Software package for the STEVAL-USBPD27S

Introduction

The STSW-USBