *Figure 8*) consists in the initialization of the RTOS objects (task, queue).

Figure 8. *USB-PD_DPM_InitOS()* function



The implementation of the fifth step depends on the use of an RTOS system by the application.

- In the RTOS case (each task is independent), the call of the function osKernelStart starts the execution of the type C state machine (function USB-PD_CAD_Process). PE tasks are launched dynamically depending upon the detection status (on the event attach, a PE task is created and only killed on the detach event).
- In the non RTOS case, the application is an infinite loop on the different layers: type C state machine detection, policy engine and user application code.

An example of code for the *USB-PD_DPM_Run()* function is detailed below.

```
void USBPD_DPM_Run(void)
{
#ifdef _RTOS
  osKernelStart();
#else
  Do {
    (void)USBPD_CAD_Process();

    if((HAL_GetTick() - DPM_Sleep_start[USB-PD_PORT_0]) >
    {

      DPM_Sleep_time[USB-PD_PORT_0] =
  #ifdef _DRP
        USBPD_PE_StateMachine_DRP(USB-PD_PORT_0);
  #elif _SRC
        USBPD_PE_StateMachine_SRC(USB-PD_PORT_0);
  #elif _SNK
        USBPD_PE_StateMachine_SNK(USB-PD_PORT_0);
  #endif /* _DRP */
      DPM_Sleep_start[USB-PD_PORT_0] = HAL_GetTick();
    }

    USBPD_DPM_UserExecute(NULL);
    (void)USBPD_TRACE_TX_Process();
 } while(1u == 1u);
#endif /* _RTOS */
}
```

Note: *In the RTOS case, the user application is still driven by the USB-PD_DPM_UserExecute() function, but its execution must be associated with an RTOS task (the initialization is managed inside the USBPD_DPM_UserInit() function).*

3.4 User application

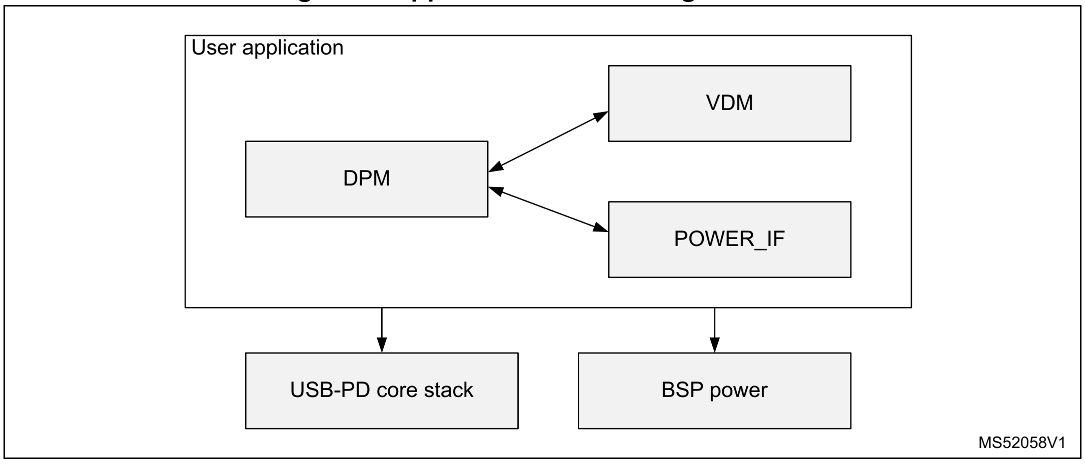
3.4.1 Purpose

The user application part is delivered to customer as a template and provides all the interfaces needed to build an USB-PD application.

The application has been divided in three different parts (see *Figure 9*):

1. DPM: device policy manager
2. VDM: vendor defined message
3. POWER_IF: power management

Figure 9. Application user file organization



3.4.2 File organization

Table 4 describes the function of each file and to which user application sub-module they belong to.

Table 4. File organization

File	Description	Sub-module
usbpd_dpm_user.c/h	Provides the PE callbacks and DPM function	DPM
usbpd_dpm_core.c/h	In charge of the application initialization, this file is optional and provided inside USB-PD core stack folder as an initialization template	
usbpd_dpm_conf.h	Mandatory, contains the ports configuration	
usbpd_vdm_user.c/h	Optional, needed only if application manages VDM feature	VDM
usbpd_pwr_if.c/h	Manages the power profile	PWR
usbpd_pdo_defs.h	Ports PDO definition	

3.4.3 Mechanisms used to interact with the stack

The application must interact with the USB-PD stack and this is done through callbacks. At initialization the application registers all the callback functions by a call to following functions:

- *USBPD_PE_Init* (managed inside the DPM core template) for policy engine module
- *USBPD_CAD_Init* for the cable detection module
- *USBPD_VDM_UserInit* for the vendor message management

Three different callbacks are used, namely action, data sharing and notification callbacks.

Action callbacks

They are used by the USB-PD stack to request an action by the application or to get a status (e.g. REJECT, ACCEPT).

All the callback functions are defined inside the interface of PE and CAD.

Data sharing callbacks

The USB-PD stack and the application must share data. For example, in the case of source application, the stack sends the message “SOURCE CAPABILITY” that must contain the Power data objects, this information is driven by the application according the available power and port profile (Power data objects can change during runtime according the application status).

Data exchange can be performed both ways, application to stack and stack to application.

Table 5. Data sharing

Function	Description
USB-PD_PE_GetDataInfo	Callback used by the stack to read data from the application
USB-PD_PE_SetDataInfo	Callback used by the stack to send data to the application

The prototypes of these functions are similar; the most important parameter is the USB-PD_CORE_DataInfoType_TypeDef DataId used to identify the kind of data (for the list see *Appendix A*).

Notification callbacks

The notifications are used by the stack to inform the application about the different events detected inside the stack (only sent in stack to application direction). The list of the available events is described in *Appendix B*.

3.4.4 DPM

DPM is mainly responsible for managing the power used by one or more USB-PD ports and performing power negotiations with distant ports according to power capability evolution. DPM must also interact with the stack to obtain information about type of cable insertion and is responsible for answering to requests sent by the port partner. DPM embeds the USB-PD stack configuration done by the user.

USBPD_DPM_UserInit

This function must be defined and can be updated by the user. Inside the ST project, it is used to initialize the application, as shown below:

```
USBPD_StatusTypeDef USBPD_DPM_UserInit(void)
{
  /* PWR_IF initialization */
  if(USBPD_OK !=  USBPD_PWR_IF_Init())
  {
    return USBPD_ERROR;
  }

 /* VDM initialisation */
 if(USBPD_TRUE == DPM_Settings[USBPD_PORT_0].PE_VDMSupport)
 {
   if (USBPD_OK != USBPD_VDM_UserInit(USBPD_PORT_0))
   {
    return USBPD_ERROR;
   }
 }

 osMessageQDef(MsgBox, DPM_BOX_MESSAGES_MAX, uint32_t);
 DPMMsgBox = osMessageCreate(osMessageQ(MsgBox), NULL);
 osThreadDef(DPM, USBPD_DPM_UserExecute, osPriorityLow, 0, 300);

 if(NULL == osThreadCreate(osThread(DPM), &DPMMsgBox))
 {
   return USBPD_ERROR;
 }

 return USBPD_OK;
}
```

This function initializes both POWER_IF and VDM modules. Optionally, a DPM task can be created to handle the user application needs.

DPM_Settings

The file usbpd_dpm_conf.h represents the USB-PD stack configuration; Structure content is read-only, meaning it cannot be updated at running time (the stack configuration cannot be updated dynamically).

For details see the core documentation USBPD_SettingsTypeDef Struct Reference.

DPM_Params

The structure DPM_Params is initialized by usbpd_dpm_core and contains a dynamic view of the type C port. This information is shared between DPM and the stack.

For details see the core documentation USBPD_ParamsTypeDef Struct Reference.

DPM_Ports

This variable is used inside our applications to store different data received from the port partners or data corresponding to the port state. The definition of this variable is only an example and can be updated by customer according to application needs.

For details see the file usbpd_dpm_user.h, which contains the structure definition for USBPD_HandleTypeDef.

3.4.5 VDM

VDM is responsible for managing all the VDM transactions and contains all the VDM settings.

Initialization

Initialization

```
USBPD_StatusTypeDef USBPD_VDM_UserInit(uint8_t PortNum)
{
(…)
  USBPD_PE_InitVDM_Callback(PortNum, (USBPD_VDM_Callbacks *)&vdmCallbacks);
(…)
}
```

All the VDM management (SOP, SOP' and SOP'') are defined in usbpd_vdm_user.c/h code. Initialization is done thanks to the function 'USBPD_VDM_UserInit' called by DPM user code.

Main goal of this function is to register all the VDM callbacks that can be called by PE stack.

To enable VDM on your project, use the Stack Library USBPDXXXX_PDX_FULL_xx.

If you just need to communicate with an EMC cable, a USBPDXXXX_PDX_CONFIG_1_xx can be used.

DPM_VDM_Settings

This structure is defined inside the fiel usbpd_vdm_user.h and includes a description of each file of the structure USBPD_VDM_SettingsTypeDef.

3.4.6 POWER_IF

The POWER_IF module manages the PDO and has no direct interface with the USB-PD stack. POWER_IF functions are only called by DPM. This module is an implementation example so the code can be updated according to application needs.

PDO definition

In usbpd_pdo_def.h, tables of uint32_t with a max size equal to USB-PD_MAX_NB_PDO must be defined, to describe PDO/APDO of the application. Tables must be defined for each port, one table for SOURCE capabilities, one for SINK capabilities. For instance, a DRP port must define two tables for a port, one for the SOURCE role and a second for the SINK role. In case of an application with two ports DRP, four different tables must be embedded:

- PORT0_PDO_ListSRC
- PORT0_PDO_ListSNK
- PORT1_PDO_ListSRC
- PORT1_PDO_ListSNK

The number of defined PDO for each role/each port has to be defined as constants:

- PORT0_NB_SOURCEPDO
- PORT1_NB_SOURCEPDO
- PORT0_NB_SINKPDO
- PORT1_NB_SINKPDO
- PORT0_NB_SOURCEAPDO
- PORT1_NB_SOURCEAPDO
- PORT0_NB_SINKAPDO
- PORT1_NB_SINKAPDO

An example of SOURCE PDO table empty definition is shown below.

```
#define PORT0_NB_SOURCEPDO         2
#define PORT0_NB_SOURCEAPDO        1

uint32_t PORT0_PDO_ListSRC[USBPD_MAX_NB_PDO] =
{
  /* PDO 1 */
  (((PWR_A_10MA(1.5)) << USBPD_PDO_SRC_FIXED_MAX_CURRENT_Pos) |
   ((PWR_V_50MV(5)) << USBPD_PDO_SRC_FIXED_VOLTAGE_Pos)       |
         USBPD_PDO_SRC_FIXED_PEAKCURRENT_EQUAL                |
         USBPD_PDO_SRC_FIXED_UNCHUNK_NOT_SUPPORTED            |
         USBPD_PDO_SRC_FIXED_DRD_SUPPORTED                    |
         USBPD_PDO_SRC_FIXED_USBCOMM_NOT_SUPPORTED            |
         USBPD_PDO_SRC_FIXED_EXT_POWER_NOT_AVAILABLE          |
         USBPD_PDO_SRC_FIXED_USBSUSPEND_NOT_SUPPORTED         |
   USBPD_PDO_SRC_FIXED_DRP_SUPPORTED     |
         USBPD_PDO_TYPE_FIXED
  )
  /* PDO 2 */
  (((PWR_A_10MA(1.5)) << USBPD_PDO_SRC_FIXED_MAX_CURRENT_Pos) |
   ((PWR_V_50MV(9)) << USBPD_PDO_SRC_FIXED_VOLTAGE_Pos)       |
   USBPD_PDO_SRC_FIXED_PEAKCURRENT_EQUAL                      |
   USBPD_PDO_TYPE_FIXED
  ),
  /* PDO 3 : SRC APDO */
  ((((PWR_A_50MA(3)) << USBPD_PDO_SRC_APDO_MAX_CURRENT_Pos)   |
   (((PWR_V_100MV(3))<< USBPD_PDO_SRC_APDO_MIN_VOLTAGE_Pos)   |
   (((PWR_V_100MV(9))<< USBPD_PDO_SRC_APDO_MAX_VOLTAGE_Pos)   |
   USBPD_PDO_TYPE_APDO
  ),
  /* PDO 4 */ (0x00000000U),
  /* PDO 5 */ (0x00000000U),
```

In this example the SRC capabilities are defined with three different PDOs (two fixed PDOs and one APDO):

1. fixed PDO 5 V 1.5 A
2. fixed PDO 9 V 1.5 A
3. APDO 3 A 3-9 V

The first mandatory PDO is specific: it integrates information about the port capabilities:

- un-chunk
- data role swap
- USB communication
- external power
- USB suspend
- dual role power support.

USB-PD standard support different kinds of source power objects (see the USB-PD specification): FIXED, VARIABLE, BATTERY and APDO power data object.

An example of SINK PDO table definition is shown below:

```
#define PORT0_NB_SINKPDO         1
#define PORT0_NB_SINKAPDO        1

uint32_t PORT0_PDO_ListSNK[USBPD_MAX_NB_PDO] =
{
  /* PDO 1 */
  (((PWR_A_10MA(1.5)) << USBPD_PDO_SNK_FIXED_OP_CURRENT_Pos) |
   ((PWR_V_50MV(5)) << USBPD_PDO_SNK_FIXED_VOLTAGE_Pos)      |
   USBPD_PDO_SNK_FIXED_FRS_NOT_SUPPORTED                     |
   USBPD_PDO_SNK_FIXED_DRD_SUPPORTED                         |
   USBPD_PDO_SNK_FIXED_USBCOMM_NOT_SUPPORTED                 |
   USBPD_PDO_SNK_FIXED_EXT_POWER_NOT_AVAILABLE               |
   USBPD_PDO_SNK_FIXED_HIGHERCAPAB_NOT_SUPPORTED             |
   USBPD_PDO_SNK_FIXED_DRP_SUPPORTED                         |
   USBPD_PDO_TYPE_FIXED
  ),
  /* PDO 2 : SRC APDO */
  ((((PWR_A_50MA(1.5)) << USBPD_PDO_SNK_APDO_MAX_CURRENT_Pos) |
   (((PWR_V_100MV(3))  << USBPD_PDO_SNK_APDO_MIN_VOLTAGE_Pos) |
   (((PWR_V_100MV(5.9)) << USBPD_PDO_SNK_APDO_MAX_VOLTAGE_Pos)|
   USBPD_PDO_TYPE_APDO
  ),
  /* PDO 3 */ (0x00000000U),
  /* PDO 4 */ (0x00000000U),
  /* PDO 5 */ (0x00000000U),
```

In this example the SNK PDO is defined with two different PDOs (one fixed and one APDO):

1. fixed PDO 5 V 1.5 A
2. APDO 1.5 A 3-5.9 V

The first mandatory PDO is specific: it integrates information about the port capabilities:

- fast role swap
- data role swap
- USB communication
- external power
- USB suspend
- dual role power support.

USBPD_PWR_IF APIs

`USBPD_StatusTypeDef USBPD_PWR_IF_Init(void)`

Initializes structures and variables related to power board profiles used by sink and source, for all available ports.

`USBPD_StatusTypeDef USBPD_PWR_IF_SetProfile(uint8_t PortNum)`

Sets the required power profile.

`USBPD_StatusTypeDef USBPD_PWR_IF_SupplyReady(uint8_t PortNum,`

`USBPD_VSAFE_StatusTypeDef Vsafe)`

Checks if the power on a specified port is ready.

`USBPD_StatusTypeDef USBPD_PWR_IF_VBUSEnable(uint8_t PortNum)`

Enables VBUS power on a specified port.

`USBPD_StatusTypeDef USBPD_PWR_IF_VBUSDisable(uint8_t PortNum)`

Disable VBUS/VCONN the power on a specified port

`USBPD_FunctionalState USBPD_PWR_IF_VBUSIsEnabled(uint8_t PortNum)`

Checks if the power on a specified port is enabled

```
USBPD_StatusTypeDef USBPD_PWR_IF_ReadVA(uint8_t PortNum, uint16_t
*pVoltage,

uint16_t *pCurrent)
```

Reads the voltage and the current on a specified port.

```
USBPD_StatusTypeDef USBPD_PWR_IF_Enable_VConn(uint8_t PortNum,
CCxPin_TypeDef CC)
```

Enables the VConn on the port.

```
USBPD_StatusTypeDef USBPD_PWR_IF_Disable_VConn(uint8_t PortNum,
CCxPin_TypeDef CC)
```

Disables the VConn on the port.

3.5 BSP power

3.5.1 Purpose

This module handles all the power aspects for a type C port and must follow the recommendations described in chapter 7 of *[2]*.

3.5.2 API

Refer to *Section 3.2.4*.

3.6 USB-PD core stack

3.6.1 Purpose

The USB-PD core stack library consists of three distinct parts:

- a stack initialization template
- the USB-PD stack in library regrouping the modules Policy engine, Protocol layer and high level of the type C detection state machine
- an optional trace system.

The stack behavior is aligned with *[2]*.

3.6.2 File organization

Table 6. Core stack files

File	Description
usbpd_dpm_core.c/h	Template file to show the stack initialization must be done in the case of an RTOS or non-RTOS system
usbpd_trace.c/h	System of trace
usbpd_def.h	Contains all the type definition
usbpd_tcpm.h	All the interfaces to link a TCPM modules with a TCPC device
usbpd_core.h	USB-PD core stack API

3.6.3 Library description

This section describes the content of each library present in the delivery package, the goal is to help the user in the package selection.

Table 7. Stack variants (Package PD3 = PD2 + PD3 dedicated option)

Library name	SRC	SNK	DRP	VCONN	DATA SWAP	CABLE	VDM	TCPM	PPS	ALERT	CAPA EXT	STATUS	BATTERY	MANU INFO	SECTY MSG	FWUP	COUNTRY
CM0_PD3_FULL	X	X	X	X	X	X	X	-	X	X	X	X	X	X	X	X	X
CM0_PD3_CONFIG_1	X	X	X	X	X	X	-	-	X	X	X	X	X	X	X	X	X
CM0_PD3_CONFIG_MINSRC	X	-	-	-	-	-	-	-	X	X	X	X	X	X	X	X	X
CM0_PD3_CONFIG_MINSNK	-	X	-	-	-	-	-	-	X	X	X	X	X	X	X	X	X
TCPM_CM0_PD3_FULL	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
TCPM_CM0_PD3_CONFIG_1	X	X	X	X	X	X	-	X	-	-	-	-	-	-	-	-	-
CM0PLUS_PD3_FULL	X	X	X	X	X	X	X	-	X	X	X	X	X	X	X	X	X
CM0PLUS_PD3_CONFIG_1	X	X	X	X	X	X	-	-	X	X	X	X	X	X	X	X	X

PD2 libraries will not be supported anymore, as no more PD2 end products can be certified by USB-IF starting from July 25, 2020.

3.6.4 TRACE system

Purpose

This module has been designed to help the debug and the tracking of the software behavior with a real time overview. Proposed Trace system is based on use of an UART (may be restrict to TX UART signal) combined with a software tool to decode the traces.

Initialization

```
void USBPD_TRACE_Init(void)
{
  /* initialize tracer module */
  TRACER_EMB_Init();

  /* Initialize PE trace */
  USBPD_PE_SetTrace(USB-PD_TRACE_Add, 3u);
}
```

The SW is based on the usage of an external component called TRACER_EMB, handling a system of trace through an UART. The function *USB-PD_TRACE_Init* performs two actions:

- calls the function *TRACER_EMB*, which initializes the trace hardware thanks to the configuration file "tracer_emb_conf.h" (see *Appendix C*)
- calls the function *USB-PD_PE_SetTrace* to configure the trace inside the USB-PD stack The enable of the trace system inside our applications is handled by a switch _TRACE. It is strongly recommended to keep the trace enabled for debug purpose.

USBPD_TRACE APIs

```
void USBPD_TRACE_Init(void)
```

Initializes the trace module.

```
void USBPD_TRACE_DeInit(void)
```

De initializes the trace module.

```
void USBPD_TRACE_Add(TRACE_EVENT Type, uint8_t PortNum,
uint8_t Sop, uint8_t *Ptr, uint32_t Size)
```

Adds debug information within the trace buffer

```
uint32_t USBPD_TRACE_TX_Process(void)
```

Processes to send the data on the media.

Interrupt management

Depends upon the selected solutions.

Usage

This USB-PD trace can be used by the end user for two purposes:

- Debug the USB-PD stack behavior (e.g. exchange of messages, notifications, attachment events), Use CubeMonitor-UCPD (see documentation about how to decode the trace)
- Put its own debug trace (see an example of how to add a trace in the code below)

```
void xxxxx_yyyy(uint8_t PortNum, …)
{
 ...
 /* trace function exit */
 USB-PD_TRACE_Add(USB-PD_TRACE_DEBUG, PortNum, 0, "EXIT function
xxxxx_yyyy", 24);
}
```

3.7 USB-PD devices

3.7.1 Purpose

The USB-PD devices correspond to physical layer + type C state machine of the USB-PD specification:

- Type C state machine: SRC, SNK and DRP
- Physical layer: message handling SOP, SOP’, SOP’’, HARDRESET, ….
- Timer server to handle GOODCRC, PRL repetition timing

3.7.2 File organization

Table 8. Device files

File	Description
usbpd_cad_hw_if.c/h	Type C state machine
usbpd_hw.c/h	UCPD hardware settings
usbpd_hw_if_it.c/h	Interrupt handler
usbpd_phy.c/h usbpd_phy_hw_if.c/h	Physical layer
usbpd_pwr_hw_if.c/ h	Power initialization
usbpd_timerserver.c/h	Timer server services
usbpd_device_conf_template	Template file to copy in the applications to define UCPD defines (e.g. UCPD instance)

3.7.3 Settings

The settings of the HW is done via the file usbpd_devices_conf.h, This file has to be derived from the template present in Device include directory (usbpd_devices_conf_template.h) and adapted to user setup). This file is used to:

- Include LL drivers corresponding to the target
- Include the BSP power
- configure UCPD instances
- configure DMA for data transfer
- initialize clock: DMA, DMAMUX
- configure timers.

4 Atomic message sequence

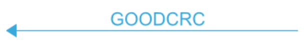
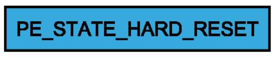
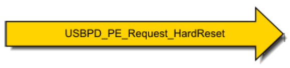
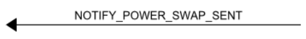
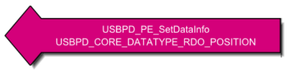
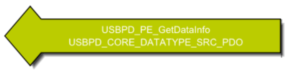
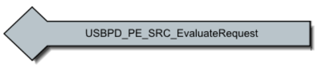
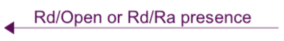
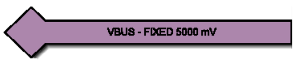
This section provides a detailed and dynamic view of interactions between the stack and the application for each atomic message sequence. The goal of the MSC shown in the next sections is to help the user to develop the application and to understand how to manage the different type C events.

To decode the MSCs the different parts have to be understood as indicated below:

POWER
User Application
USB-PD Core Stack
PHY

- vertical lines describe the time evolution for modules (i.e. state changes, notifications, callbacks)
- the described modules are:
  – POWER: VBUS and VCONN state
  – User Application: the user DPM code
  – USB-PD Core Stack: the protocol stack
  – PHY: physical event message and detection

Table 9. AMS color legend

	USB-PD message exchange
	Internal state of the core state machine
	Request sent by the application to the USB-PD stack
	Notification sent by the stack
	Data transfer from the stack to the application
	Data transfer from the application to the stack
	Callback call
	Type C event
	Power action

Most of AMS sequences described in this document can be reproduced using the trace system (only the callback call does not appear in the trace). The combination of the trace system and the MSC is an indispensable tool to implement a USB-PD solution.

4.1 Connection management

4.1.1 Sequence for a SINK connection

A SINK must present an Rd resistor and detection is done in two steps:

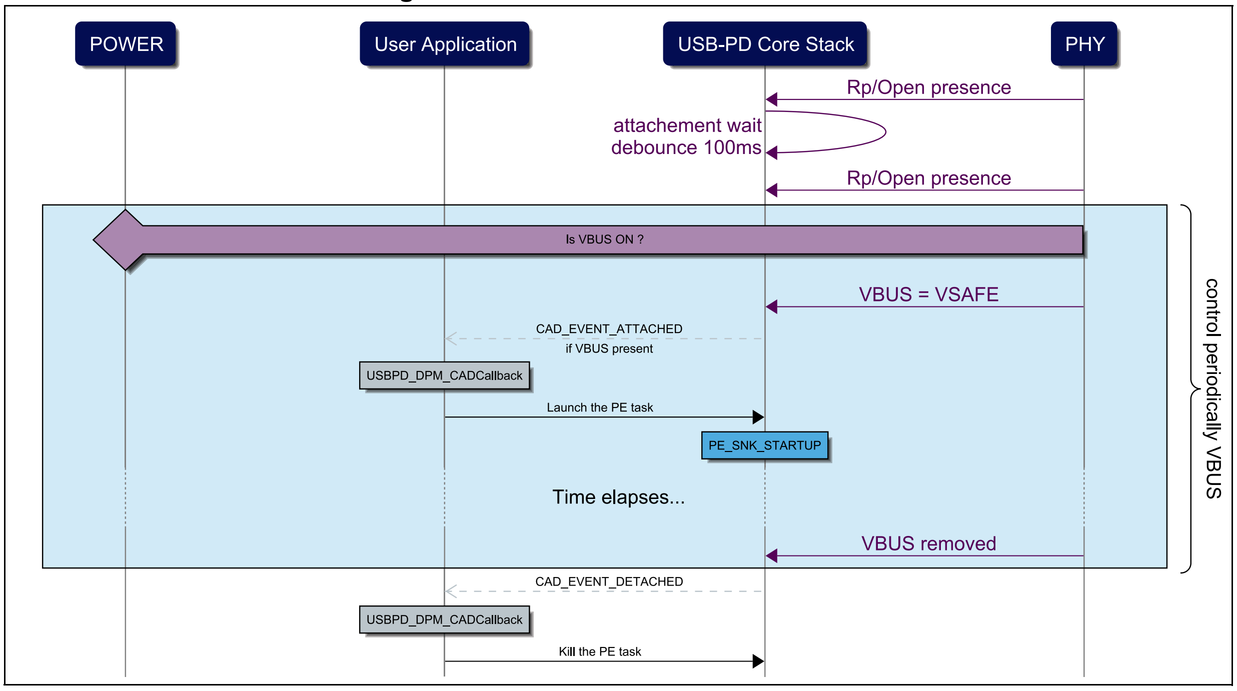
1. detect the presence of Rp resistor
2. detect a VBUS ON.

The attach event triggers the launch of the USB-PD stack (called PE stack), followed by the AMS power negotiation.

Note: The detachment is done on the VBUS removed.

The MSC in *Figure 10* shows the interaction with the user application.

Figure 10. Sink connection and detach



4.1.2 Sequence for a SOURCE connection

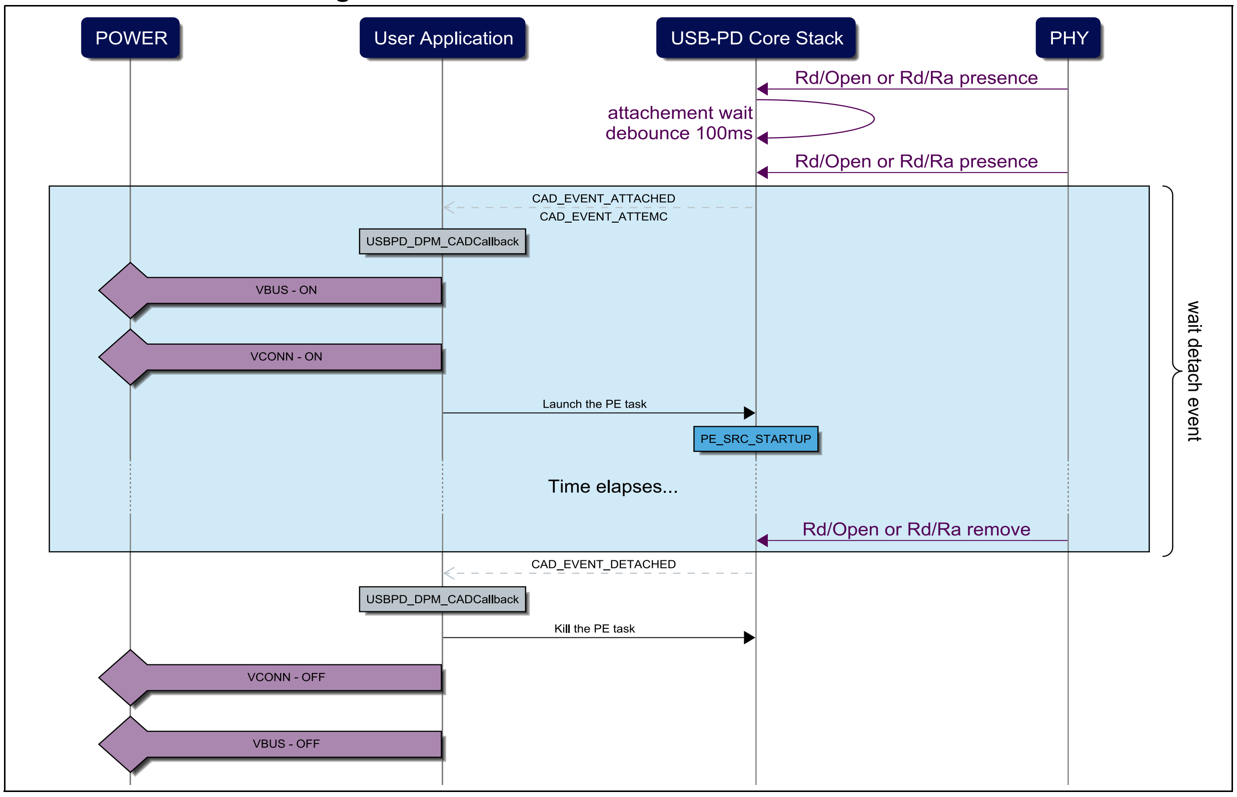
A SRC must present an Rp resistor and detection is done in two steps:

1. detect the presence of Rd resistor
2. wait a debounce time ($t_{CCDebounce}$ at 100 ms) to confirm the attachment.

The attach event triggers an action: the turn on of VBUS and the launch of the USB-PD stack (called PE stack), followed by the AMS power negotiation.

The MSC in *Figure 11* shows the interaction with the user application.

Figure 11. Source connection and disconnection



Note: *VCONN power must be handled only on the event CAD_EVENT_ATTEMC, which corresponds to a detection of an EMC cable and only if the application wants to support VCONN power.*

4.1.3 Sequence for a dual role connection

The dual role port is a merge of the SINK and of the SRC behaviors. The difference is that the port switches between SINK and SRC role when the detection state is Detached. Thus, the MSCs for a SINK and a SRC are valid for DRP port.

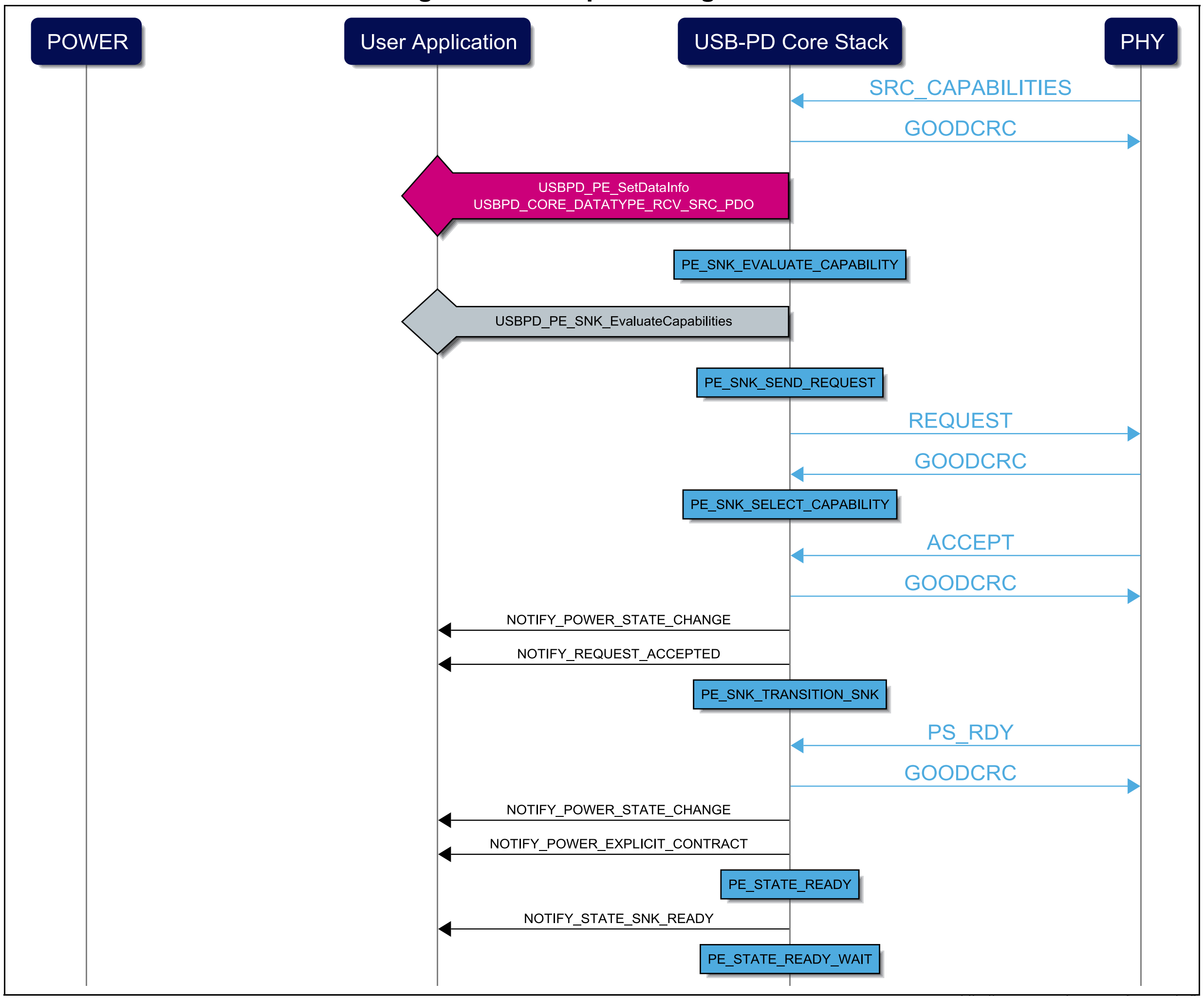
4.2 Sink AMS

This section presents the MCS applicable in the SINK context.

4.2.1 Power negotiation

At the attachment, the PWR available is linked to the detected value of Rp resistor (RpDefault, Rp1.5A or Rp3.0A). The source sends its capabilities and the sink must choose the one it wants to build the request data object (RDO). The RDO information is forwarded to the stack through the call of *USBPD_PE_SNK_EvaluteCapabilities* so the choice is under the responsibility of the application. The MSC in *Figure 12* shows a dynamic view of the application behavior.

Figure 12. Sink power negotiation

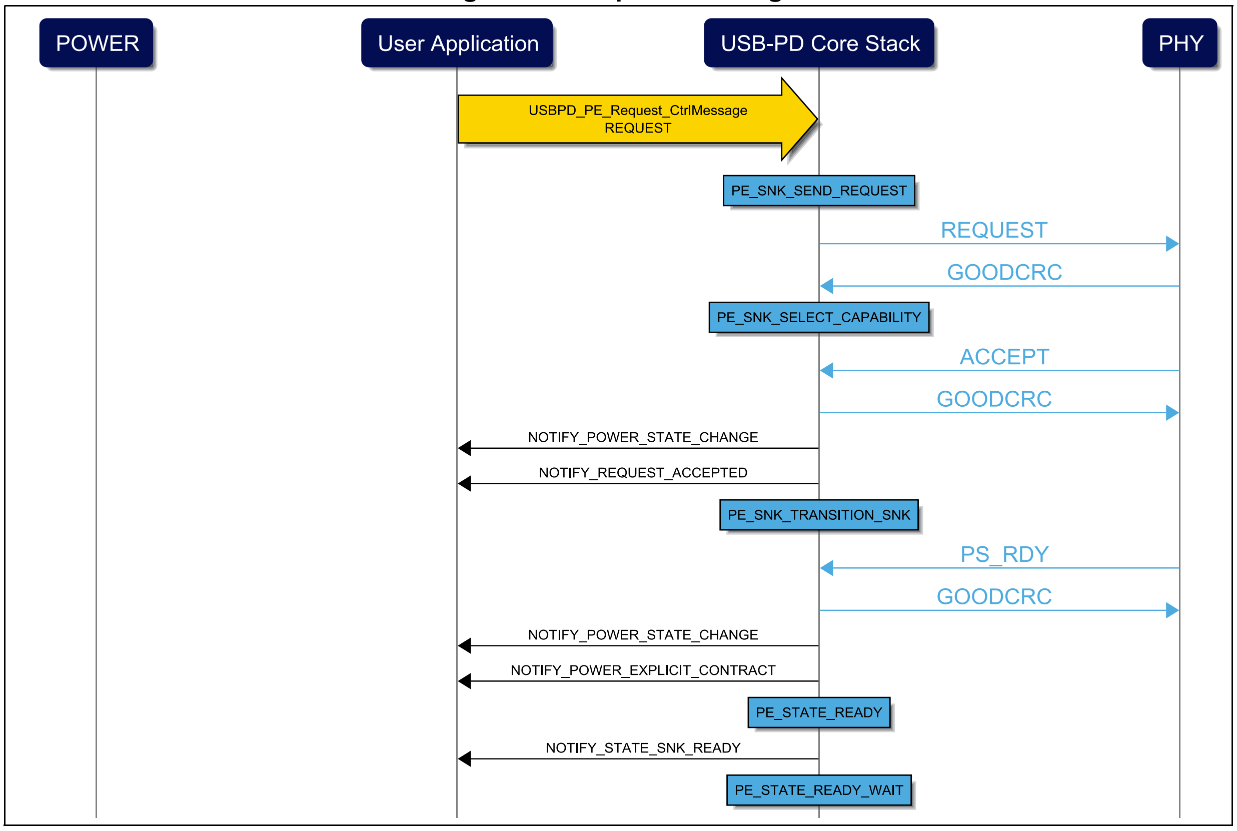


The availability of the new power must be linked with stack notification NOTIFY_POWER_EXPLICIT_CONTRACT.

4.2.2 Send request message

A SINK must send a request message to ask a new power level, the rest of the sequence is identical to AMS power negotiation.

Figure 13. Request message



4.2.3 Power role swap

The PRS consists in switching the power role from sink to source (consumer to provider), or vice-versa. This swap is allowed only if both port partners are DRP-capable. An application knows if its partner is DRP-capable by checking the first PDO (contains some characteristics of the port partner).

The following flowcharts describe the PRS procedure initiated by the sink port (a similar ones, when initiated by source port, can be found in *Section 4.3.3*).

Application sends the PRS

Case 1: PRS has been accepted by the port partner. The sequence shown in *Figure 14* is executed.

Case 2: PRS is rejected by the port partner. The sequence shown in *Figure 15* is executed.

Figure 14. SINK orders a PRS accepted

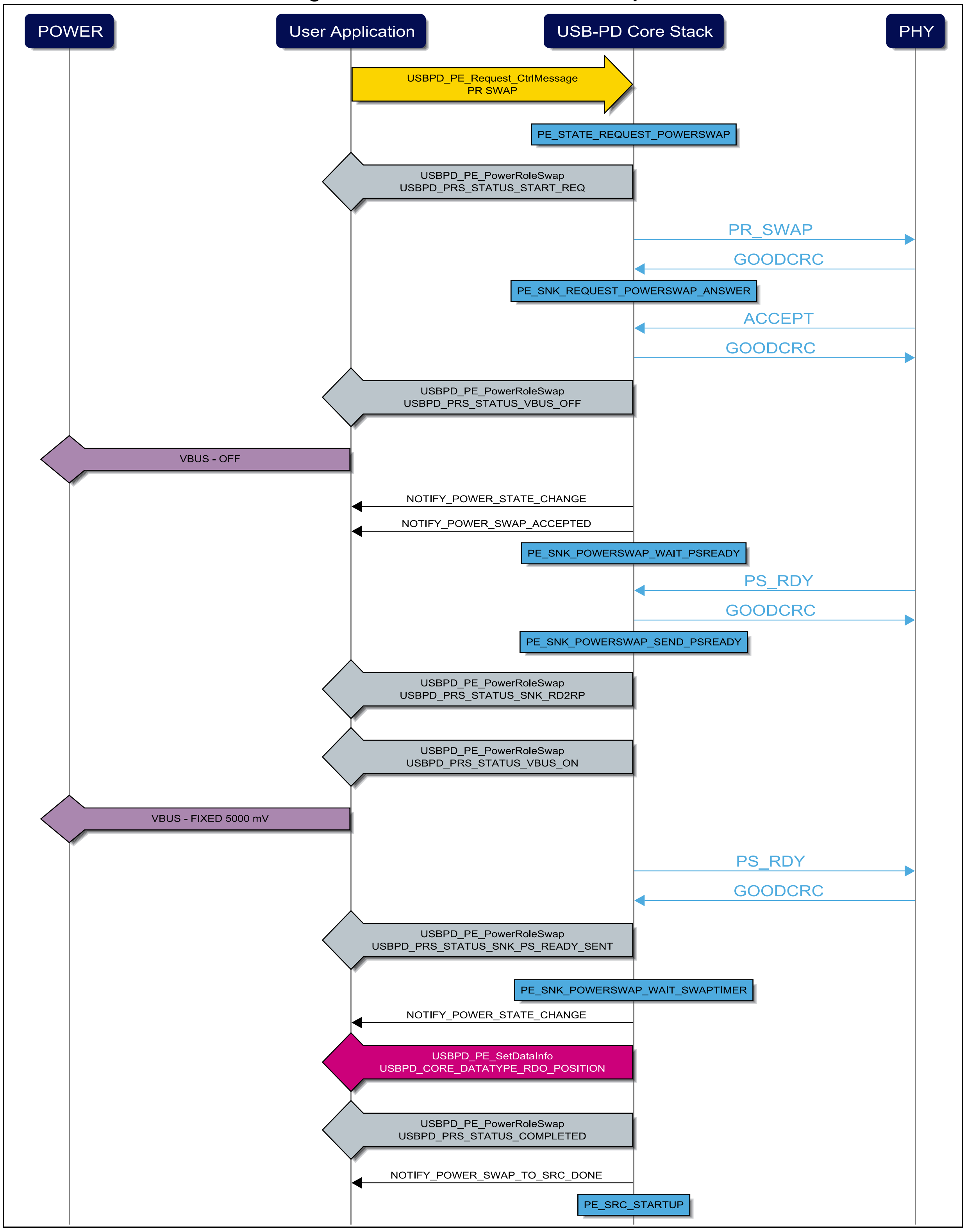
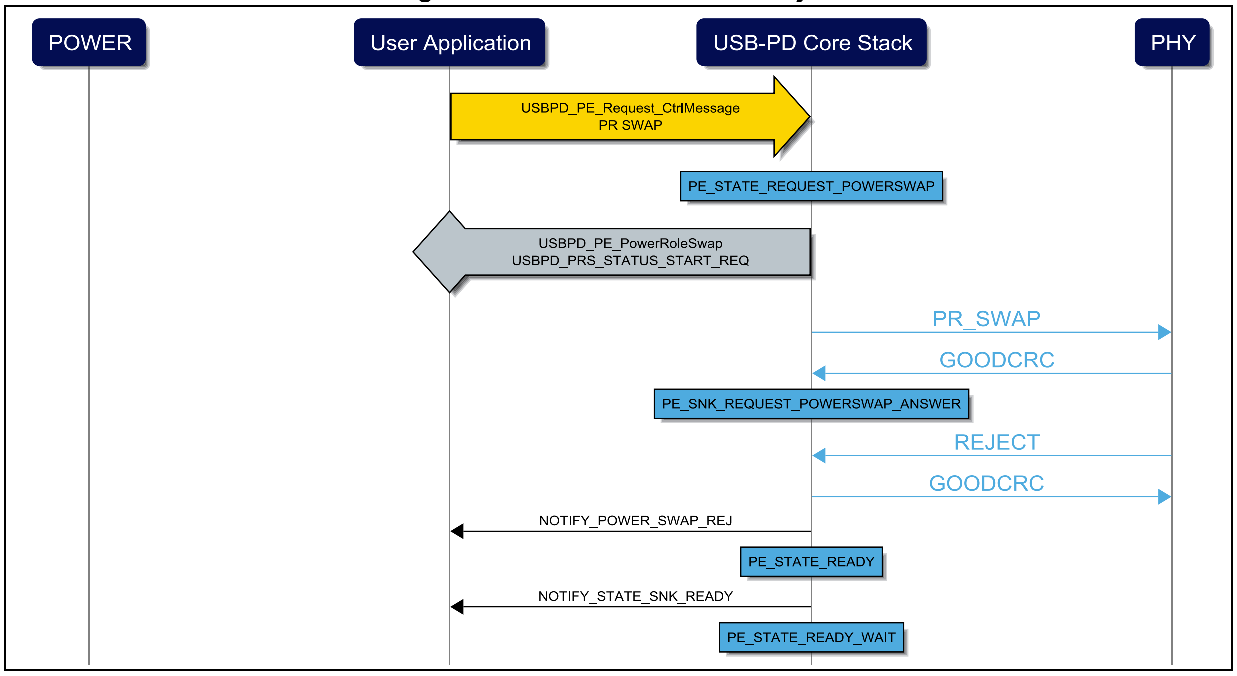

POWER
User Application
USB-PD Core Stack
PHY
USBPD_PE_Request_CtrlMessage
PR SWAP
PE_STATE_REQUEST_POWERSWAP
USBPD_PE_PowerRoleSwap
USBPD_PRS_STATUS_START_REQ
PR_SWAP
GOODCRC
PE_SNK_REQUEST_POWERSWAP_ANSWER
ACCEPT
GOODCRC
USBPD_PE_PowerRoleSwap
USBPD_PRS_STATUS_VBUS_OFF
VBUS - OFF
NOTIFY_POWER_STATE_CHANGE
NOTIFY_POWER_SWAP_ACCEPTED
PE_SNK_POWERSWAP_WAIT_PSREADY
PS_RDY
GOODCRC
PE_SNK_POWERSWAP_SEND_PSREADY
USBPD_PE_PowerRoleSwap
USBPD_PRS_STATUS_SNK_RD2RP
USBPD_PE_PowerRoleSwap
USBPD_PRS_STATUS_VBUS_ON
VBUS - FIXED 5000 mV
PS_RDY
GOODCRC
USBPD_PE_PowerRoleSwap
USBPD_PRS_STATUS_SNK_PS_READY_SENT
PE_SNK_POWERSWAP_WAIT_SWAPTIMER
NOTIFY_POWER_STATE_CHANGE
USBPD_PE_SetDataInfo
USBPD_CORE_DATATYPE_RDO_POSITION
USBPD_PE_PowerRoleSwap
USBPD_PRS_STATUS_COMPLETED
NOTIFY_POWER_SWAP_TO_SRC_DONE
PE_SRC_STARTUP

Figure 15. SINK orders a PRS reject



In the rejection case, the previous power contract is kept between both port partners.

Application receives the PRS

According to the application choice the PRS can be accepted or rejected, this is done by calling the function USB-PD_PE_EvaluatePRSwap, which must return USBBP_ACCEPT or USB-PD_REJECT. The MSCs in *Figure 16* and *Figure 17* show both AMS.

Figure 16. SINK accepts a PRS

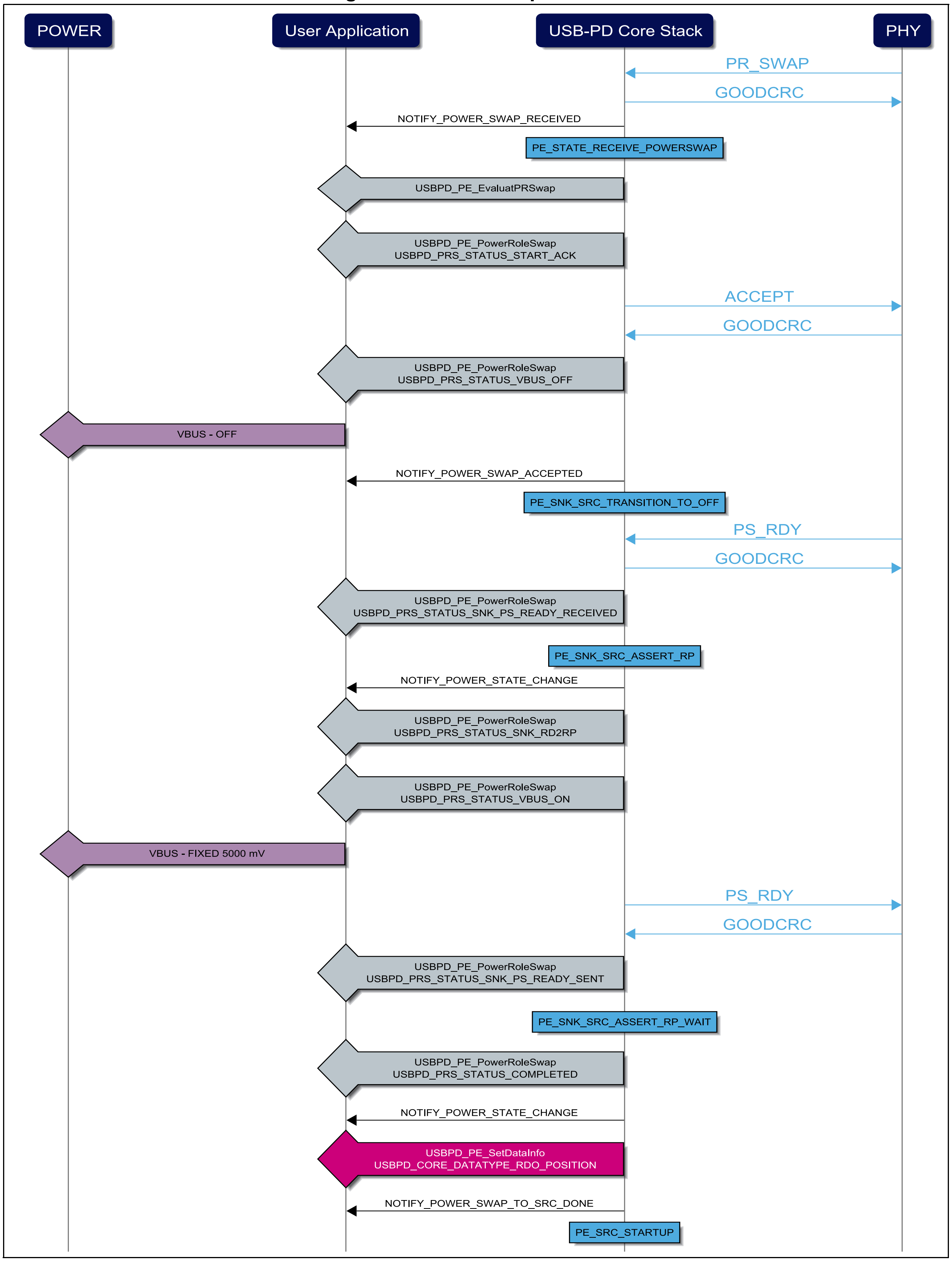
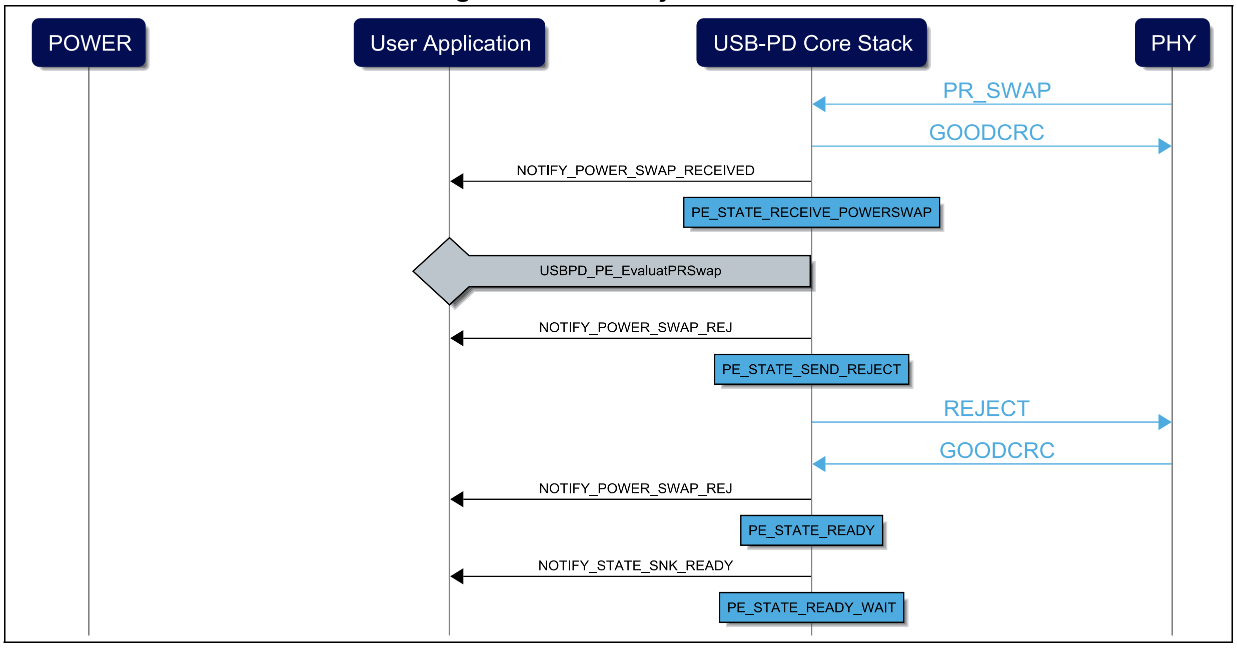

POWER
User Application
USB-PD Core Stack
PHY
PR_SWAP
GOODCRC
NOTIFY_POWER_SWAP_RECEIVED
PE_STATE_RECEIVE_POWERSWAP
USBPD_PE_EvaluatPRSwap
USBPD_PE_PowerRoleSwap USBPD_PRS_STATUS_START_ACK
ACCEPT
GOODCRC
USBPD_PE_PowerRoleSwap USBPD_PRS_STATUS_VBUS_OFF
VBUS - OFF
NOTIFY_POWER_SWAP_ACCEPTED
PE_SNK_SRC_TRANSITION_TO_OFF
PS_RDY
GOODCRC
USBPD_PE_PowerRoleSwap USBPD_PRS_STATUS_SNK_PS_READY_RECEIVED
PE_SNK_SRC_ASSERT_RP
NOTIFY_POWER_STATE_CHANGE
USBPD_PE_PowerRoleSwap USBPD_PRS_STATUS_SNK_RD2RP
USBPD_PE_PowerRoleSwap USBPD_PRS_STATUS_VBUS_ON
VBUS - FIXED 5000 mV
PS_RDY
GOODCRC
USBPD_PE_PowerRoleSwap USBPD_PRS_STATUS_SNK_PS_READY_SENT
PE_SNK_SRC_ASSERT_RP_WAIT
USBPD_PE_PowerRoleSwap USBPD_PRS_STATUS_COMPLETED
NOTIFY_POWER_STATE_CHANGE
USBPD_PE_SetDataInfo USBPD_CORE_DATATYPE_RDO_POSITION
NOTIFY_POWER_SWAP_TO_SRC_DONE
PE_SRC_STARTUP

Figure 17. SINK rejects a PRS



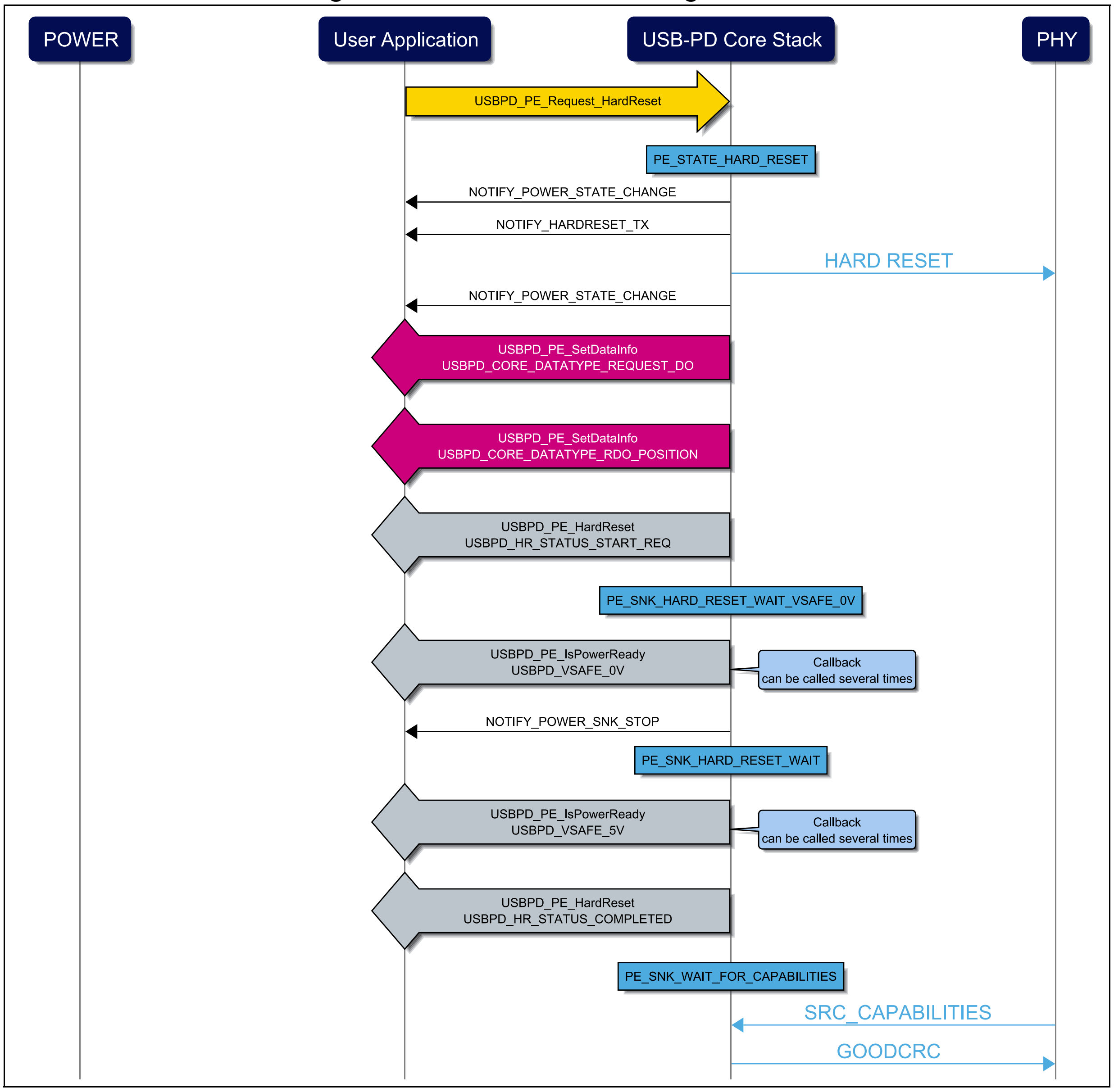
4.2.4 Hard reset

In case of mismatch between partner ports, a message “hard reset” resets the contract. Thus, a direct consequence of this message for sink is to stop the power consumption.

Generate a hard reset

Figure 18 describes the sequence when the application generates a hard reset:

Figure 18. Successful hard reset generation

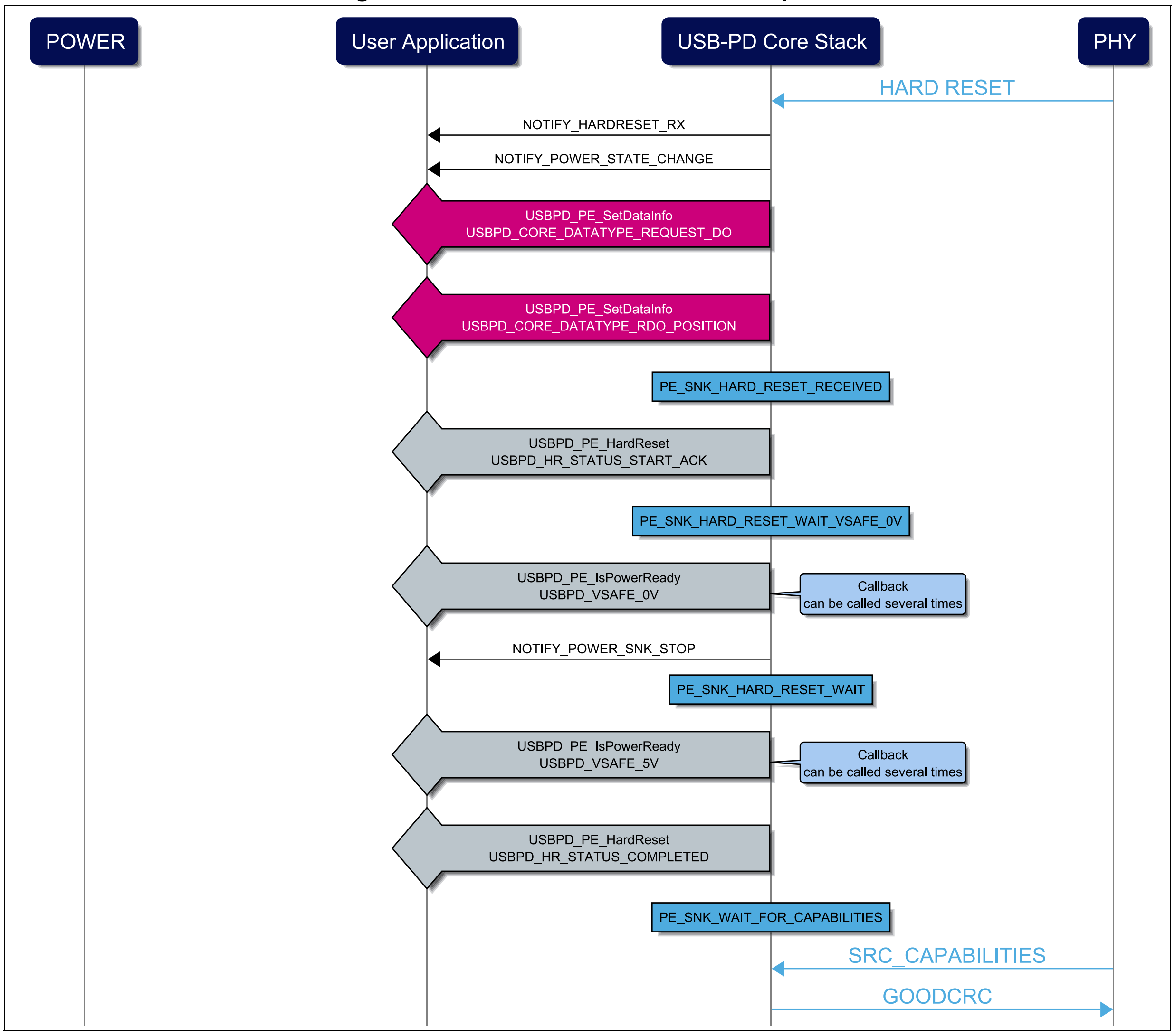

POWER
User Application
USB-PD Core Stack
PHY
USBPD_PE_Request_HardReset
PE_STATE_HARD_RESET
NOTIFY_POWER_STATE_CHANGE
NOTIFY_HARDRESET_TX
HARD RESET
NOTIFY_POWER_STATE_CHANGE
USBPD_PE_SetDataInfo USBPD_CORE_DATATYPE_REQUEST_DO
USBPD_PE_SetDataInfo USBPD_CORE_DATATYPE_RDO_POSITION
USBPD_PE_HardReset USBPD_HR_STATUS_START_REQ
PE_SNK_HARD_RESET_WAIT_VSAFE_0V
USBPD_PE_IsPowerReady USBPD_VSAFE_0V
Callback can be called several times
NOTIFY_POWER_SNK_STOP
PE_SNK_HARD_RESET_WAIT
USBPD_PE_IsPowerReady USBPD_VSAFE_5V
Callback can be called several times
USBPD_PE_HardReset USBPD_HR_STATUS_COMPLETED
PE_SNK_WAIT_FOR_CAPABILITIES
SRC_CAPABILITIES
GOODCRC

If the hard reset message is generated by the USB-PD stack, the previous AMS has to be considered.

Hard reset received by port partner

Figure 19 describes the sequence when the port receives a hard reset.

Figure 19. Successful hard reset reception


POWER
User Application
USB-PD Core Stack
PHY
HARD RESET
NOTIFY_HARDRESET_RX
NOTIFY_POWER_STATE_CHANGE
USBPD_PE_SetDataInfo USBPD_CORE_DATATYPE_REQUEST_DO
USBPD_PE_SetDataInfo USBPD_CORE_DATATYPE_RDO_POSITION
PE_SNK_HARD_RESET_RECEIVED
USBPD_PE_HardReset USBPD_HR_STATUS_START_ACK
PE_SNK_HARD_RESET_WAIT_VSAFE_0V
USBPD_PE_IsPowerReady USBPD_VSAFE_0V
Callback can be called several times
NOTIFY_POWER_SNK_STOP
PE_SNK_HARD_RESET_WAIT
USBPD_PE_IsPowerReady USBPD_VSAFE_5V
Callback can be called several times
USBPD_PE_HardReset USBPD_HR_STATUS_COMPLETED
PE_SNK_WAIT_FOR_CAPABILITIES
SRC_CAPABILITIES
GOODCRC

Fail to reach vSafe0V / vSafe5Vduring hard reset sequence

Figure 20 details the case where there is a problem to reach vSafe0V after tSafe0V ms (set to 650 ms in our stack), and *Figure 21* details the case where there is a problem to reach vSafe5V after (tSrcRecover + tSrcTurnOn) ms, set to (1000 + 275) ms in our stack:

Figure 20. Fail to reach vSafe0V during hard reset sequence in SNK

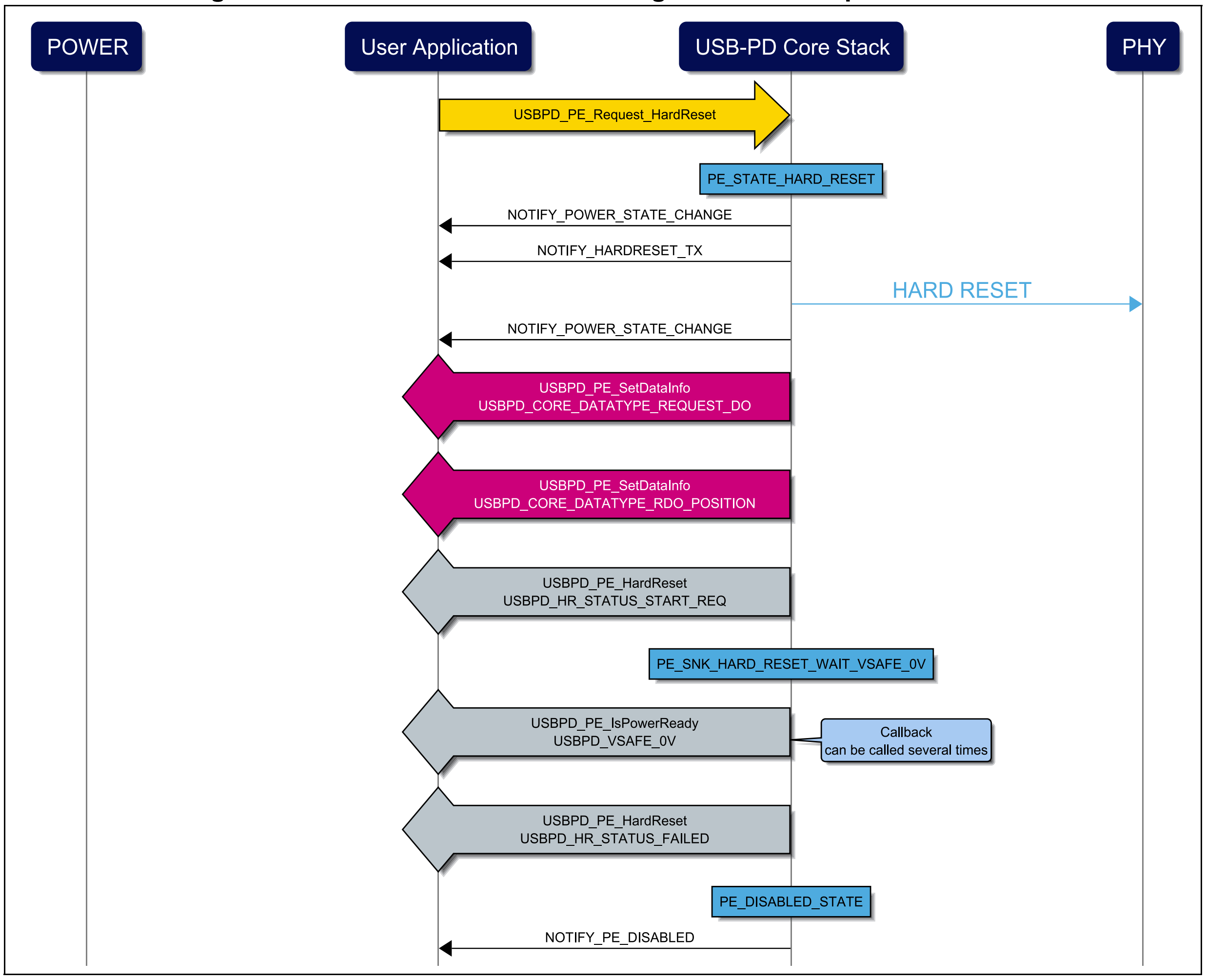
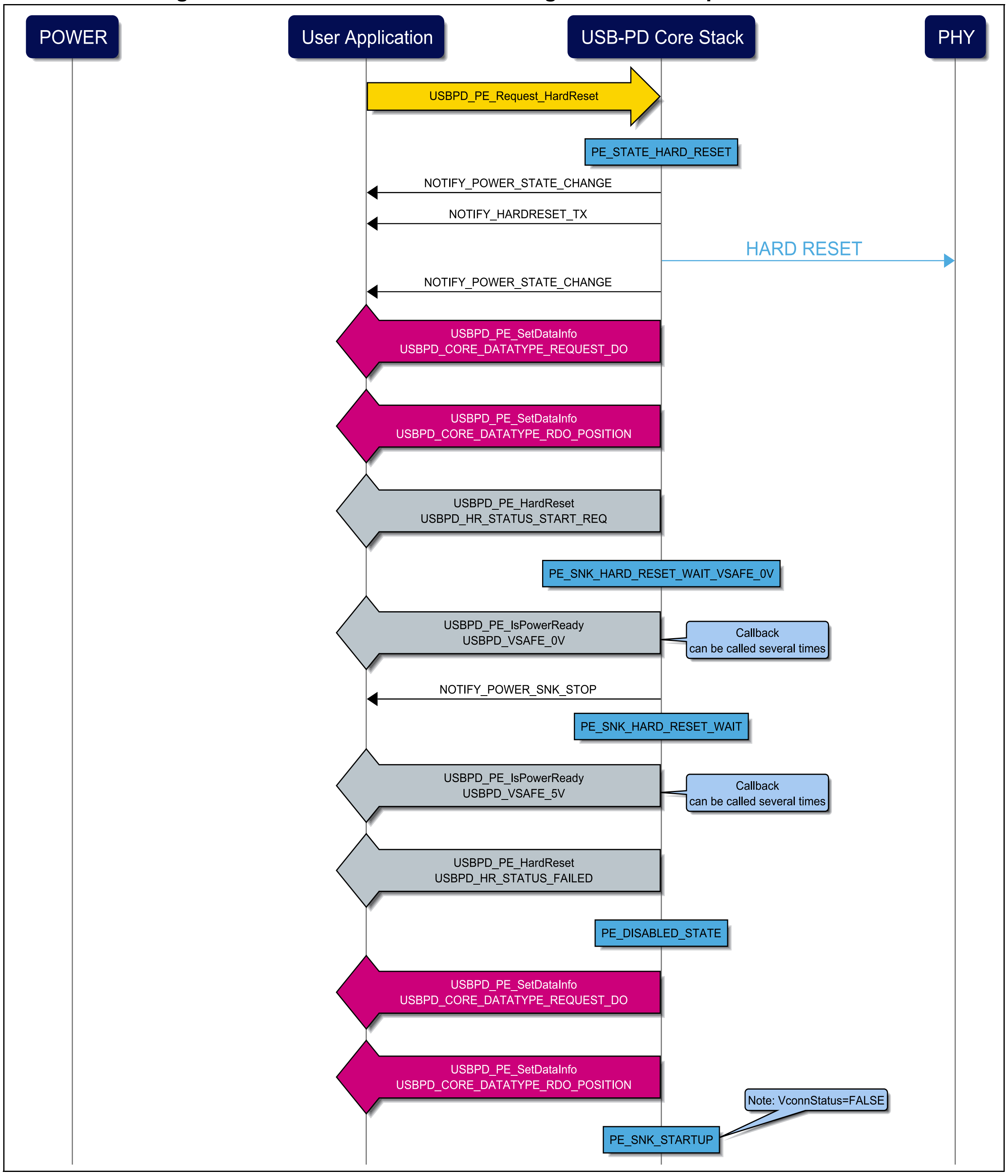


Figure 21. Fail to reach vSafe5V during hard reset sequence in SNK


POWER
User Application
USB-PD Core Stack
PHY
USBPD_PE_Request_HardReset
PE_STATE_HARD_RESET
NOTIFY_POWER_STATE_CHANGE
NOTIFY_HARDRESET_TX
HARD RESET
NOTIFY_POWER_STATE_CHANGE
USBPD_PE_SetDataInfo USBPD_CORE_DATATYPE_REQUEST_DO
USBPD_PE_SetDataInfo USBPD_CORE_DATATYPE_RDO_POSITION
USBPD_PE_HardReset USBPD_HR_STATUS_START_REQ
PE_SNK_HARD_RESET_WAIT_VSAFE_0V
USBPD_PE_IsPowerReady USBPD_VSAFE_0V
Callback can be called several times
NOTIFY_POWER_SNK_STOP
PE_SNK_HARD_RESET_WAIT
USBPD_PE_IsPowerReady USBPD_VSAFE_5V
Callback can be called several times
USBPD_PE_HardReset USBPD_HR_STATUS_FAILED
PE_DISABLED_STATE
USBPD_PE_SetDataInfo USBPD_CORE_DATATYPE_REQUEST_DO
USBPD_PE_SetDataInfo USBPD_CORE_DATATYPE_RDO_POSITION
Note: VconnStatus=FALSE
PE_SNK_STARTUP

4.2.5 VCONN management

The VCONN feature must be supported only if the application wants to discuss with a cable (SOP’ and SOP’’). The USB-PD stack behavior is in relation with VCONN status (ON or OFF), this status is handled through the variable VconnStatus from the structure DPM_Param.

Figure 22 and *Figure 23* show the behavior when VconnStatus = USBPD_FALSE (VCONN OFF) and VconnStatus = USBPD_TRUE (VCONN ON), respectively:

Figure 22. VCONN SWAP with VCONN OFF

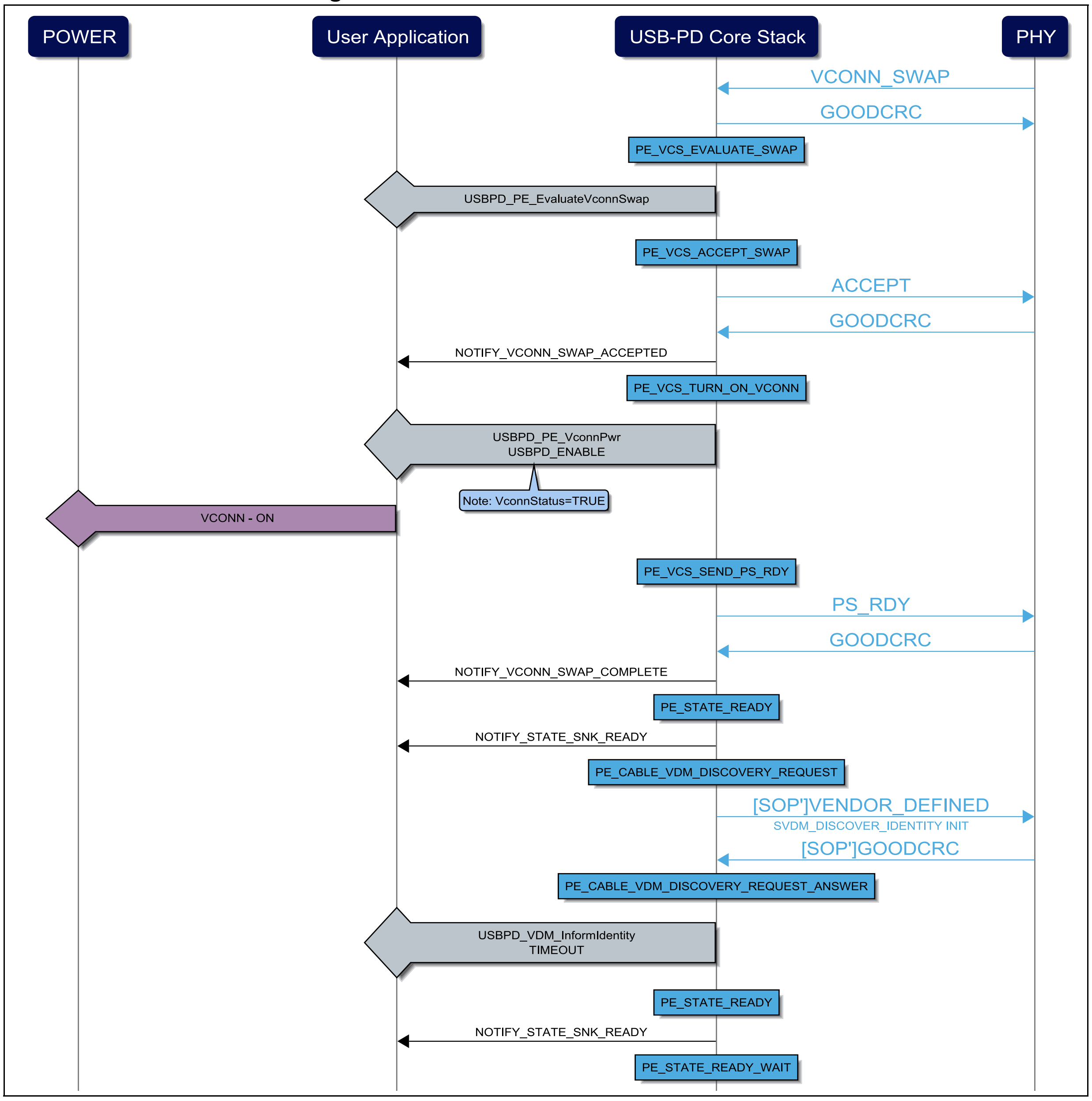
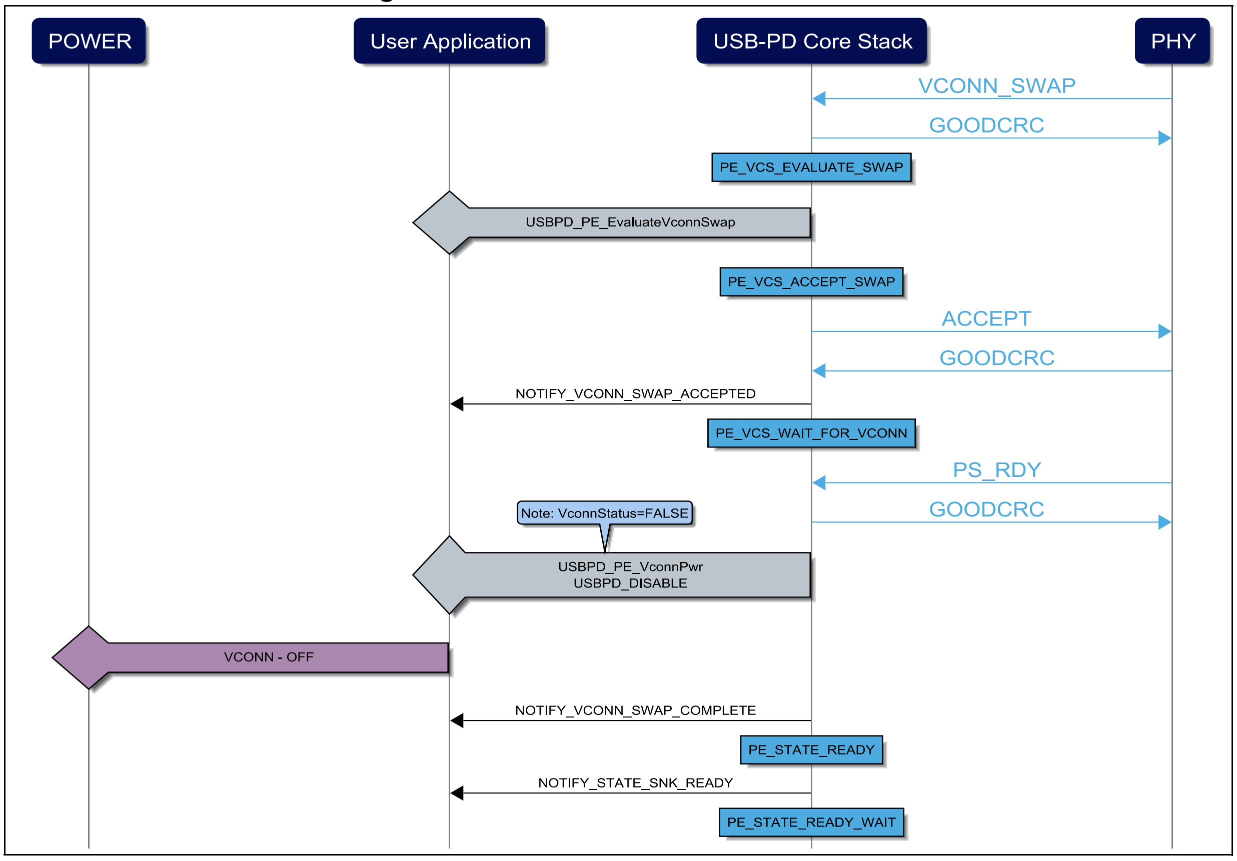


Figure 23. VCONN SWAP with VCONN ON



4.3 Source AMS

This section presents the MSCs applicable in the SOURCE context.

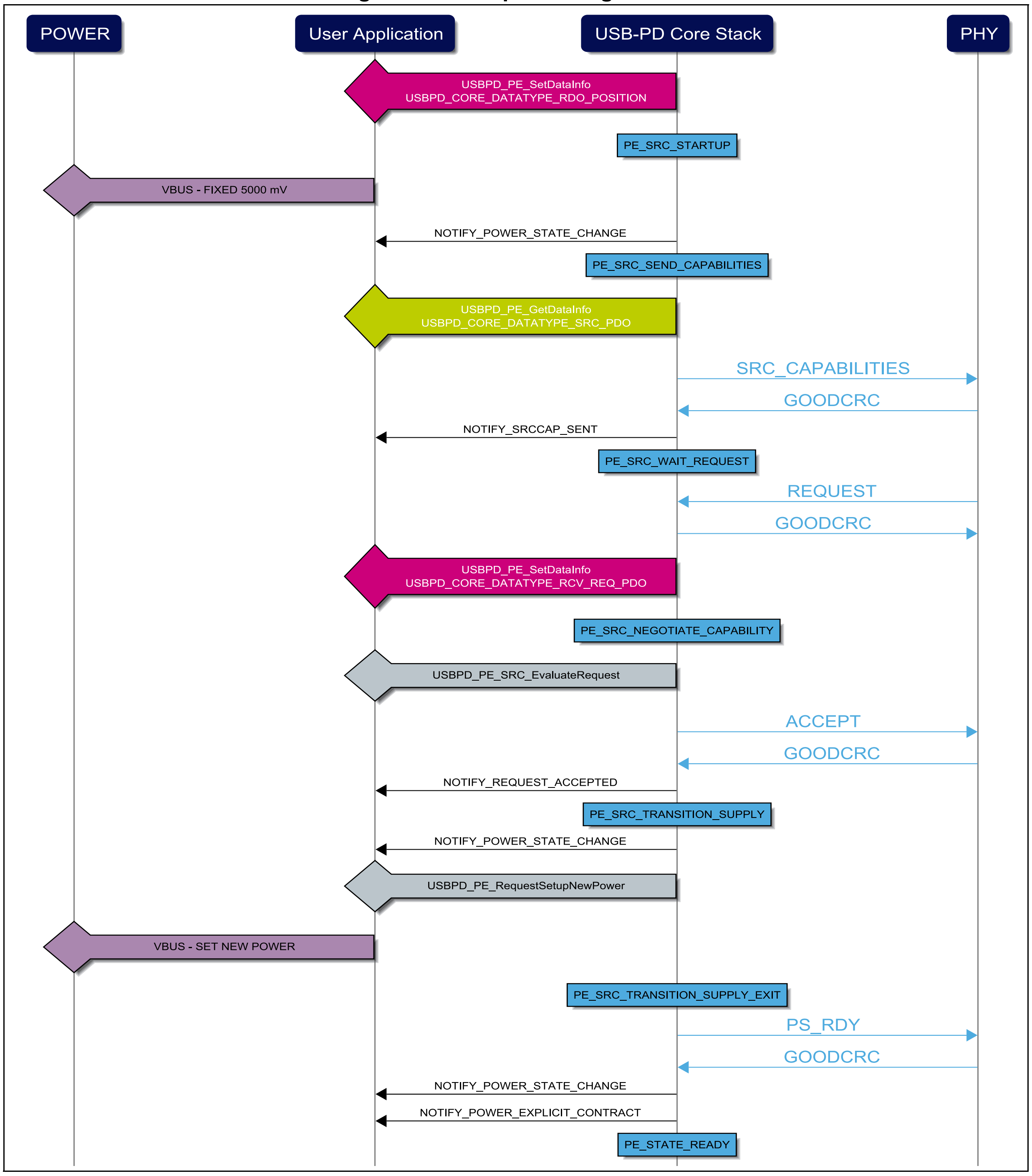
4.3.1 Power negotiation

At the attachment, the source presents a default VBUS (5 V) and sends its capabilities to the port partner to establish an explicit contract. During the power negotiation, the USB-PD stack uses the data sharing mechanism:

- USBPD_CORE_DATATYPE_RDO_POSITION: resets the value to zero
- USBPD_CORE_DATATYPE_SRC_PDO: reads the SRC PDOs
- USBPD_CORE_ DATATYPE_RCV_REQ_PDO: forwards the RDO to the application.

The USB-PD stack calls the function USBPD_PE_SRC_EvaluateRequest for response on demand: USBPD_ACCEPT, USBPD_WAIT, USBPD_REJECT or USBPD_GOTOMIN, and then calls USBPD_PE_RequestSetupNewPower to prepare the power before sending the message PS_READY.

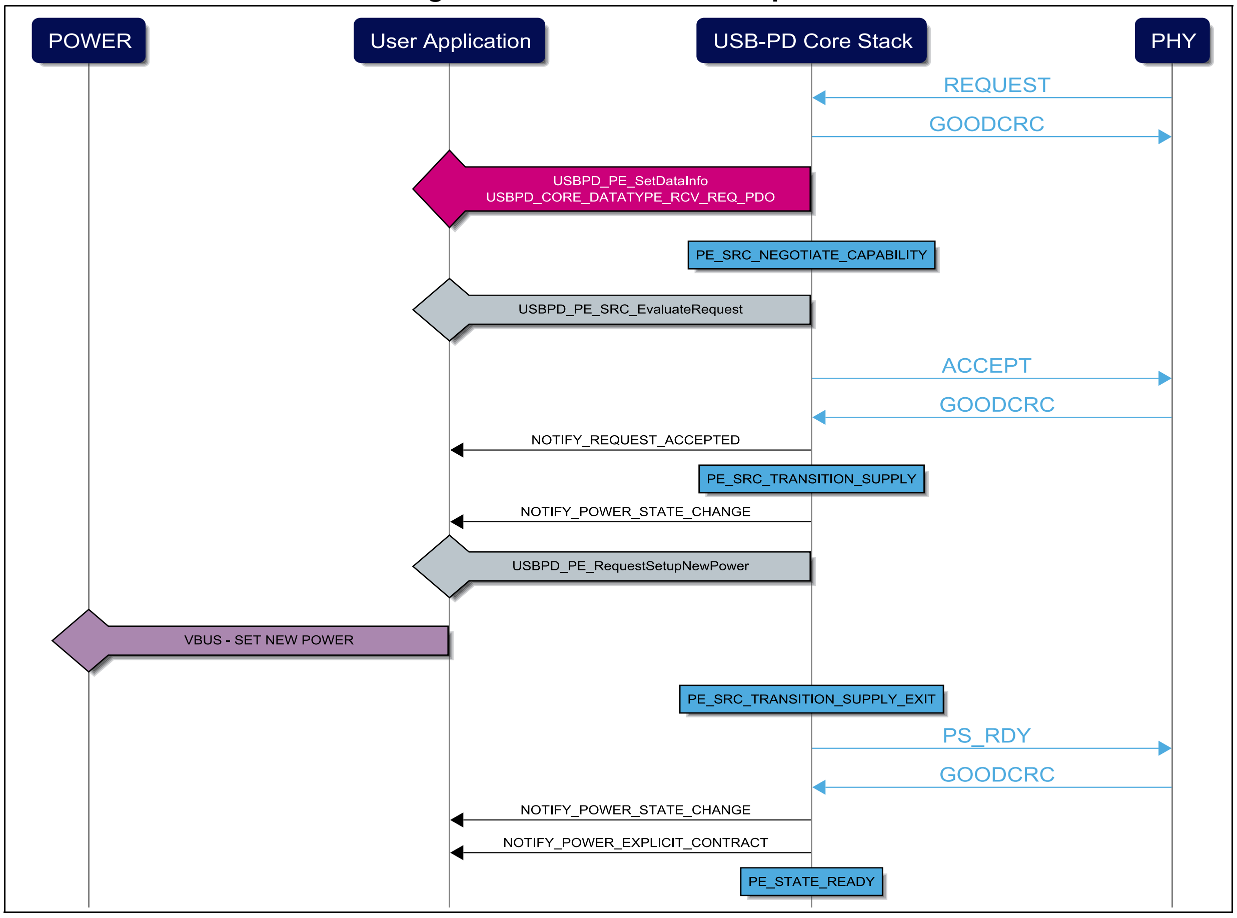
Figure 24. SRC power negotiation



4.3.2 Receive a REQUEST message

The USB-PD stack behavior is identical to the end of the AMS power negotiation.

Figure 25. SRC receives a request


POWER
User Application
USB-PD Core Stack
PHY
REQUEST
GOODCRC
USBPD_PE_SetDataInfo
USBPD_CORE_DATATYPE_RCV_REQ_PDO
PE_SRC_NEGOTIATE_CAPABILITY
USBPD_PE_SRC_EvaluateRequest
ACCEPT
GOODCRC
NOTIFY_REQUEST_ACCEPTED
PE_SRC_TRANSITION_SUPPLY
NOTIFY_POWER_STATE_CHANGE
USBPD_PE_RequestSetupNewPower
VBUS - SET NEW POWER
PE_SRC_TRANSITION_SUPPLY_EXIT
PS_RDY
GOODCRC
NOTIFY_POWER_STATE_CHANGE
NOTIFY_POWER_EXPLICIT_CONTRACT
PE_STATE_READY

4.3.3 Power role swap

The PRS consists in switching the power role from a source to a sink (provider to consumer) or vice-versa. This switch is only allowed if both port partners are DRP-capable. An application can know if its partner is DRP-capable by requesting the sink capabilities and checking the first PDO (contains some characteristics of the port partner).

The following flowcharts describe the PRS procedure initiated by the sink port (a similar one, when initiated by source port, can be found in *Section 4.2.3*).

Application sends the PRS

If the PRS is accepted by the port partner, the sequence in *Figure 26* is executed. At the end, the policy engine restarts with a sink and waits for the source capabilities from distant port.

Note: The VCONN power is not affected by a PRS so the port continues to power the VCONN.

The AMS with rejection is shown in *Figure 27*.

Figure 26. SRC PRS accepted

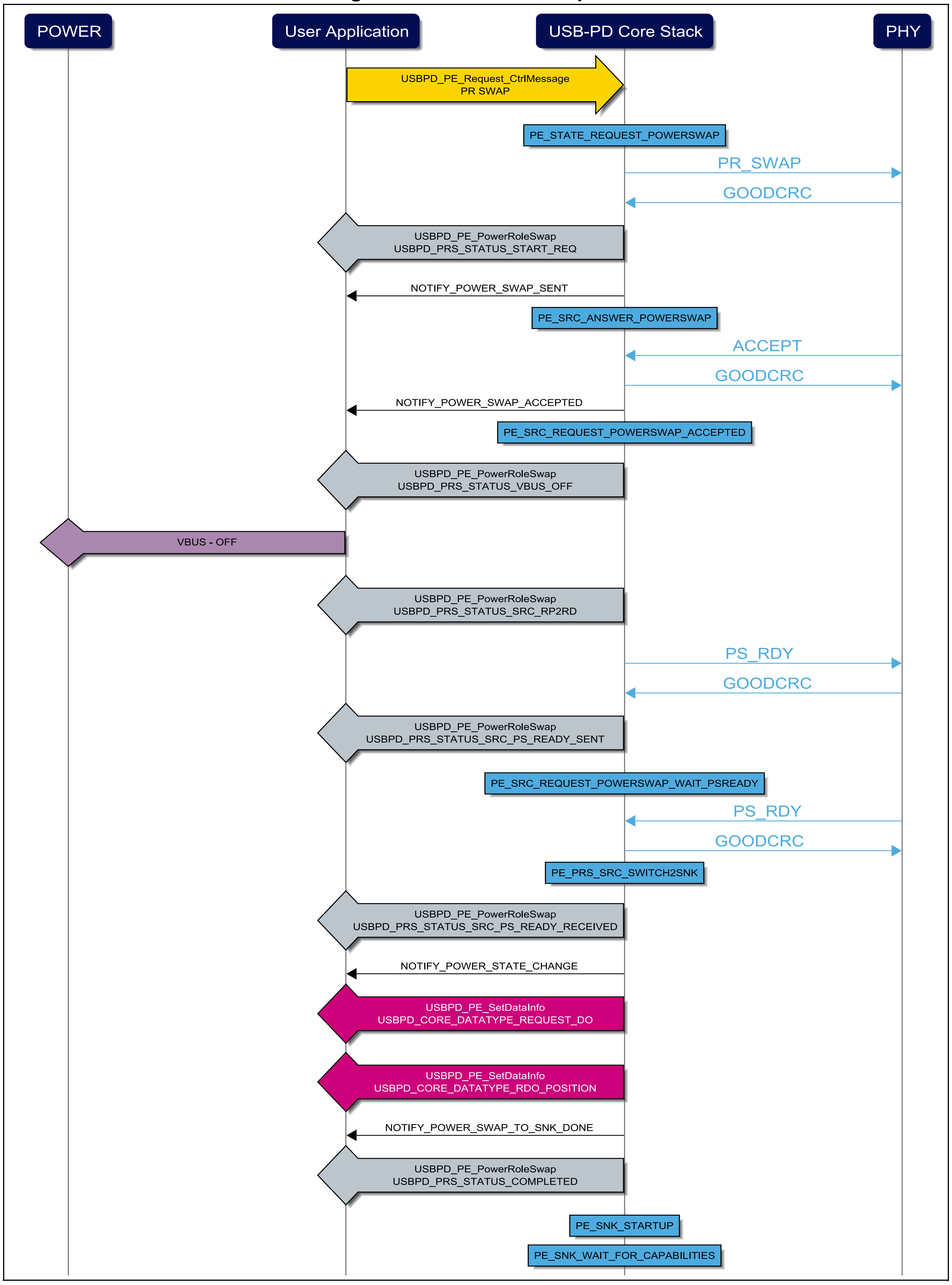
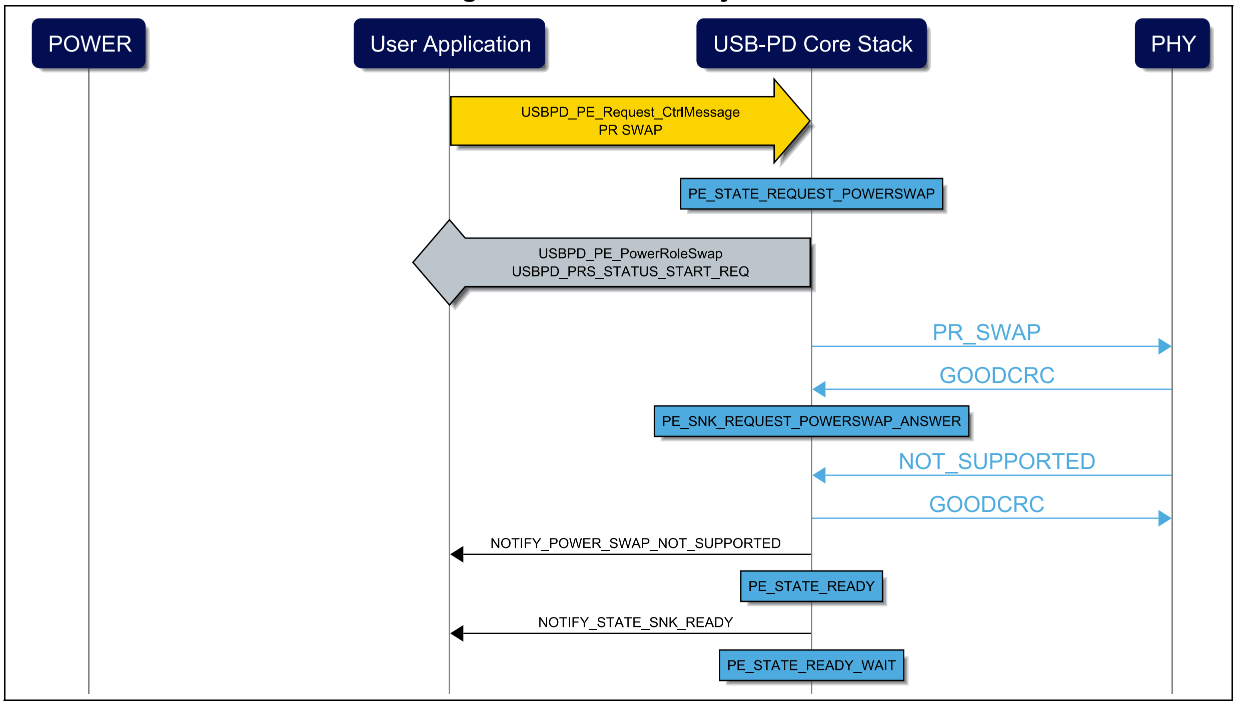

POWER
User Application
USB-PD Core Stack
PHY
USBPD_PE_Request_CtrlMessage PR SWAP
PE_STATE_REQUEST_POWERSWAP
PR_SWAP
GOODCRC
USBPD_PE_PowerRoleSwap USBPD_PRS_STATUS_START_REQ
NOTIFY_POWER_SWAP_SENT
PE_SRC_ANSWER_POWERSWAP
ACCEPT
GOODCRC
NOTIFY_POWER_SWAP_ACCEPTED
PE_SRC_REQUEST_POWERSWAP_ACCEPTED
USBPD_PE_PowerRoleSwap USBPD_PRS_STATUS_VBUS_OFF
VBUS - OFF
USBPD_PE_PowerRoleSwap USBPD_PRS_STATUS_SRC_RP2RD
PS_RDY
GOODCRC
USBPD_PE_PowerRoleSwap USBPD_PRS_STATUS_SRC_PS_READY_SENT
PE_SRC_REQUEST_POWERSWAP_WAIT_PSREADY
PS_RDY
GOODCRC
PE_PRS_SRC_SWITCH2SNK
USBPD_PE_PowerRoleSwap USBPD_PRS_STATUS_SRC_PS_READY_RECEIVED
NOTIFY_POWER_STATE_CHANGE
USBPD_PE_SetDataInfo USBPD_CORE_DATATYPE_REQUEST_DO
USBPD_PE_SetDataInfo USBPD_CORE_DATATYPE_RDO_POSITION
NOTIFY_POWER_SWAP_TO_SNK_DONE
USBPD_PE_PowerRoleSwap USBPD_PRS_STATUS_COMPLETED
PE_SNK_STARTUP
PE_SNK_WAIT_FOR_CAPABILITIES

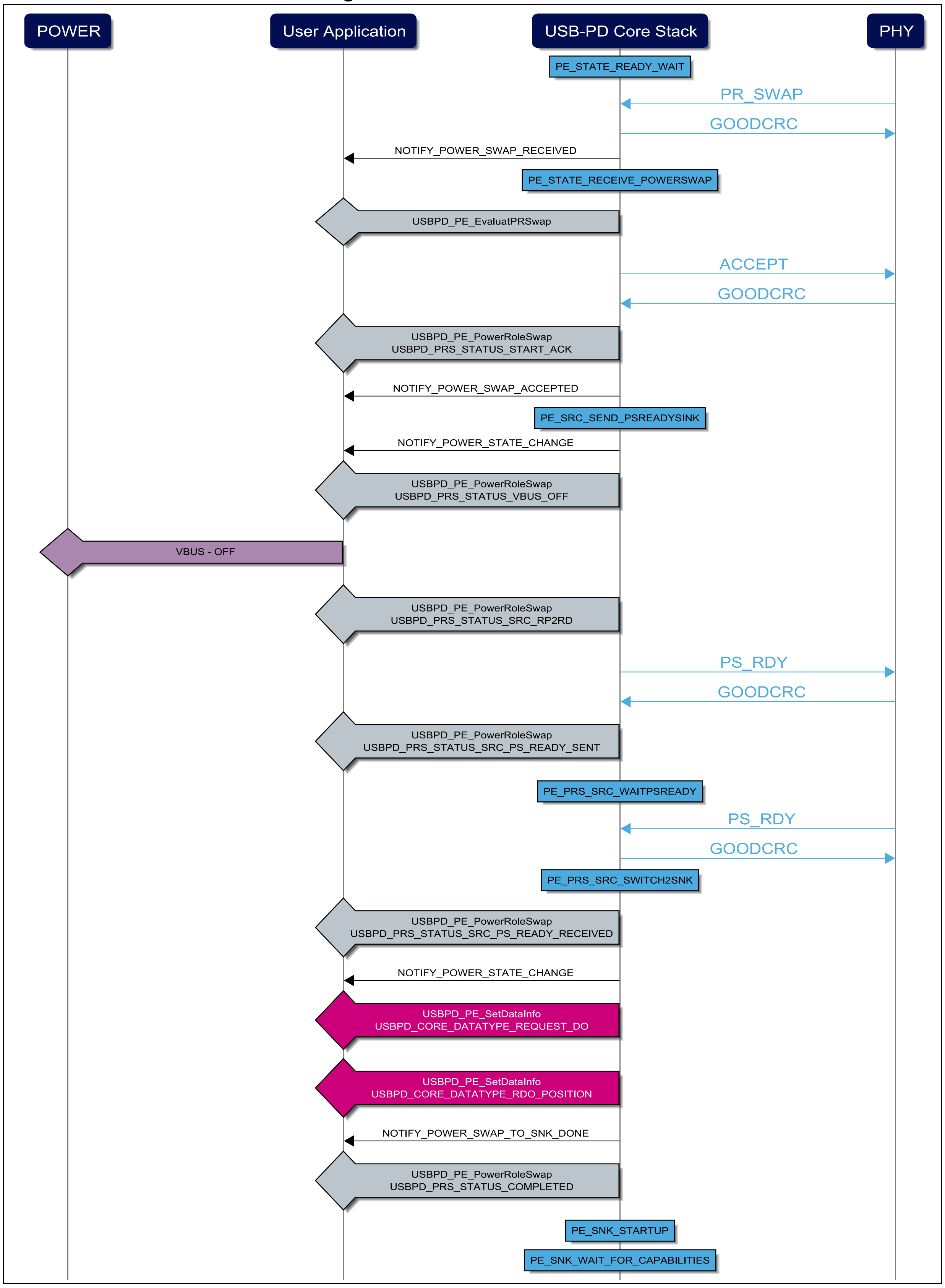
Figure 27. SRC PRS rejected



Application receives a PRS

According to the application choice, the PRS can be accepted or rejected, this is done by calling the function *USB-PD_PE_EvaluatePRSwap()*, which must return USBBP_ACCEPT or USBPD_REJECT. *Figure 28* shows the acceptance case.

Figure 28. SRC receives a PRS

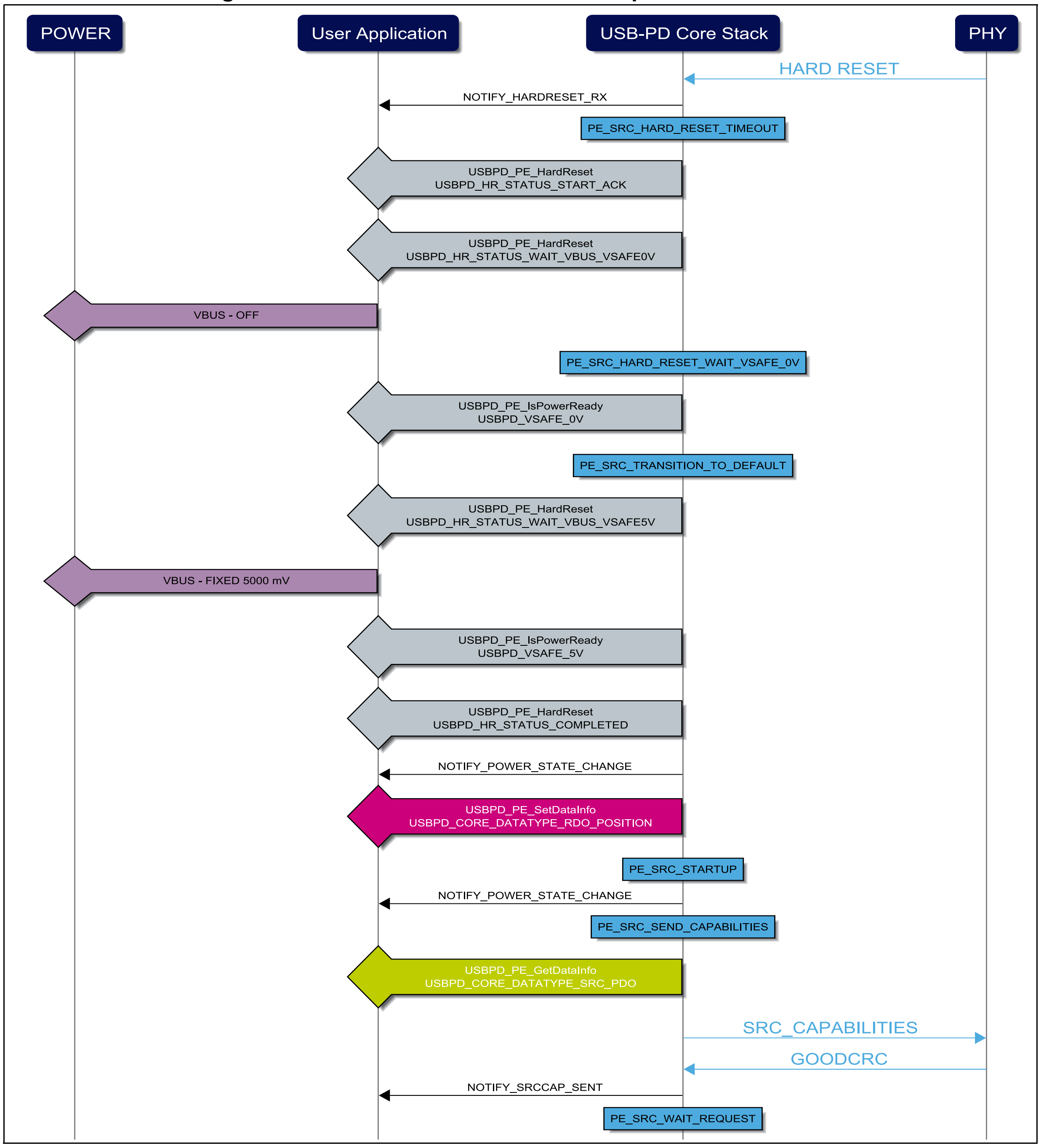

POWER
User Application
USB-PD Core Stack
PHY
PE_STATE_READY_WAIT
PR_SWAP
GOODCRC
NOTIFY_POWER_SWAP_RECEIVED
PE_STATE_RECEIVE_POWERSWAP
USBPD_PE_EvaluatPRSwap
ACCEPT
GOODCRC
USBPD_PE_PowerRoleSwap USBPD_PRS_STATUS_START_ACK
NOTIFY_POWER_SWAP_ACCEPTED
PE_SRC_SEND_PSREADYSINK
NOTIFY_POWER_STATE_CHANGE
USBPD_PE_PowerRoleSwap USBPD_PRS_STATUS_VBUS_OFF
VBUS - OFF
USBPD_PE_PowerRoleSwap USBPD_PRS_STATUS_SRC_RP2RD
PS_RDY
GOODCRC
USBPD_PE_PowerRoleSwap USBPD_PRS_STATUS_SRC_PS_READY_SENT
PE_PRS_SRC_WAITPSREADY
PS_RDY
GOODCRC
PE_PRS_SRC_SWITCH2SNK
USBPD_PE_PowerRoleSwap USBPD_PRS_STATUS_SRC_PS_READY_RECEIVED
NOTIFY_POWER_STATE_CHANGE
USBPD_PE_SetDataInfo USBPD_CORE_DATATYPE_REQUEST_DO
USBPD_PE_SetDataInfo USBPD_CORE_DATATYPE_RDO_POSITION
NOTIFY_POWER_SWAP_TO_SNK_DONE
USBPD_PE_PowerRoleSwap USBPD_PRS_STATUS_COMPLETED
PE_SNK_STARTUP
PE_SNK_WAIT_FOR_CAPABILITIES

4.3.4 Hard reset

Compared to sink hard reset, there is only one difference regarding power. During a hard reset AMS, the source must disable to power (VCONN and VBUS) during tSrcRecover (660 to 1000 ms).

The stack implements the different timings using during hard reset sequence defined in the USB-PD specification.

Figure 29. Source VBUS and VCONN response to hard reset


POWER
User Application
USB-PD Core Stack
PHY
HARD RESET
NOTIFY_HARDRESET_RX
PE_SRC_HARD_RESET_TIMEOUT
USBPD_PE_HardReset USBPD_HR_STATUS_START_ACK
USBPD_PE_HardReset USBPD_HR_STATUS_WAIT_VBUS_VSAFE0V
VBUS - OFF
PE_SRC_HARD_RESET_WAIT_VSAFE_0V
USBPD_PE_IsPowerReady USBPD_VSAFE_0V
PE_SRC_TRANSITION_TO_DEFAULT
USBPD_PE_HardReset USBPD_HR_STATUS_WAIT_VBUS_VSAFE5V
VBUS - FIXED 5000 mV
USBPD_PE_IsPowerReady USBPD_VSAFE_5V
USBPD_PE_HardReset USBPD_HR_STATUS_COMPLETED
NOTIFY_POWER_STATE_CHANGE
USBPD_PE_SetDataInfo USBPD_CORE_DATATYPE_RDO_POSITION
PE_SRC_STARTUP
NOTIFY_POWER_STATE_CHANGE
PE_SRC_SEND_CAPABILITIES
USBPD_PE_GetDataInfo USBPD_CORE_DATATYPE_SRC_PDO
SRC_CAPABILITIES
GOODCRC
NOTIFY_SRCCAP_SENT
PE_SRC_WAIT_REQUEST

Timing values follow those defined in the USB-PD specification.

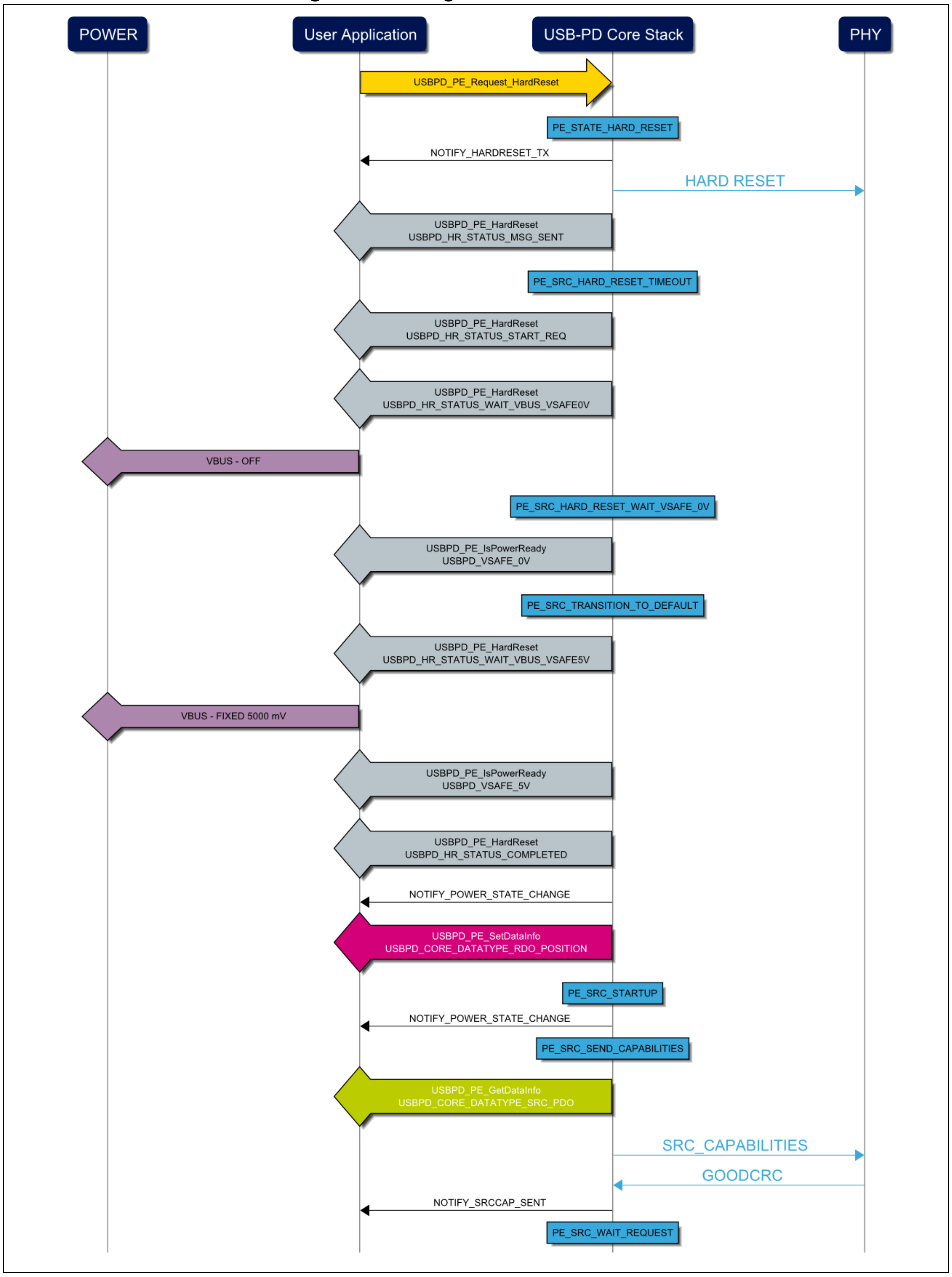
Table 10. Timing values from specification

Parameter	Description	Min	Max
tSafe0V	Time to reach vSafe0V max	-	650 ms
tSafe5V	Time to reach vSafe5V max	-	275 ms
tSrcRecover	Time allotted for the Source to recover	660 ms	1000 ms
tSrcTurnOn	Transition time from vSafe0V to vSafe5V	-	275 ms

Hard reset received by port partner

Figure 30 details the hard reset sending case.

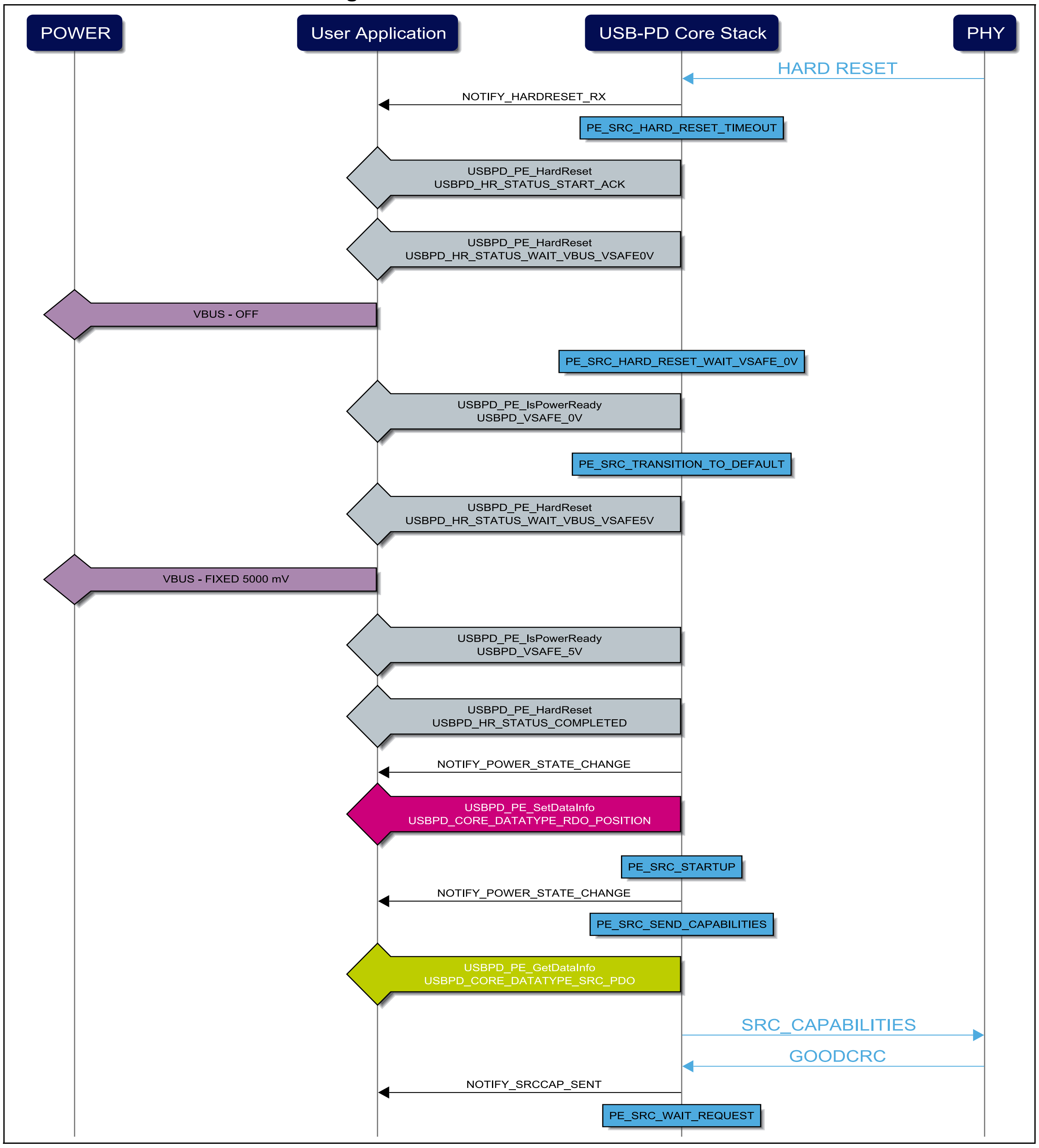
Figure 30. SRC generates a hard reset


POWER
User Application
USB-PD Core Stack
PHY
USBPD_PE_Request_HardReset
PE_STATE_HARD_RESET
NOTIFY_HARDRESET_TX
HARD RESET
USBPD_PE_HardReset USBPD_HR_STATUS_MSG_SENT
PE_SRC_HARD_RESET_TIMEOUT
USBPD_PE_HardReset USBPD_HR_STATUS_START_REQ
USBPD_PE_HardReset USBPD_HR_STATUS_WAIT_VBUS_VSAFE0V
VBUS - OFF
PE_SRC_HARD_RESET_WAIT_VSAFE_0V
USBPD_PE_IsPowerReady USBPD_VSAFE_0V
PE_SRC_TRANSITION_TO_DEFAULT
USBPD_PE_HardReset USBPD_HR_STATUS_WAIT_VBUS_VSAFE5V
VBUS - FIXED 5000 mV
USBPD_PE_IsPowerReady USBPD_VSAFE_5V
USBPD_PE_HardReset USBPD_HR_STATUS_COMPLETED
NOTIFY_POWER_STATE_CHANGE
USBPD_PE_SetDataInfo USBPD_CORE_DATATYPE_RDO_POSITION
PE_SRC_STARTUP
NOTIFY_POWER_STATE_CHANGE
PE_SRC_SEND_CAPABILITIES
USBPD_PE_GetDataInfo USBPD_CORE_DATATYPE_SRC_PDO
SRC_CAPABILITIES
GOODCRC
NOTIFY_SRCCAP_SENT
PE_SRC_WAIT_REQUEST

Generate a hard reset

Figure 31 describes the hard reset reception case.

Figure 31. SRC receives a hard reset



Fail to reach vSafe0V / vSafe5V during hard reset sequence

The MSC in *Figure 32* and *Figure 33* describe, respectively, the case where there is a problem to reach vSafe0V after tSafe0V ms (set to 650 ms in our stack) and the case where there is a problem to reach vSafe5V after (tSrcRecoverMin + tSrcTurnOn) ms, set to (800 + 275) ms in our stack.

Figure 32. Fail to reach vSafe0V during hard reset sequence in SRC

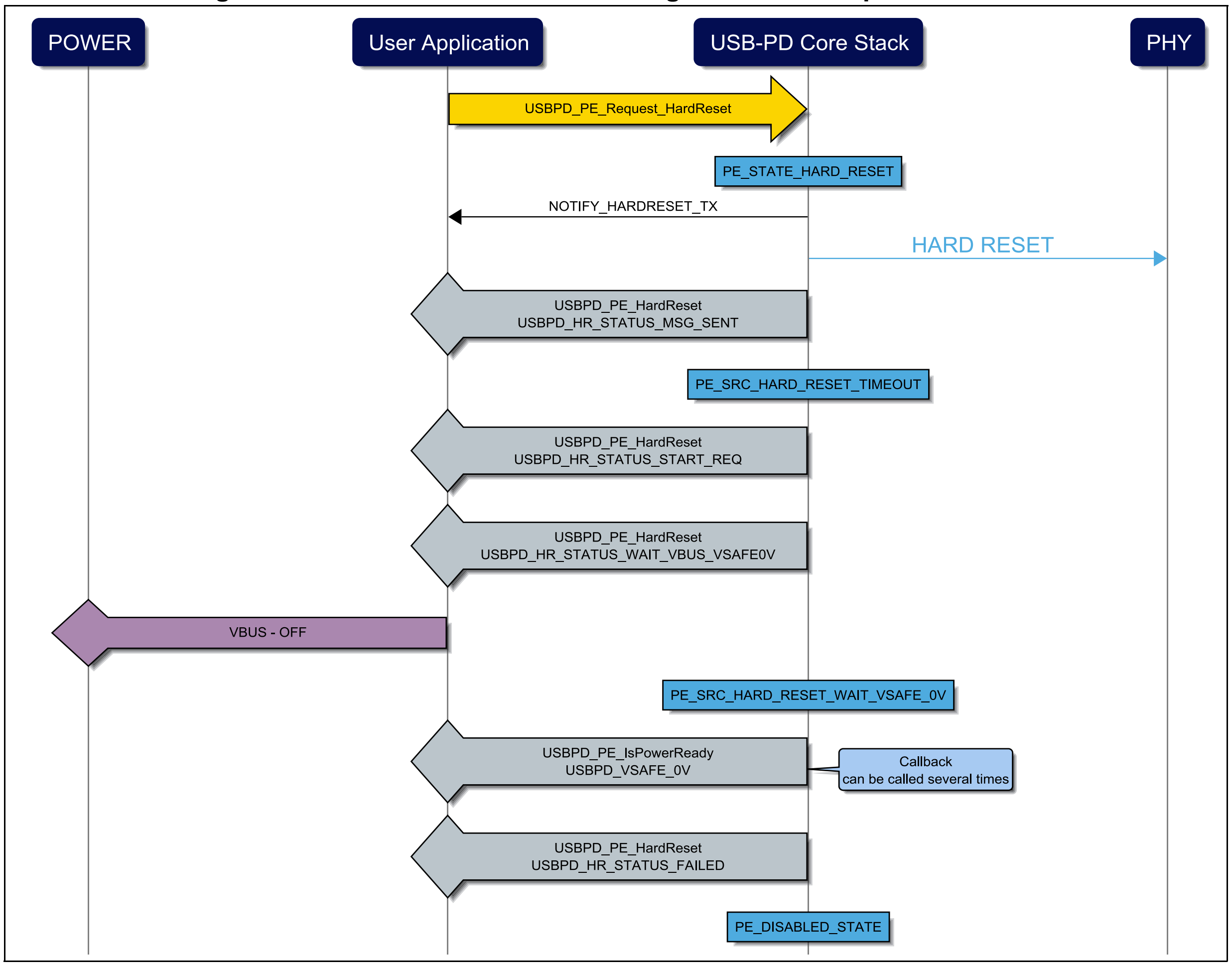
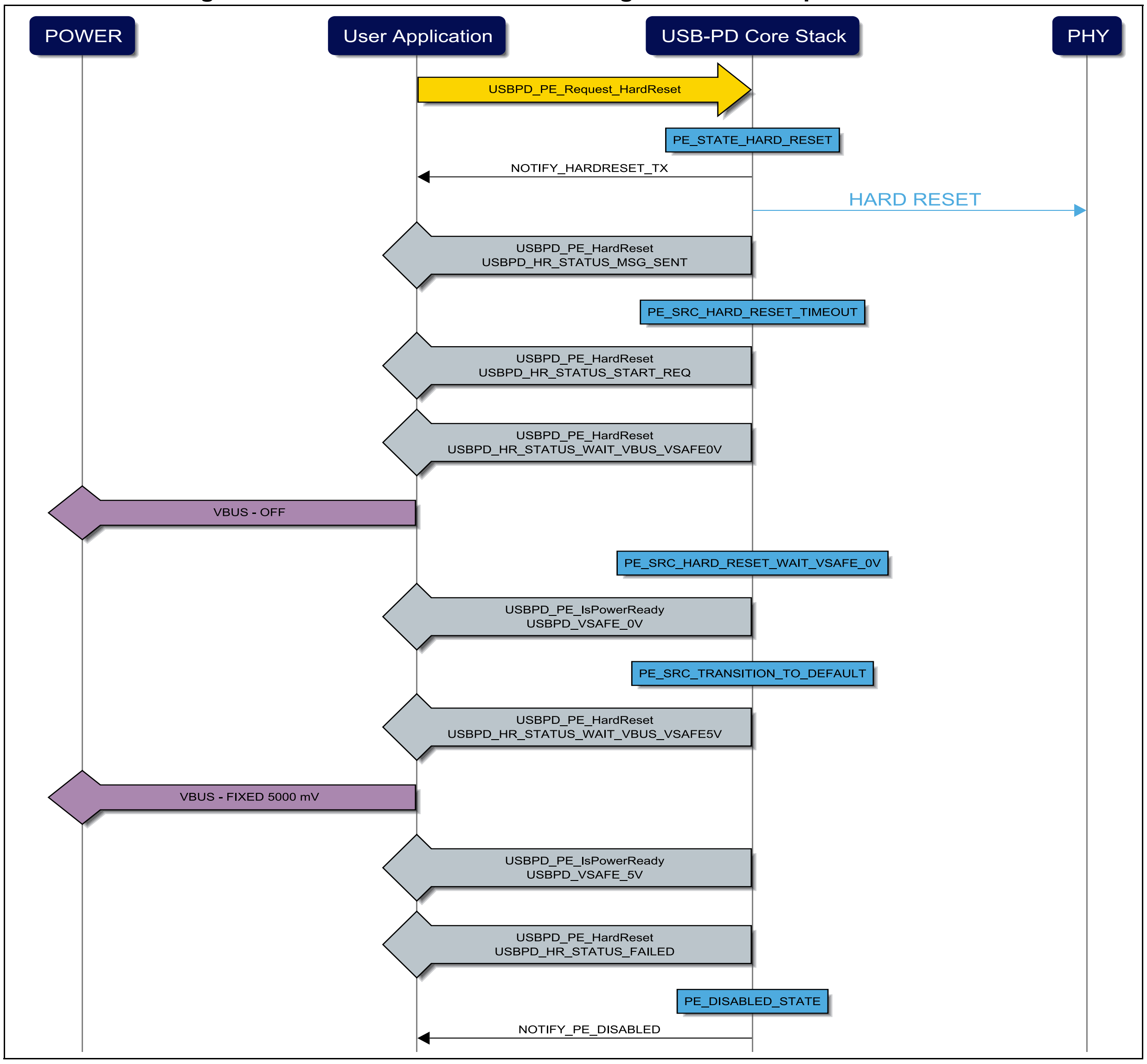

POWER
User Application
USB-PD Core Stack
PHY
USBPD_PE_Request_HardReset
PE_STATE_HARD_RESET
NOTIFY_HARDRESET_TX
HARD RESET
USBPD_PE_HardReset USBPD_HR_STATUS_MSG_SENT
PE_SRC_HARD_RESET_TIMEOUT
USBPD_PE_HardReset USBPD_HR_STATUS_START_REQ
USBPD_PE_HardReset USBPD_HR_STATUS_WAIT_VBUS_VSAFE0V
VBUS - OFF
PE_SRC_HARD_RESET_WAIT_VSAFE_0V
USBPD_PE_IsPowerReady USBPD_VSAFE_0V
Callback can be called several times
USBPD_PE_HardReset USBPD_HR_STATUS_FAILED
PE_DISABLED_STATE

Figure 33. Fail to reach vSafe5V during hard reset sequence in SRC


POWER
User Application
USB-PD Core Stack
PHY
USBPD_PE_Request_HardReset
PE_STATE_HARD_RESET
NOTIFY_HARDRESET_TX
HARD RESET
USBPD_PE_HardReset USBPD_HR_STATUS_MSG_SENT
PE_SRC_HARD_RESET_TIMEOUT
USBPD_PE_HardReset USBPD_HR_STATUS_START_REQ
USBPD_PE_HardReset USBPD_HR_STATUS_WAIT_VBUS_VSAFE0V
VBUS - OFF
PE_SRC_HARD_RESET_WAIT_VSAFE_0V
USBPD_PE_IsPowerReady USBPD_VSAFE_0V
PE_SRC_TRANSITION_TO_DEFAULT
USBPD_PE_HardReset USBPD_HR_STATUS_WAIT_VBUS_VSAFE5V
VBUS - FIXED 5000 mV
USBPD_PE_IsPowerReady USBPD_VSAFE_5V
USBPD_PE_HardReset USBPD_HR_STATUS_FAILED
PE_DISABLED_STATE
NOTIFY_PE_DISABLED

4.3.5 VCONN management AMS

The main difference with the sink case is that a source must power VCONN on the detection of an EMC cable.

The MSCs in *Figure 34* and *Figure 35* show, respectively, the behavior when VconnStatus = USBPD_TRUE (VCONN ON) and when VconnStatus = USBPD_FALSE (VCONN OFF).

Figure 34. VCONN swap when VCONN is ON

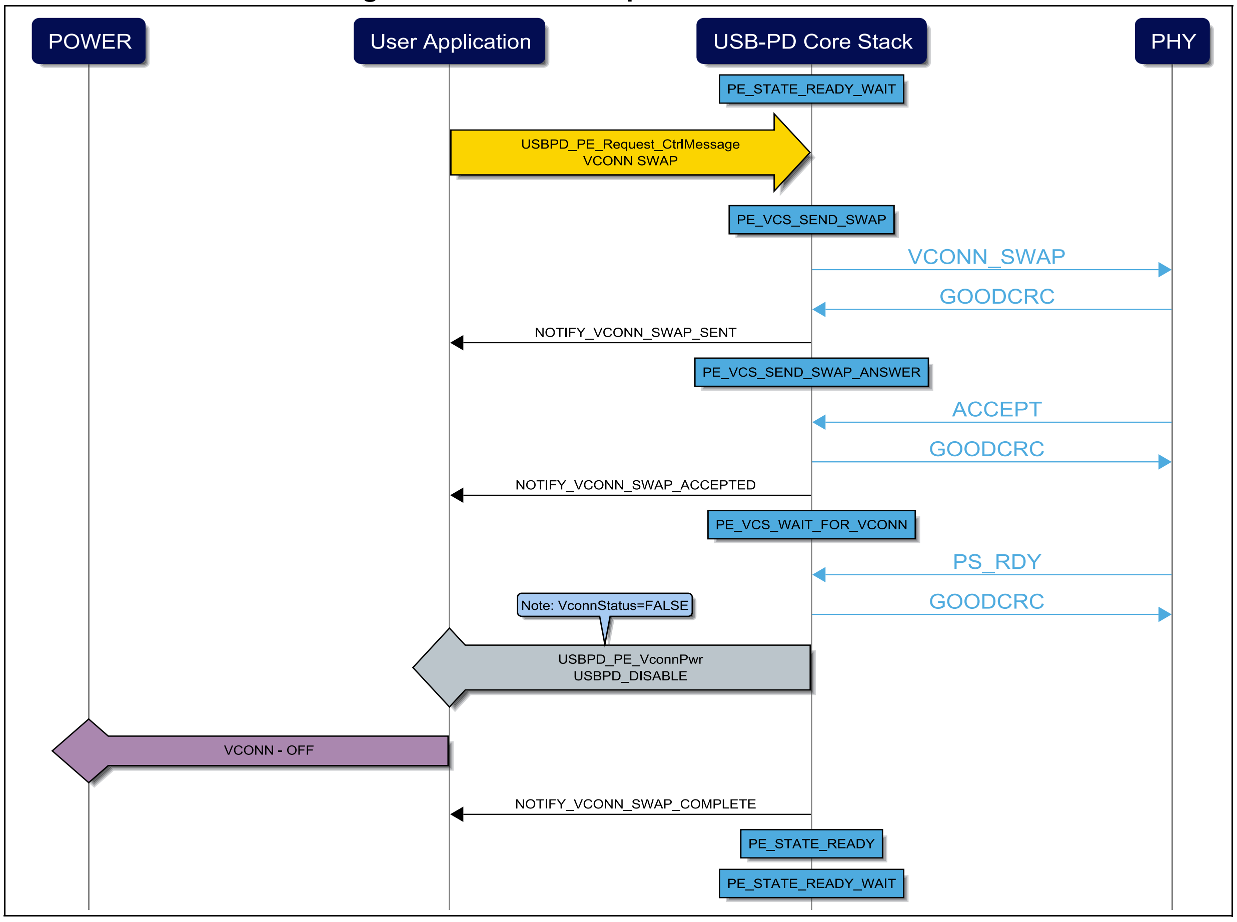
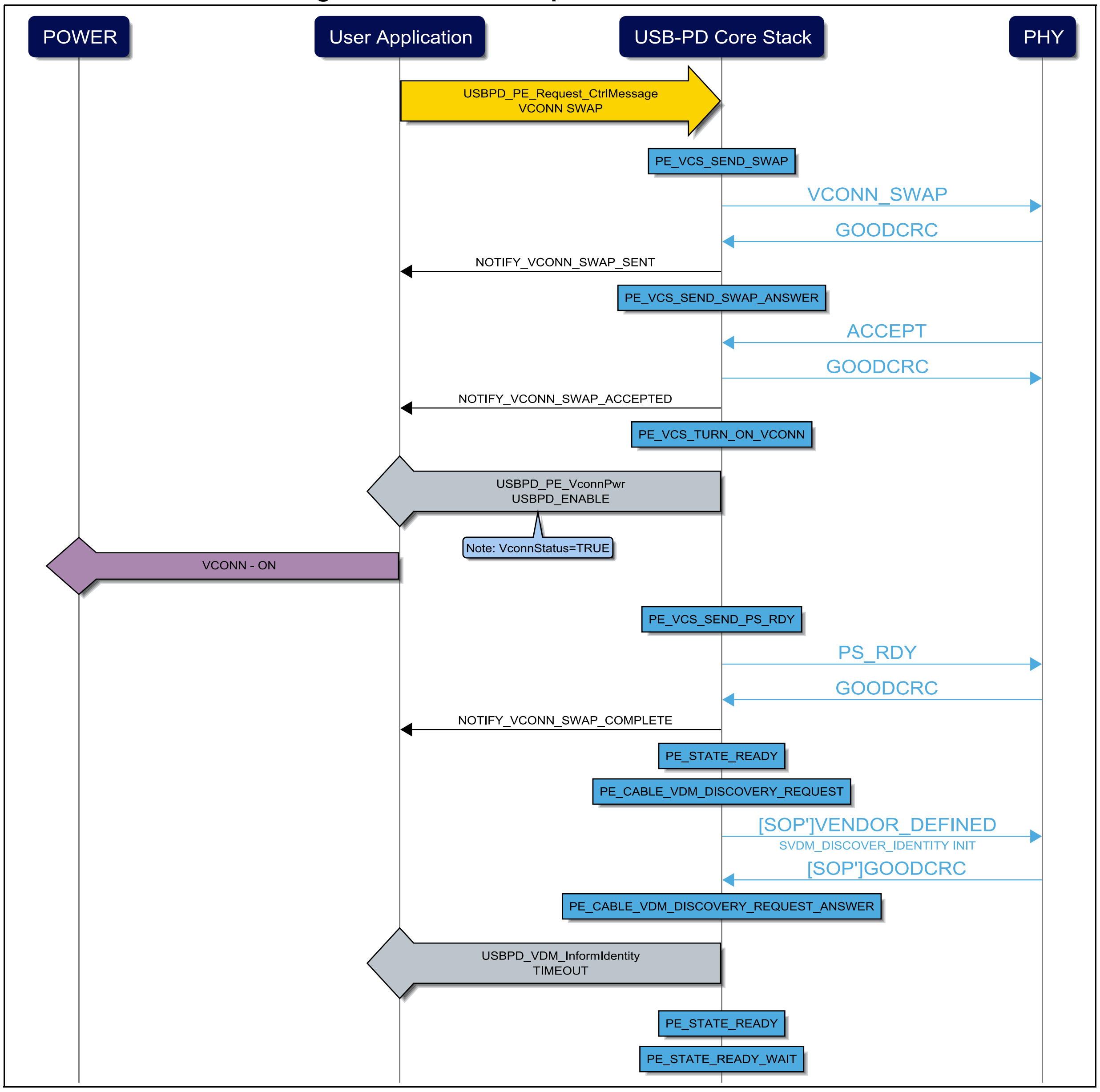


Figure 35. VCONN swap when VCONN is OFF



4.4 Generic AMS

4.4.1 Data role swap

A default data role is associated with each power role:

- SOURCE is DFP (USB host)
- SINK is UFP (USB device)

The DRS procedure switches the data role of both port partners: UFP → DFP and DFP → UFP without impact on the negotiated contract. The application can check if the port partners support DRS by checking the first source or sink PDO.

The MSC in *Figure 36* describes how to request a DRS, and that in *Figure 37* describes the reception of a DRS message.

Figure 36. Data role swap generation

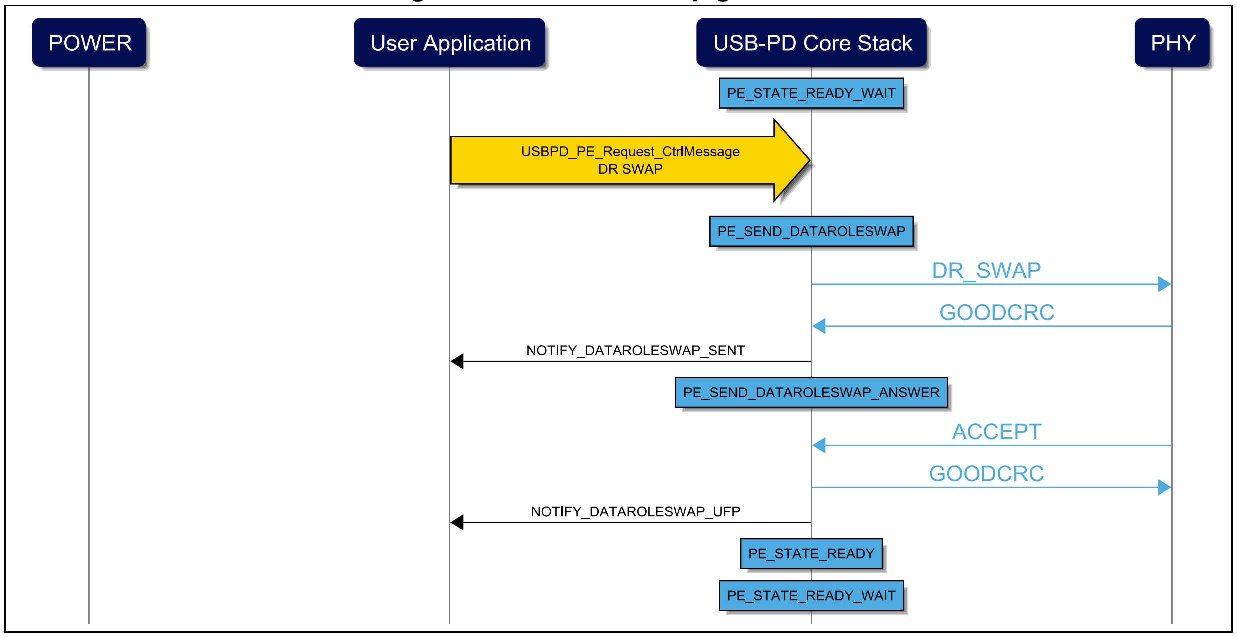
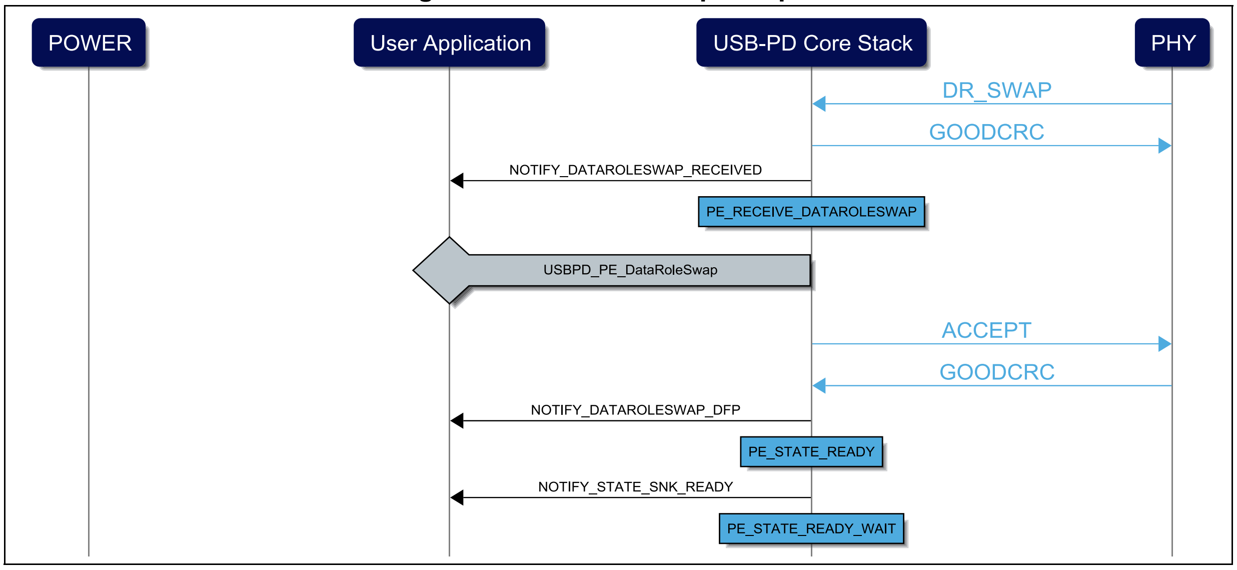


Figure 37. Data role swap reception



4.4.2 Soft reset

A soft reset message is mainly used to recover from protocol layer errors (for example if a message has not been acknowledged by a GOODCRC a soft reset message is automatically generated by the USB-PD stack). The application can be interested to force a soft reset to an ongoing AMS, or to be informed about the soft reset generation.

The MSC in *Figure 38* describes how to request a soft reset, and that in *Figure 39* its reception.

Figure 38. Soft reset generation

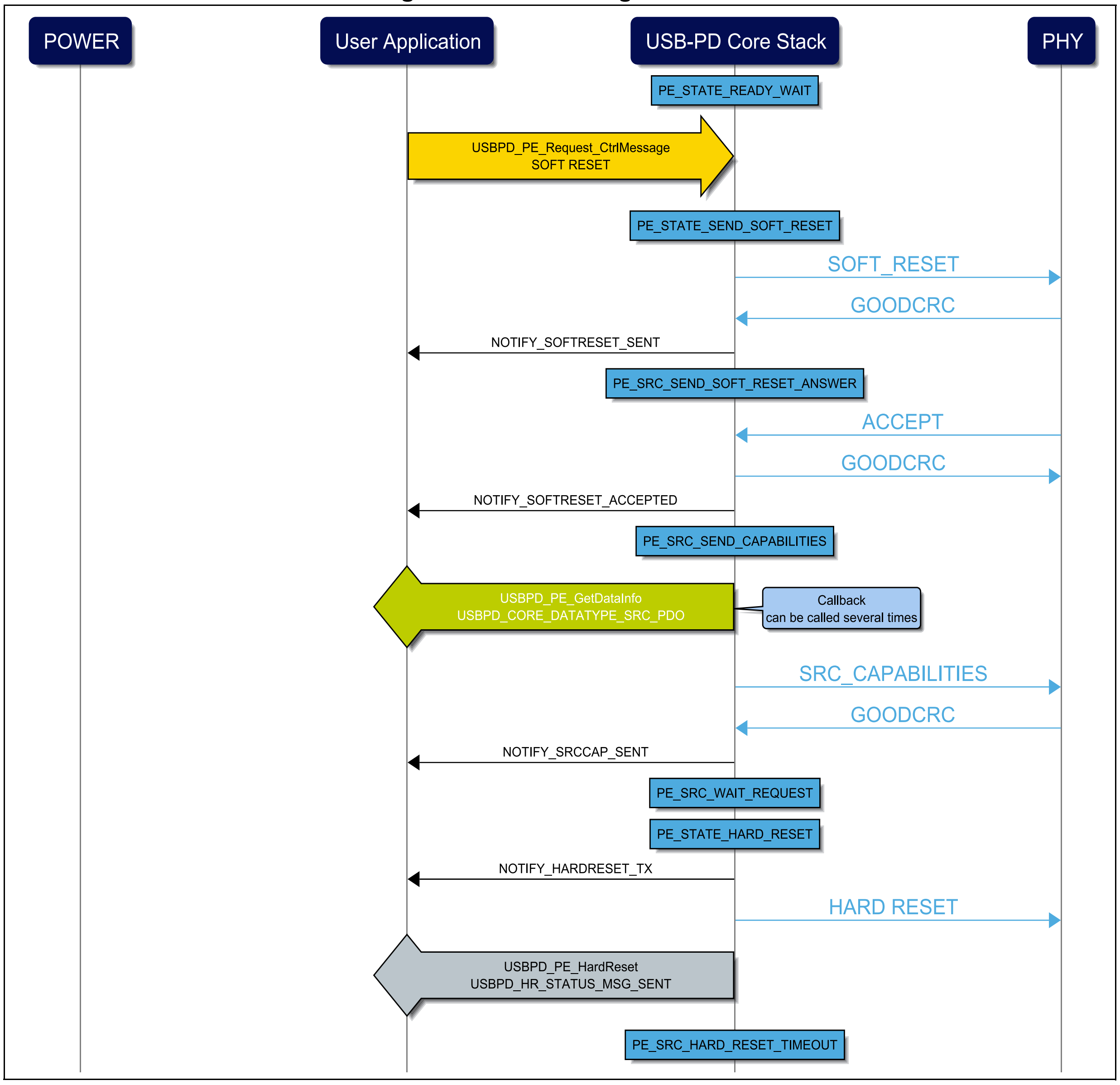
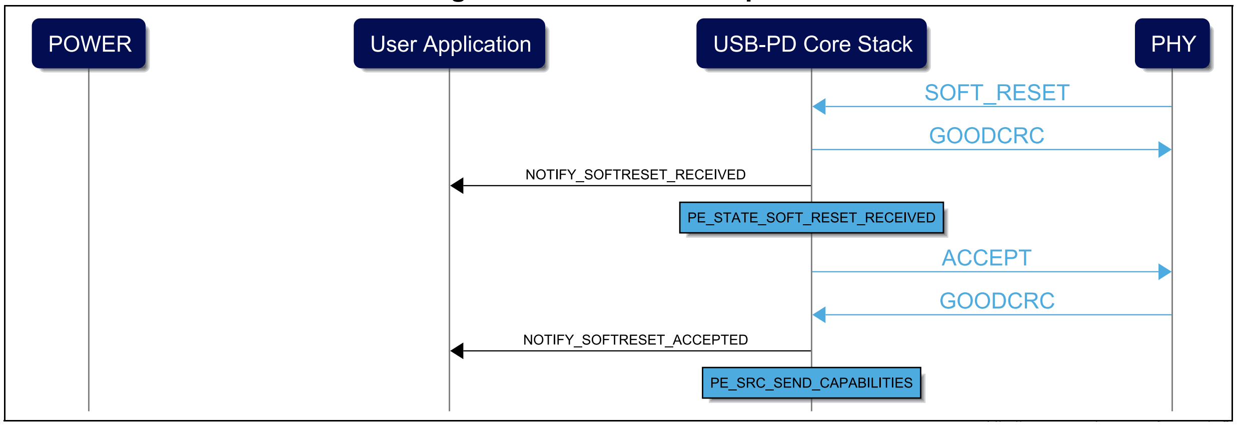


Figure 39. Soft reset reception



4.4.3 Source extended capabilities

A port may request additional information about a distant port source capabilities by sending *Get_Source_Cap_Extended* request.

A source-capable port may expose extended capabilities to provide more information to its partner, by answering *Get_Source_Cap_Extended* request. If not supported, response to the request will be NOT_SUPPORTED.

To get more detail about the extended capability check USBPD_SCEDB_TypeDef structure.

The AMS in *Figure 40* shows how to get source extended capabilities of a distant port, and that in *Figure 41* how to reply a get Source extended capability request.

Figure 40. Generated GET_SRC_CAPEXT

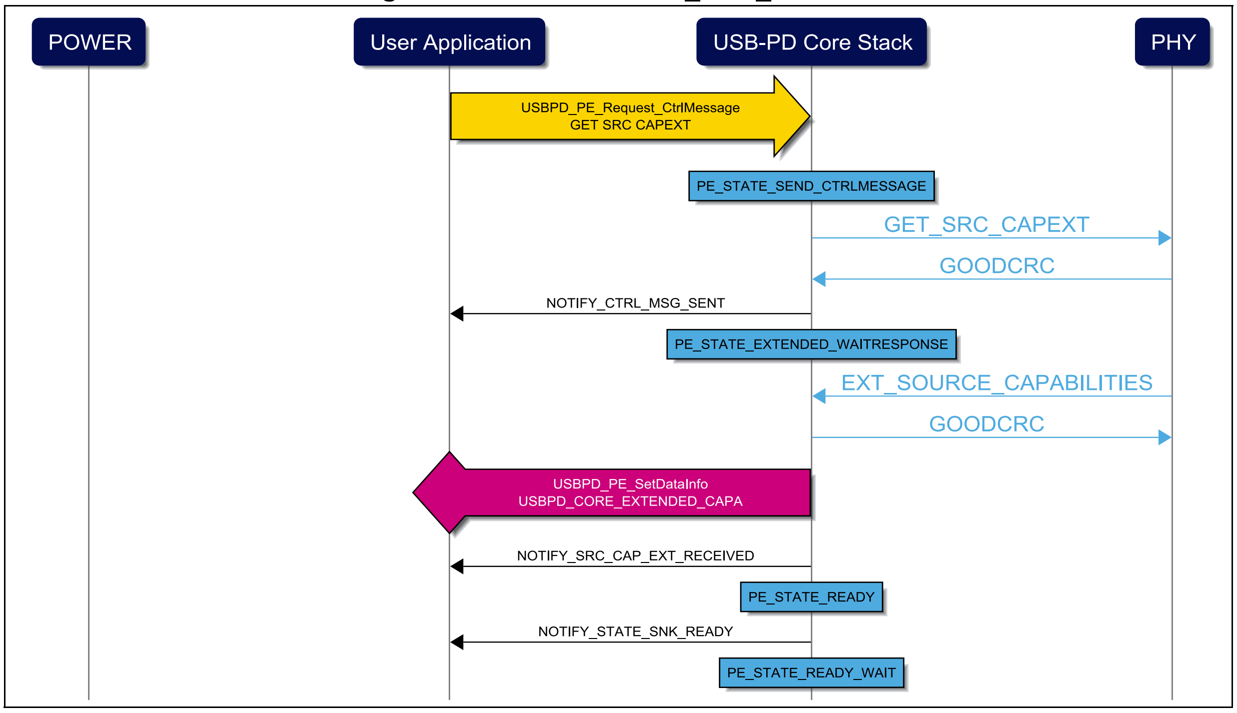
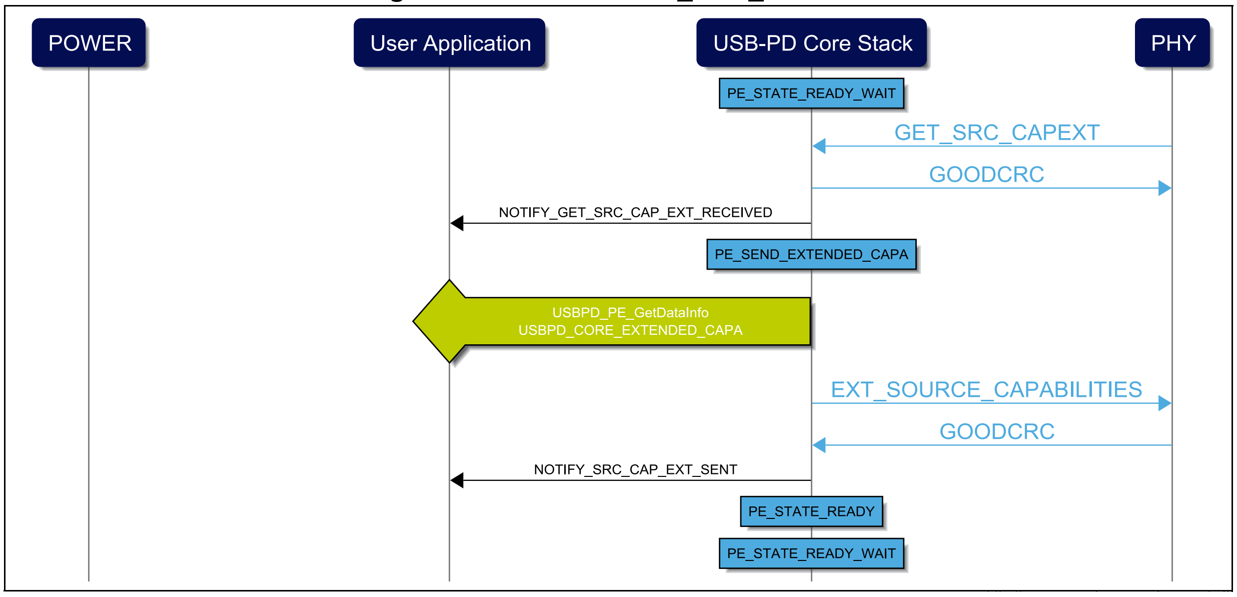


Figure 41. Received GET_SRC_CAPEXT



4.4.4 Sink extended capabilities

A port may request additional information about a distant Port's Sink Capabilities by sending *Get_Sink_Cap_Extended* request.

A sink-capable port may expose extended capabilities to provide more information to its partner, by answering Get_Sink_Cap_Extended request. If not supported, response to the request will be NOT_SUPPORTED.

To get more detail about the extended capability check USBPD_SKEDB_TypeDef structure.

Figure 42 shows how to get sink extended capabilities of a distant port, and *Figure 43* how to reply a get sink extended capability request.

Figure 42. Generate GET_SNK_CAPEXT

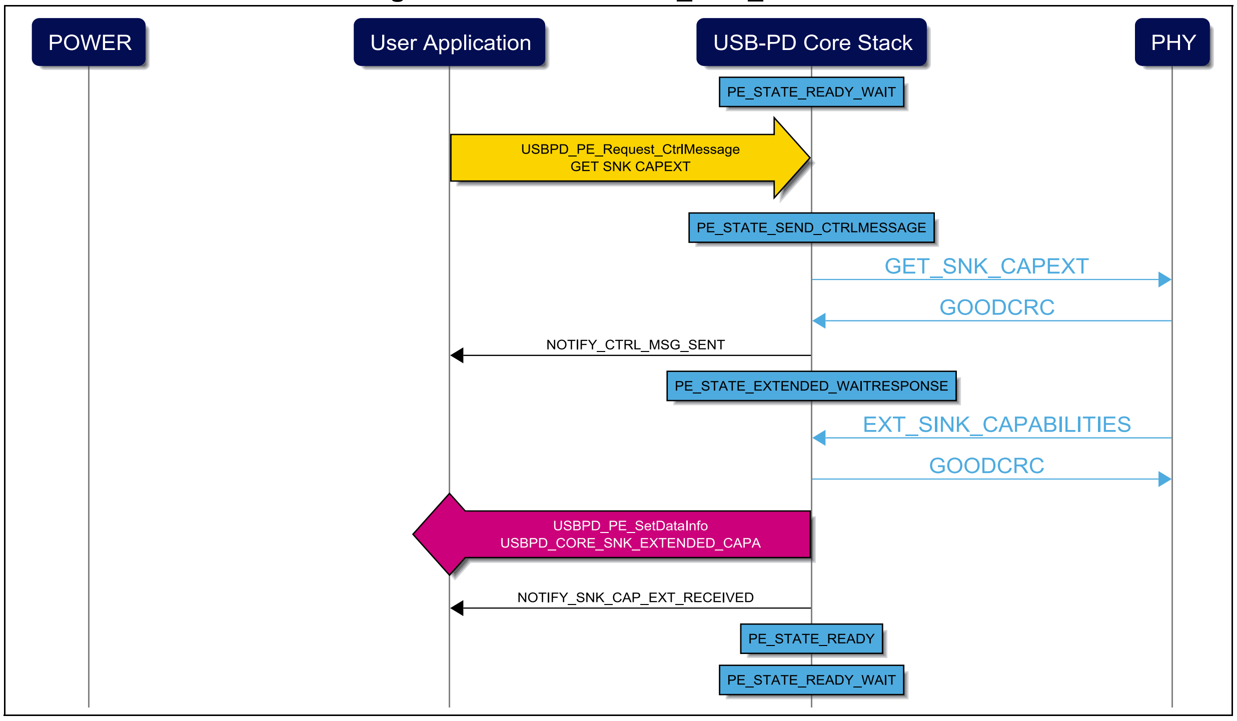
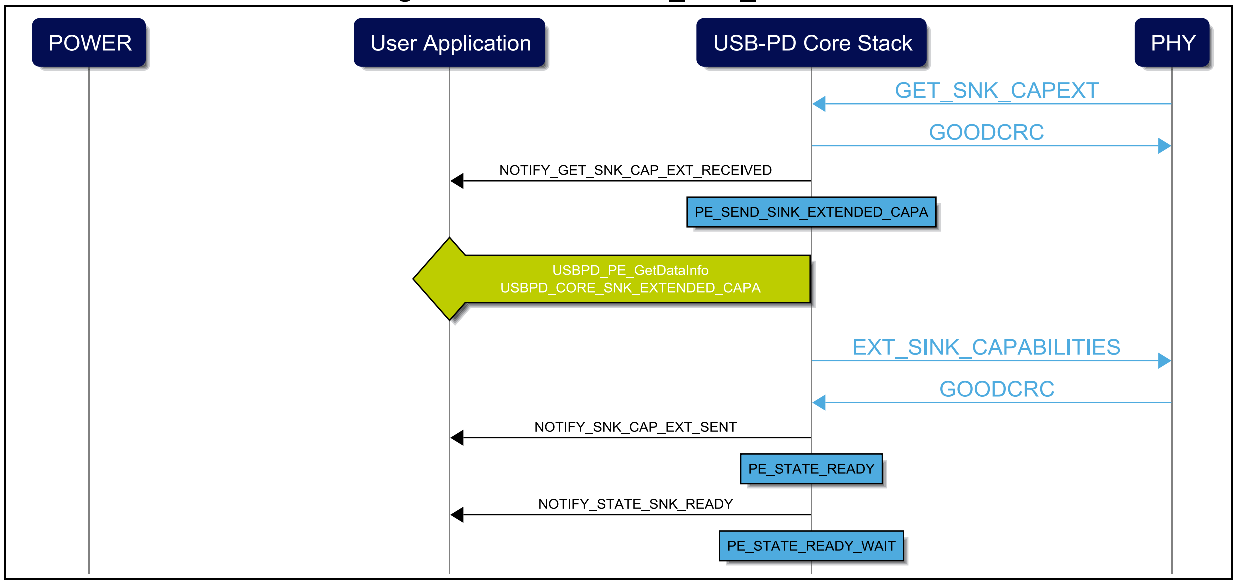


Figure 43. Receive GET_SNK_CAPEXT



4.5 VDM AMS

This section describes AMS for different VDM sequences (in DFP and UFP modes).

To enable VDM on the application:

- Use libraries USB-PDXXXX_PDX_FULL_xx[a]
- DPM_Settings[USBPD_PORT_x].PE_VDMSupport = USBPD_TRUE
  - Stack will answer if possible to VDM messages
- DPM_Settings[USBPD_PORT_x].PE_RespondsToDiscovSOP = USBPD_TRUE
  - Stack will answer to VDM messages
- DPM_Settings[USBPD_PORT_x].PE_AttemptsDiscovSOP = USBPD_TRUE
  - Stack allows DPM to request VDM messages
- Need to implement different VDM callbacks when necessary

Request is accepted by stack:

- For SOP*: only if a contract has been established
  - Contract not necessary for Discovery Identify on SOP', SOP".
- For SOP' or SOP": VCONN has been enabled (DPM_Params[USB-PD_PORT_x].VconnStatus)

4.5.1 DFP

Table 11. DFP AMS

AMS	Description	Reference
Discovery identity	Discover VDM identity from port partner or EMC cable	*Figure 44*
Discovery SVID	Discover different SVIDs supported by port partner or EMC cable	*Figure 45*
Discovery modes	Discover different modes linked to a SVID supported by port partner or EMC cable	*Figure 46*
Enter mode	Enter in one mode when different modes have been returned in VDM discovery modes ACK	*Figure 47*
Exit mode	Exit from a mode	*Figure 48*
Attention	Request for attention	*Figure 49*

a. If you just need to communicate with an EMC cable, a USB-PDXXXX_PDX_CONFIG_1_xx can be used.

Figure 44. VDM discovers identity request

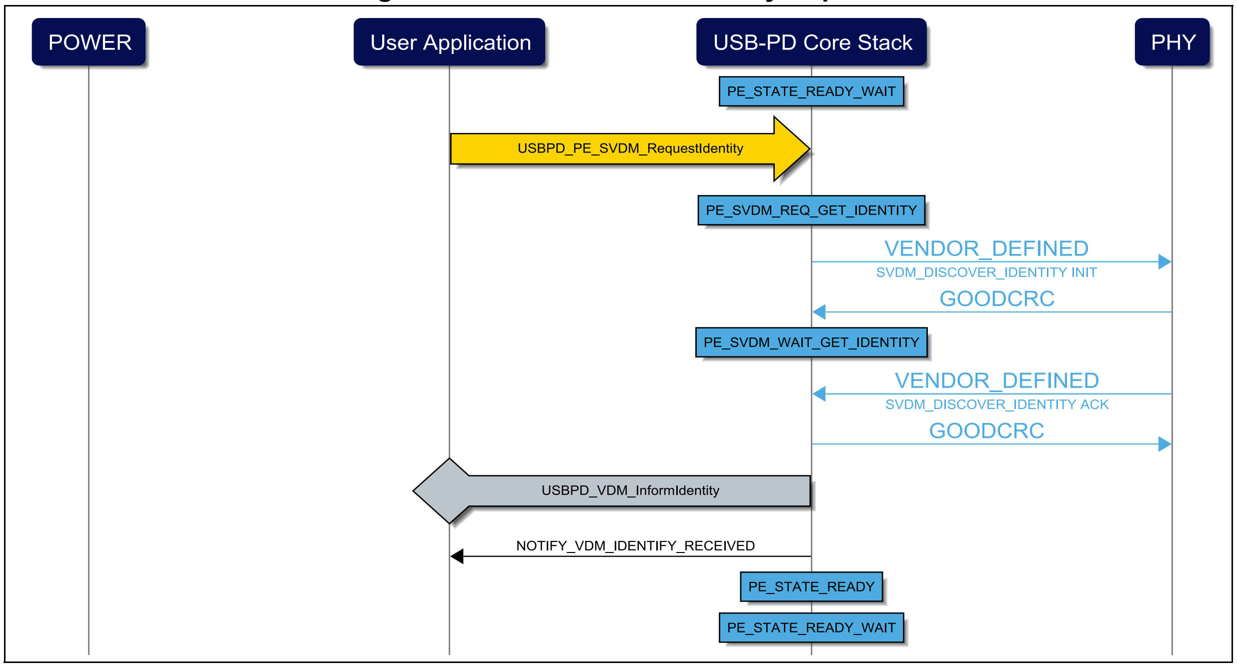


Figure 45. VDM discovers SVID request

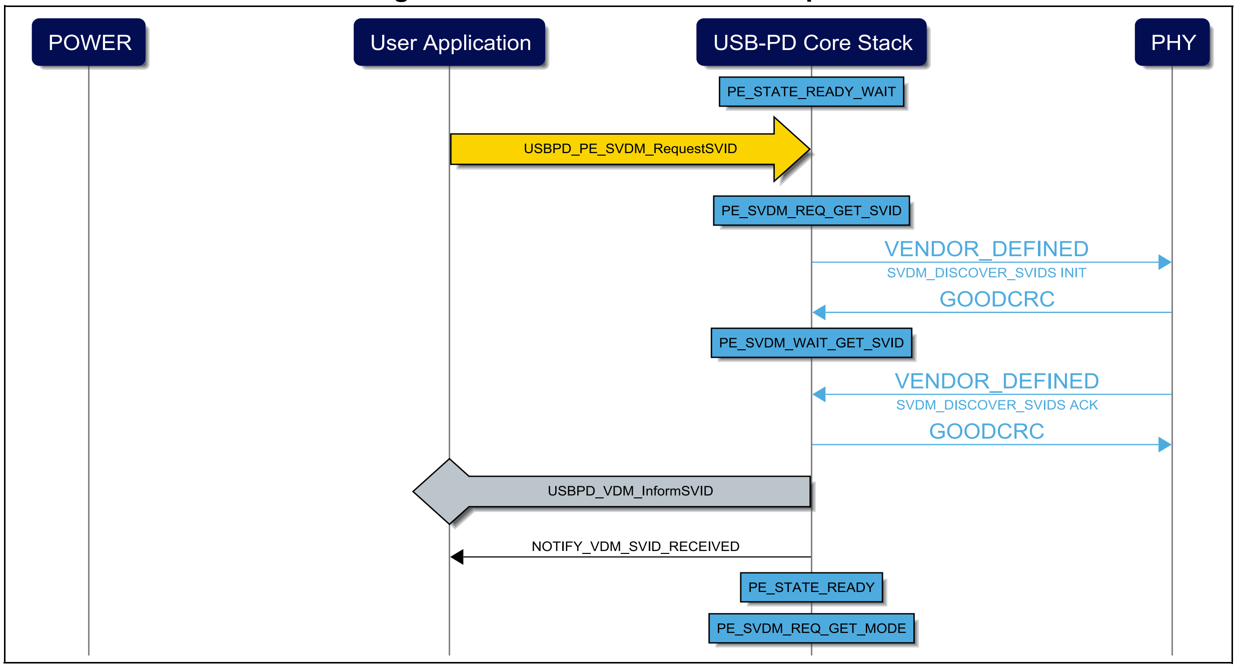


Figure 46. VDM discovers mode request

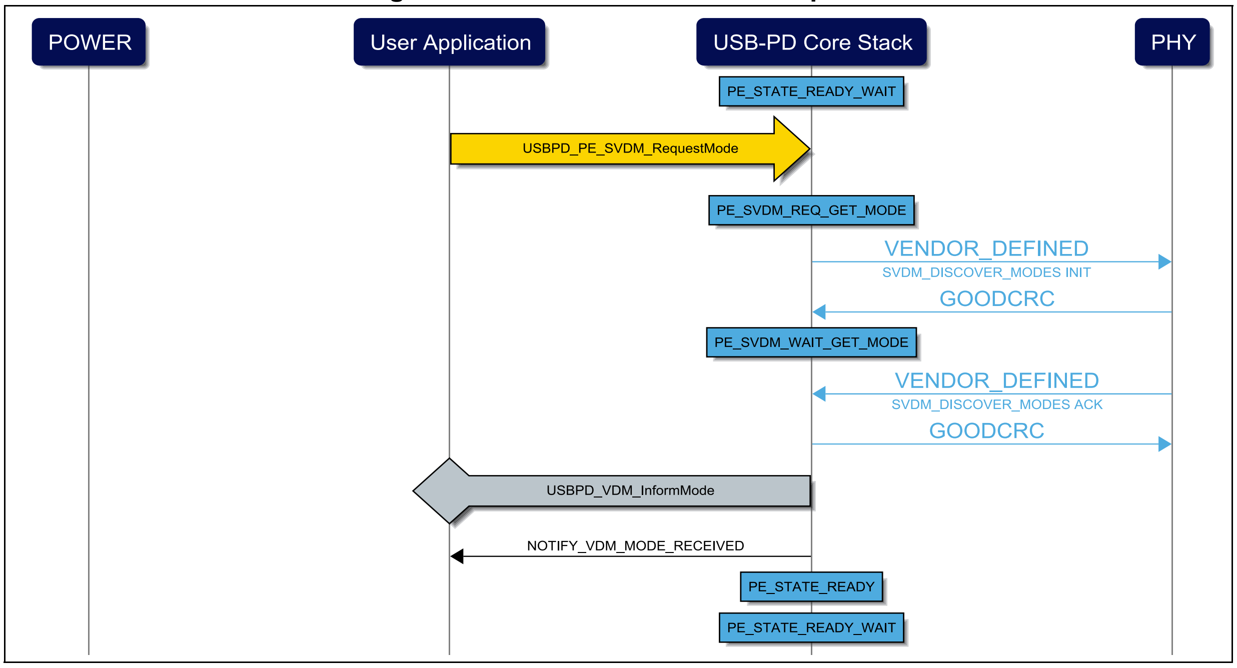


Figure 47. VDM enters mode request

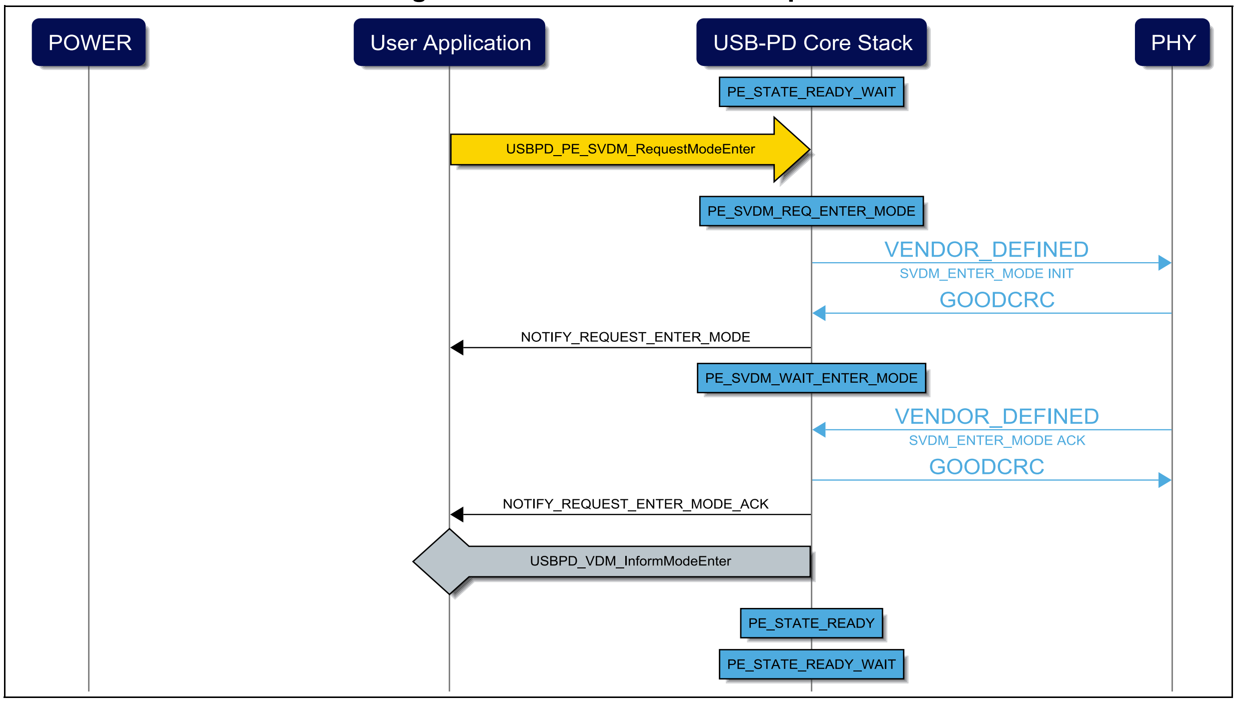


Figure 48. VDM exits mode request

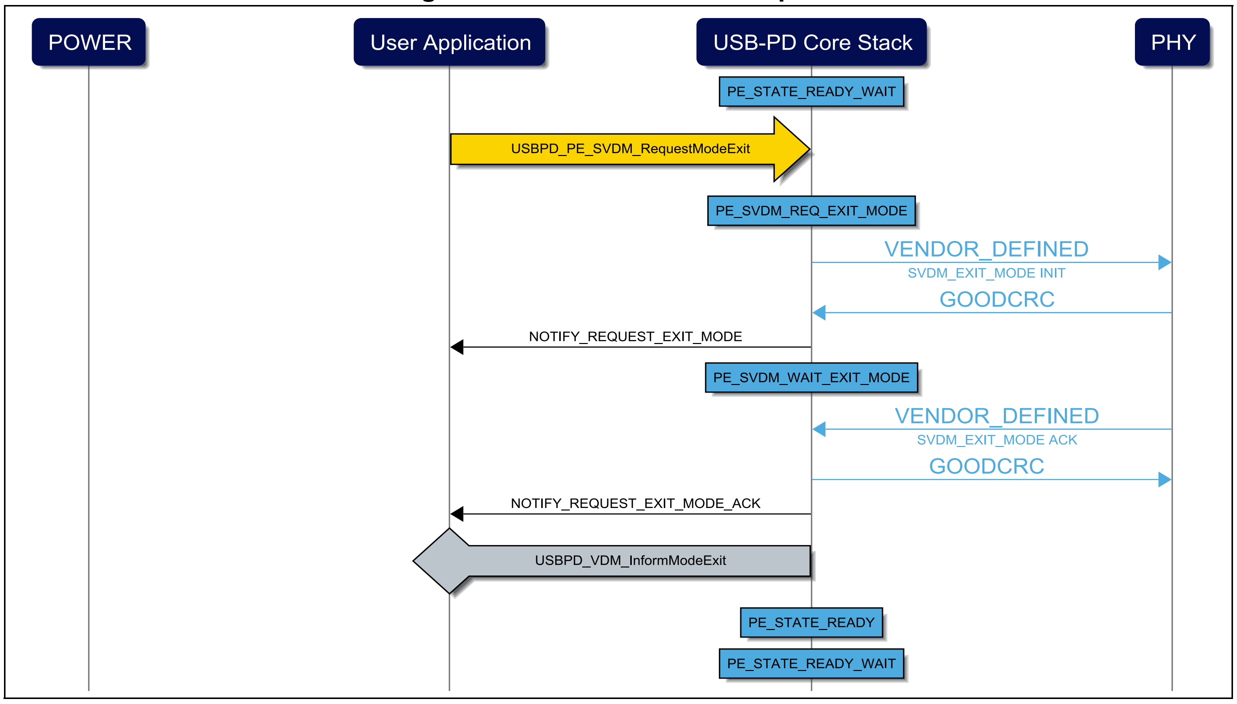
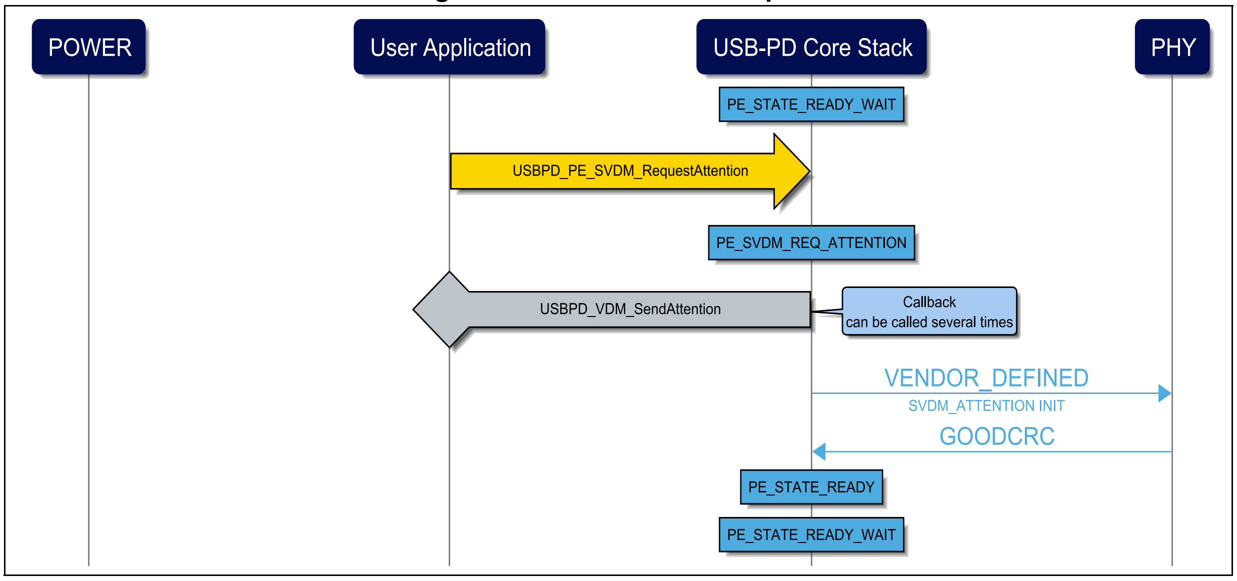


Figure 49. VDM attention request



Specific sequence: display port

VDM specific messages are used for different modes (e.g. DP / HDMI / Thunderbold).

The following sequences are used for display port initialization. For further details refer to *[3]*.

DP configure request / DP status

If the application wants to send a DP configure / a DP status request message to port partner, the sequence of *Figure 50* / *Figure 51*, respectively, must be used.

Figure 50. VDM display port configure request

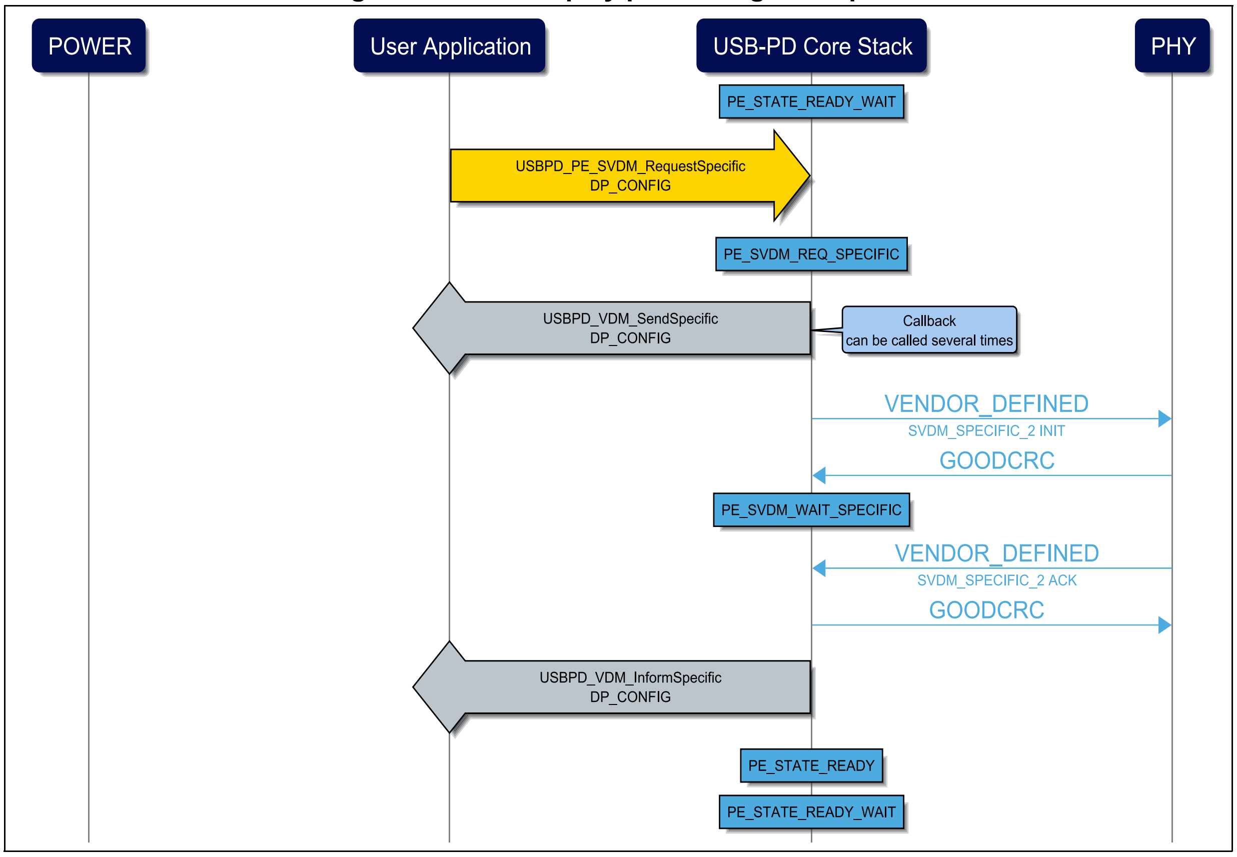
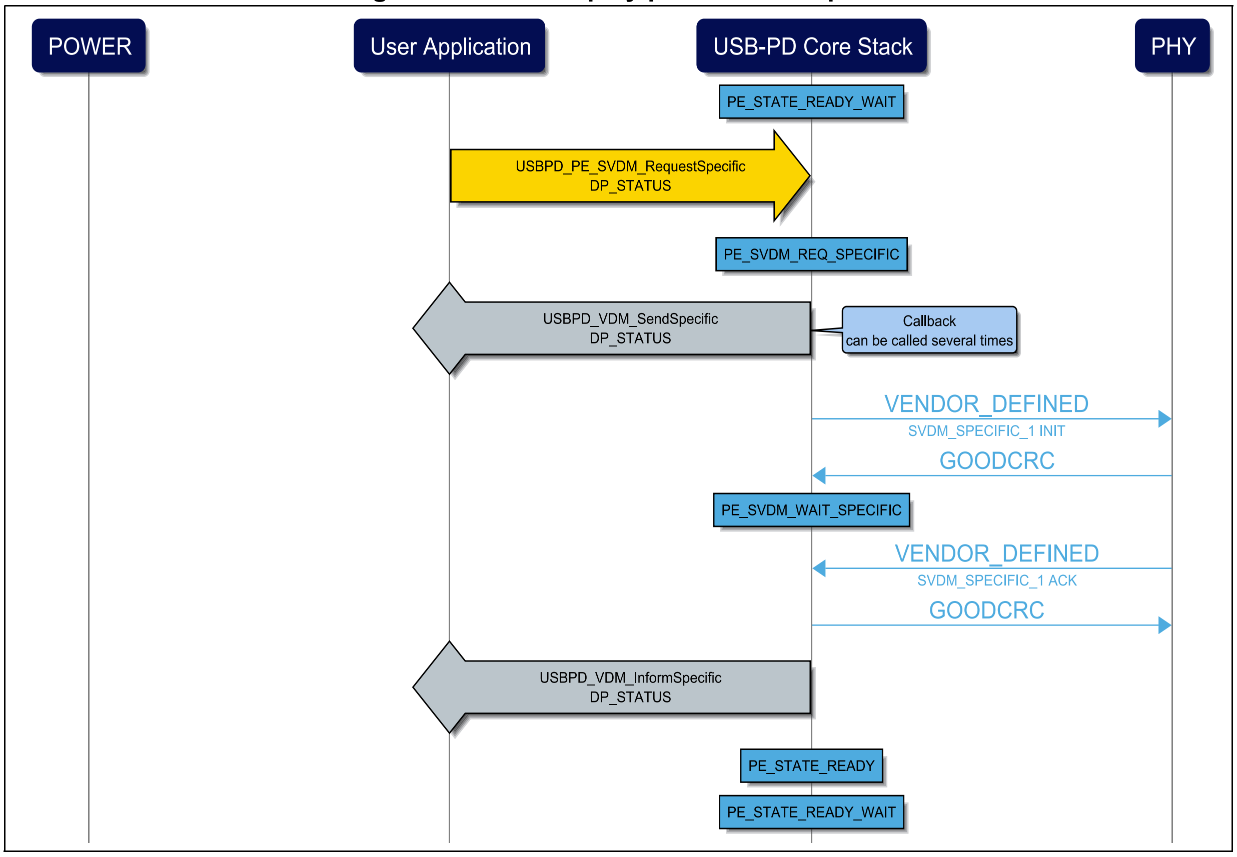


Figure 51. VDM display port status request



4.5.2 UFP

Table 12. DFP UFP

AMS	Description	Reference
Discovery identity	Manage a VDM Identity from a port partner	*Figure 52*
Discovery SVID	Manage a VDM Discovery SVID received by a port partner	*Figure 53*
Discovery modes	Manage a VDM discovery mode received by a port partner	*Figure 54*
Enter mode	Manage a request to enter in a mode received by a port partner	*Figure 55*
Exit mode	Manage a request to exit a mode received by a port partner	*Figure 56*
Attention	Request for attention received by a port partner	*Figure 57*

Figure 52. VDM discover identity received

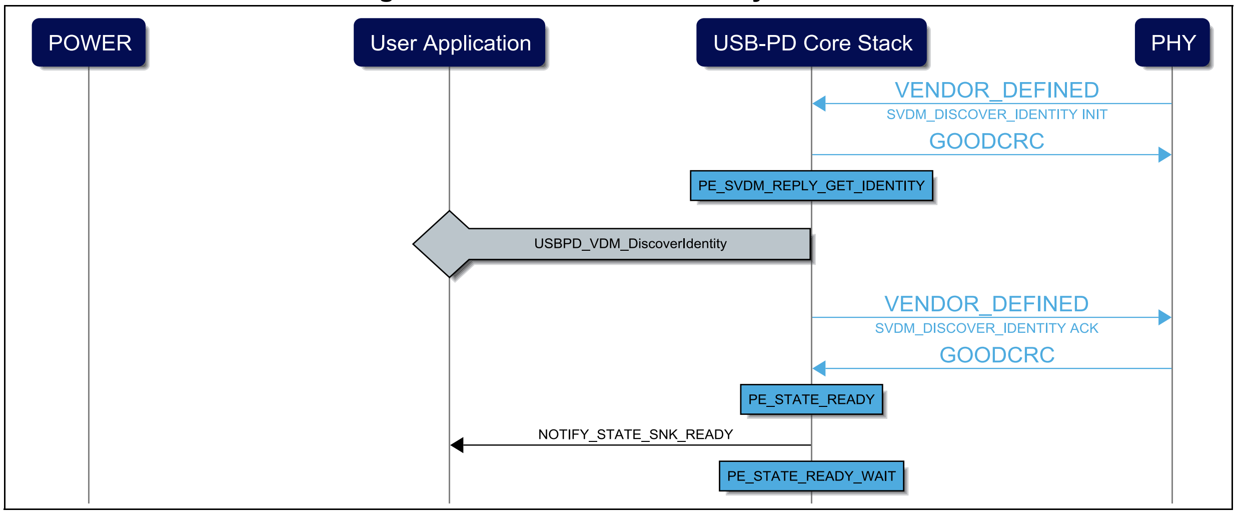


Figure 53. VDM discover SVID received

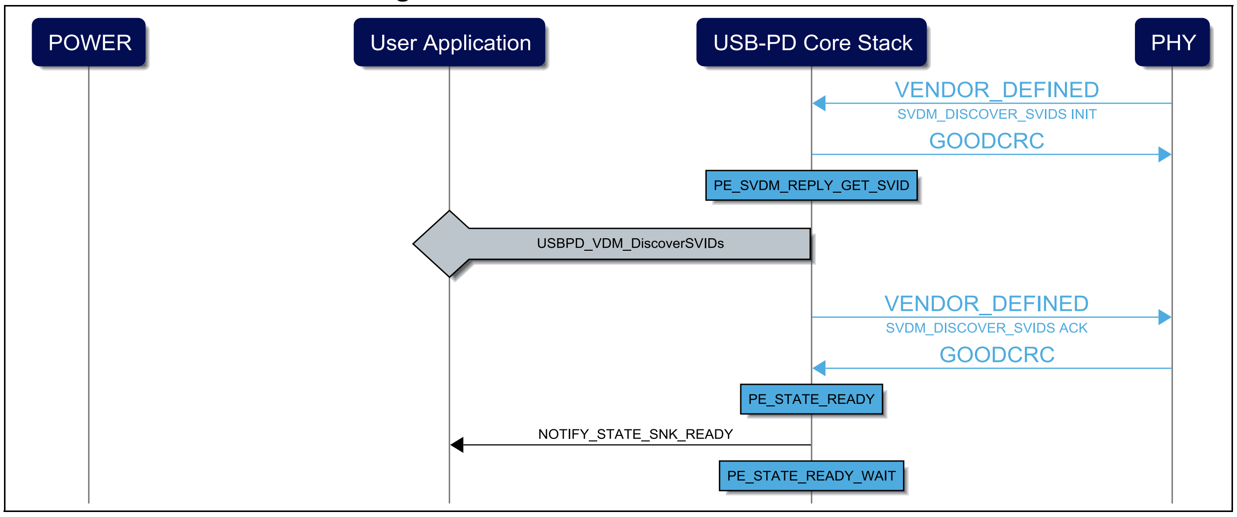


Figure 54. VDM discover modes received

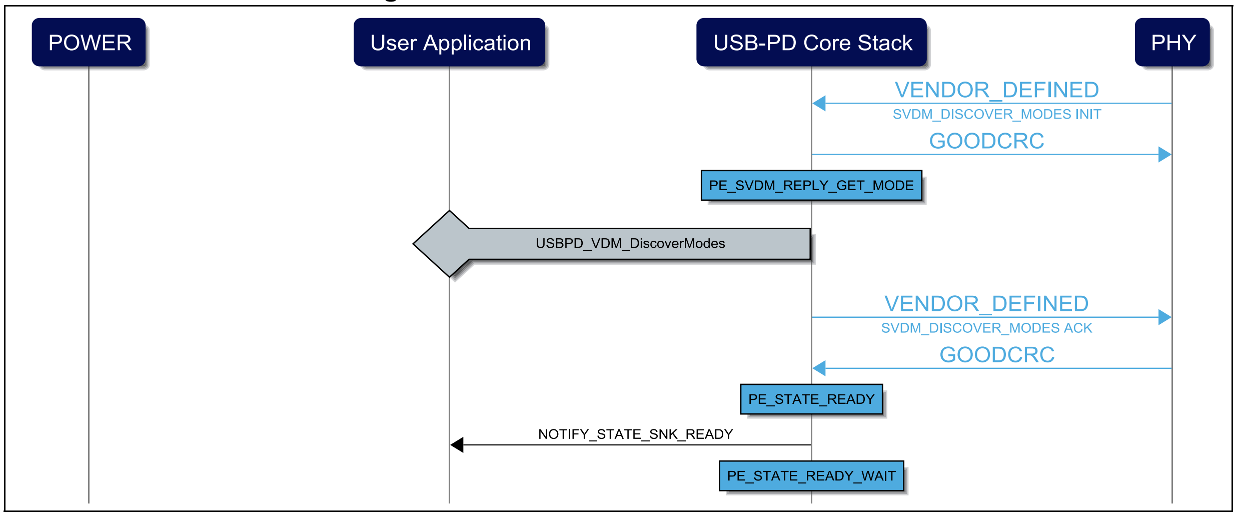


Figure 55. VDM enter mode received

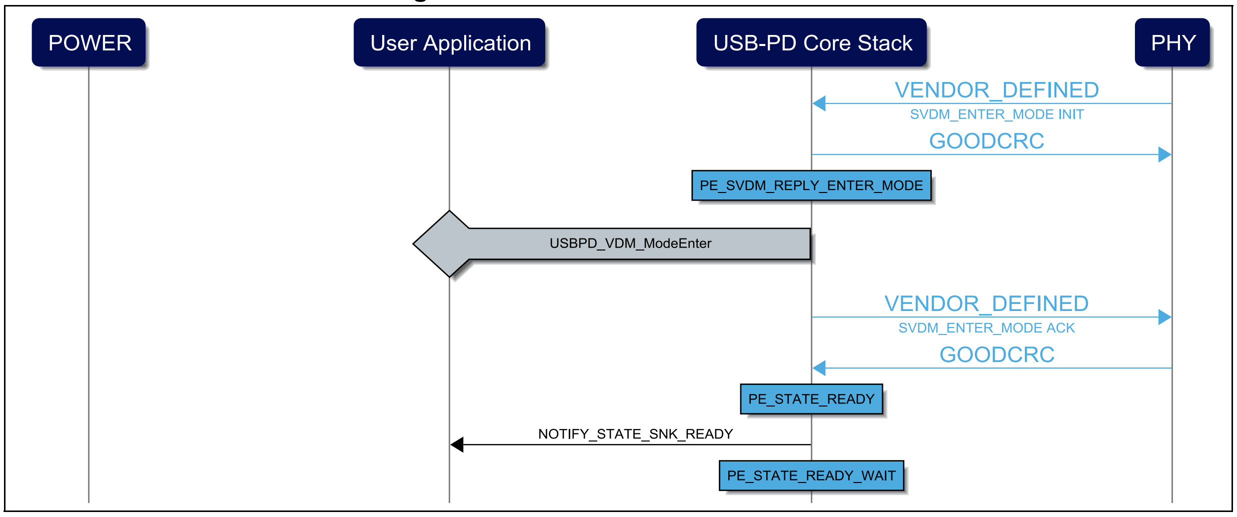


Figure 56. VDM exit mode received

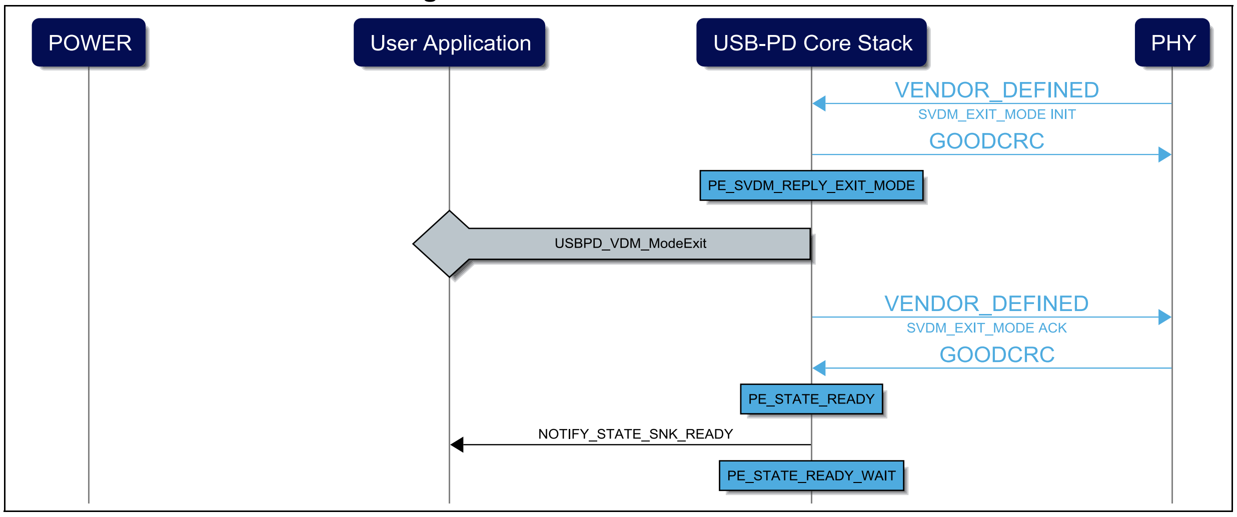
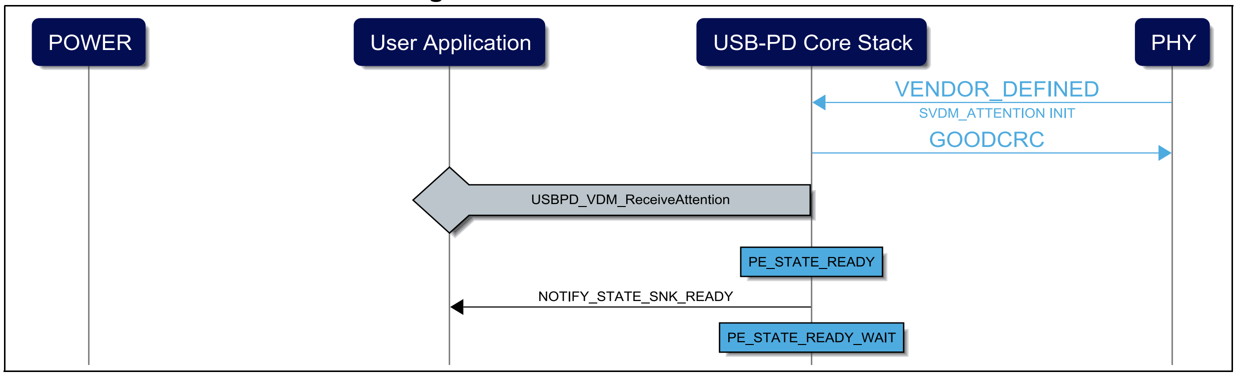


Figure 57. VDM attention received



Specific sequence: Display port

DP configure (*Figure 58*) and DP status (*Figure 59*) sequences are used to manage specific messages received by port partner (in this document focus is mostly on Display port sequences).

Figure 58. VDM Display port configure received

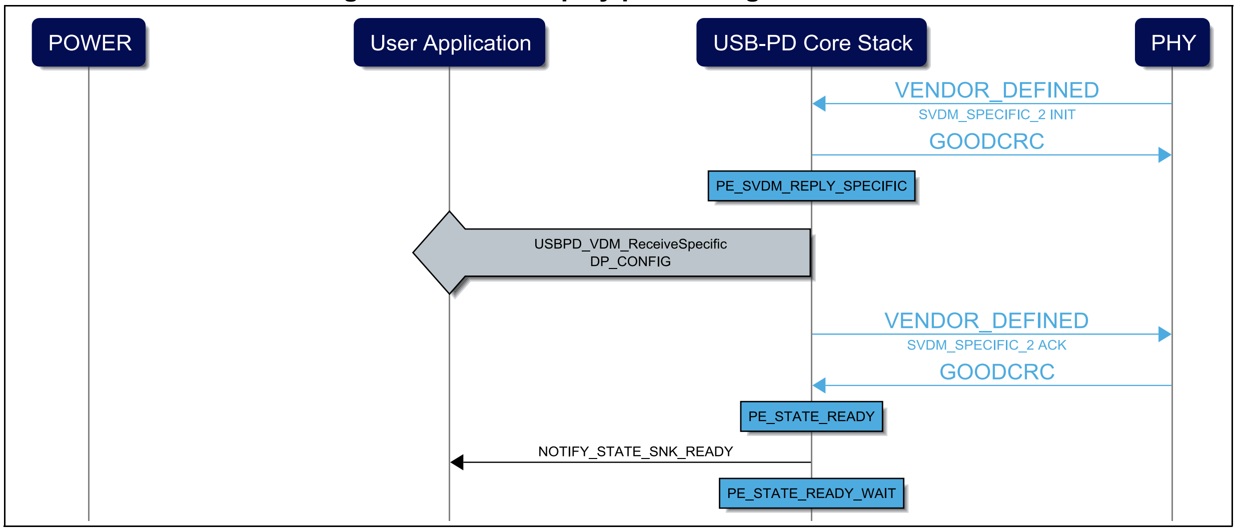
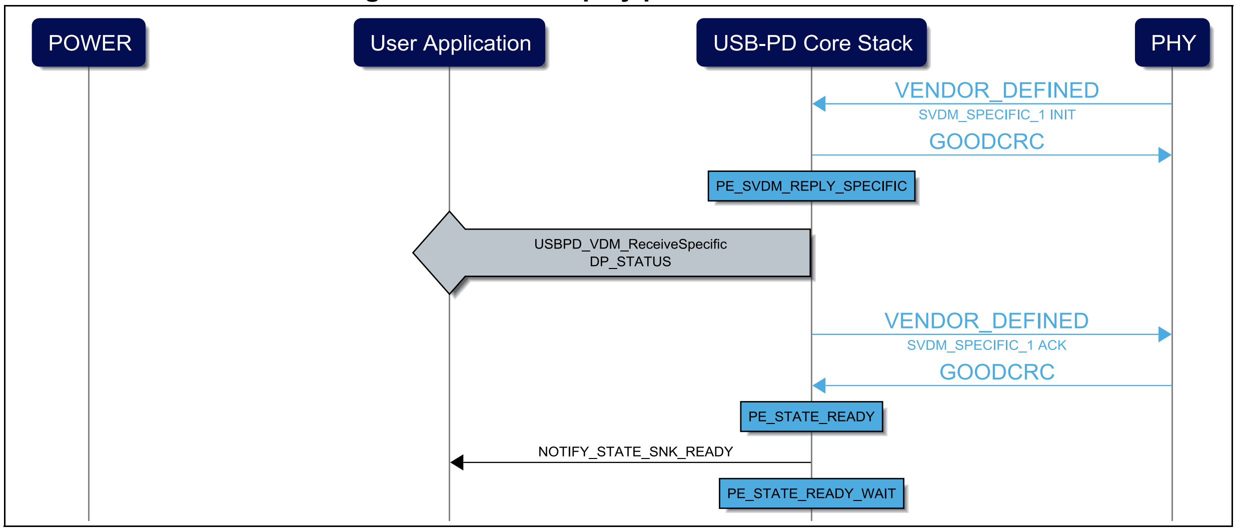


Figure 59. VDM Display port status received



Appendix A DATA_ID

Table 13. DATA_ID

DATA_ID	Description
USBPD_CORE_DATATYPE_SRC_PDO	Port source PDO
USBPD_CORE_DATATYPE_SNK_PDO	Port sink PDO
USBPD_CORE_DATATYPE_RDO_POSITION	Stores the requested DO position in PDO list
USBPD_CORE_DATATYPE_REQ_VOLTAGE	Stores the requested voltage value
USBPD_CORE_DATATYPE_RCV_SRC_PDO	Stores the received source PDO values
USBPD_CORE_DATATYPE_RCV_SNK_PDO	Stores the received sink PDO values
USBPD_CORE_DATATYPE_RCV_REQ_PDO	Stores the received sink request PDO value
USBPD_CORE_DATATYPE_REQUEST_DO	Stores the P DO to be used in request message (from sink to source)
USBPD_CORE_EXTENDED_CAPA	Extended capability
USBPD_CORE_INFO_STATUS	Information status
USBPD_CORE_PPS_STATUS	PPS status data
USBPD_CORE_ALERT	Alert
USBPD_CORE_GET_MANUFACTURER_INFO	Get manufacturer info
USBPD_CORE_MANUFACTURER_INFO	Manufacturer info
USBPD_CORE_GET_BATTERY_STATUS	Get battery status
USBPD_CORE_BATTERY_STATUS	Battery status
USBPD_CORE_GET_BATTERY_CAPABILITY	Get battery capability
USBPD_CORE_BATTERY_CAPABILITY	Battery capability
USBPD_CORE_UNSTRUCTURED_VDM	Unstructured VDM message
USBPD_CORE_SNK_EXTENDED_CAPA	Sink extended capability

Appendix B Notification ID

- USBPD_NOTIFY_REQUEST_ACCEPTED
- USBPD_NOTIFY_REQUEST_REJECTED
- USBPD_NOTIFY_REQUEST_WAIT
- USBPD_NOTIFY_REQUEST_GOTOMIN
- USBPD_NOTIFY_GETSNKCAP_SENT
- USBPD_NOTIFY_GETSNKCAP_RECEIVED
- USBPD_NOTIFY_GETSNKCAP_ACCEPTED
- USBPD_NOTIFY_GETSNKCAP_REJECTED
- USBPD_NOTIFY_GETSNKCAP_TIMEOUT
- USBPD_NOTIFY_SNKCAP_SENT
- USBPD_NOTIFY_GETSRCCAP_SENT
- USBPD_NOTIFY_GETSRCCAP_RECEIVED
- USBPD_NOTIFY_GETSRCCAP_ACCEPTED
- USBPD_NOTIFY_GETSRCCAP_REJECTED
- USBPD_NOTIFY_SRCCAP_SENT
- USBPD_NOTIFY_POWER_EXPLICIT_CONTRACT
- USBPD_NOTIFY_POWER_SRC_READY
- USBPD_NOTIFY_POWER_SNK_READY
- USBPD_NOTIFY_POWER_SNK_STOP
- USBPD_NOTIFY_POWER_SWAP_TO_SNK_DONE
- USBPD_NOTIFY_POWER_SWAP_TO_SRC_DONE
- USBPD_NOTIFY_POWER_SWAP_REJ
- USBPD_NOTIFY_POWER_SWAP_NOT_SUPPORTED
- USBPD_NOTIFY_RESISTOR_ASSERT_RP
- USBPD_NOTIFY_RESISTOR_ASSERT_RD
- USBPD_NOTIFY_SVDM_ACK
- USBPD_NOTIFY_SVDM_NACK
- USBPD_NOTIFY_SVDM_BUSY
- USBPD_NOTIFY_SVDM_TIMEOUT
- USBPD_NOTIFY_HARDRESET_RX
- USBPD_NOTIFY_HARDRESET_TX
- USBPD_NOTIFY_STATE_SNK_READY
- USBPD_NOTIFY_STATE_SRC_DISABLED
- USBPD_NOTIFY_DATAROLESWAP_SENT
- USBPD_NOTIFY_DATAROLESWAP_RECEIVED
- USBPD_NOTIFY_DATAROLESWAP_UFP
- USBPD_NOTIFY_DATAROLESWAP_DFP
- USBPD_NOTIFY_DATAROLESWAP_WAIT
- USBPD_NOTIFY_DATAROLESWAP_REJECTED

- USBPD_NOTIFY_DATAROLESWAP_NOT_SUPPORTED
- USBPD_NOTIFY_GOTOMIN_SENT
- USBPD_NOTIFY_GOTOMIN_POWERREADY
- USBPD_NOTIFY_SNK_GOTOMIN
- USBPD_NOTIFY_SNK_GOTOMIN_READY
- USBPD_NOTIFY_REQUEST_ERROR
- USBPD_NOTIFY_REQUEST_COMPLETE
- USBPD_NOTIFY_REQUEST_CANCELED
- USBPD_NOTIFY_SOFTRESET_SENT
- USBPD_NOTIFY_SOFTRESET_ACCEPTED
- USBPD_NOTIFY_SOFTRESET_RECEIVED
- USBPD_NOTIFY_PING_RECEIVED
- USBPD_NOTIFY_REQUEST_ENTER_MODE
- USBPD_NOTIFY_REQUEST_ENTER_MODE_ACK
- USBPD_NOTIFY_REQUEST_ENTER_MODE_NAK
- USBPD_NOTIFY_REQUEST_ENTER_MODE_BUSY
- USBPD_NOTIFY_PD_SPECIFICATION_CHANGE
- USBPD_NOTIFY_POWER_SWAP_SENT
- USBPD_NOTIFY_POWER_SWAP_ACCEPTED
- USBPD_NOTIFY_POWER_SWAP_WAIT
- USBPD_NOTIFY_POWER_SWAP_RECEIVED
- USBPD_NOTIFY_VCONN_SWAP_RECEIVED
- USBPD_NOTIFY_VCONN_SWAP_SENT
- USBPD_NOTIFY_VCONN_SWAP_ACCEPTED
- USBPD_NOTIFY_VCONN_SWAP_WAIT
- USBPD_NOTIFY_VCONN_SWAP_REJECTED
- USBPD_NOTIFY_VCONN_SWAP_COMPLETE
- USBPD_NOTIFY_VCONN_SWAP_NOT_SUPPORTED
- USBPD_NOTIFY_CTRL_MSG_SENT
- USBPD_NOTIFY_DATA_MSG_SENT
- USBPD_NOTIFY_GET_SRC_CAP_EXT_RECEIVED
- USBPD_NOTIFY_SRC_CAP_EXT_RECEIVED
- USBPD_NOTIFY_SRC_CAP_EXT_SENT
- USBPD_NOTIFY_GET_PPS_STATUS_RECEIVED
- USBPD_NOTIFY_GET_PPS_STATUS_SENT
- USBPD_NOTIFY_PPS_STATUS_RECEIVED
- USBPD_NOTIFY_PPS_STATUS_SENT
- USBPD_NOTIFY_GET_STATUS_RECEIVED
- USBPD_NOTIFY_STATUS_RECEIVED
- USBPD_NOTIFY_STATUS_SENT
- USBPD_NOTIFY_ALERT_RECEIVED
- USBPD_NOTIFY_VDM_IDENTIFY_RECEIVED

- USBPD_NOTIFY_VDM_CABLE_IDENT_RECEIVED
- USBPD_NOTIFY_VDM_SVID_RECEIVED
- USBPD_NOTIFY_VDM_MODE_RECEIVED
- USBPD_NOTIFY_REQUEST_EXIT_MODE
- USBPD_NOTIFY_REQUEST_EXIT_MODE_ACK
- USBPD_NOTIFY_REQUEST_EXIT_MODE_NAK
- USBPD_NOTIFY_REQUEST_EXIT_MODE_BUSY
- USBPD_NOTIFY_MSG_NOT_SUPPORTED
- USBPD_NOTIFY_POWER_STATE_CHANGE
- USBPD_NOTIFY_REQUEST_DISCARDED
- USBPD_NOTIFY_AMS_INTERRUPTED
- USBPD_NOTIFY_ALERT_SENT
- USBPD_NOTIFY_CABLERESET_TX
- USBPD_NOTIFY_PE_DISABLED
- USBPD_NOTIFY_GET_SNK_CAP_EXT_RECEIVED
- USBPD_NOTIFY_SNK_CAP_EXT_SENT
- USBPD_NOTIFY_SNK_CAP_EXT_RECEIVED

Appendix C TRACER_EMB configuration

```
/**

*************************************************************************
****
  * @file    usbpd_devices_conf.h
  * @author  MCD Application Team
  * @brief   This file contains the device define.

*************************************************************************
****
  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2017 STMicroelectronics.
  * All rights reserved.</center></h2>
  *
  * This software component is licensed by ST under Ultimate Liberty license
  * SLA0044, the "License"; You may not use this file except in compliance
with
  * the License. You may obtain a copy of the License at:
  *                              www.st.com/SLA0044
  *

*************************************************************************
****
  */

#ifndef TRACER_EMB_CONF_H
#define TRACER_EMB_CONF_H

#ifdef __cplusplus
extern "C" {
#endif

/* Includes ---------------------------------------------------------------
---*/
#include "stm32g0xx_ll_gpio.h"
#include "stm32g0xx_ll_rcc.h"
#include "stm32g0xx_ll_usart.h"
#if defined(LPUART1)
#include "stm32g0xx_ll_lpuart.h"
#endif /* LPUART1 */
#include "stm32g0xx_ll_bus.h"
#include "stm32g0xx_ll_dma.h"
```

```
/* Private typedef -----------------------------------------------------
---*/
/* Private function prototypes --------------------------------------------
---*/
/* Private functions ------------------------------------------------------
---*/
/* Private define ---------------------------------------------------------
---*/

/* -------------------------------------------------------------------------
-----
      Definitions for TRACE feature
---------------------------------------------------------------------------
-----*/
#define TRACER_EMB_BAUDRATE                          921600UL

#define TRACER_EMB_DMA_MODE                          1UL
#define TRACER_EMB_IT_MODE                           0UL

#define TRACER_EMB_BUFFER_SIZE                       1024UL

/* -------------------------------------------------------------------------
-----
      Definitions for TRACE Hw information
---------------------------------------------------------------------------
-----*/

#define TRACER_EMB_USART_INSTANCE                    LPUART1
#define TRACER_EMB_TX_GPIO                           GPIOA
#define TRACER_EMB_TX_PIN                            LL_GPIO_PIN_2
#define TRACER_EMB_TX_AF                             LL_GPIO_AF_6
#define TRACER_EMB_TX_GPIO_ENABLE_CLOCK()
LL_IOP_GRP1_EnableClock(LL_IOP_GRP1_PERIPH_GPIOA)
```

Revision history

Table 14. Document revision history

Date	Revision	Changes
24-May-2019	1	Initial release.
01-Oct-2019	2	Updated *Section 1: Generalities* and *Section 2.2: Memory budget*. Updated *Table 2: Standards*. Updated *Figure 4: USB-PD files organization*, *Figure 10: Sink connection and detach*, *Figure 11: Source connection and disconnection* and *Figure 30: SRC generates a hard reset*.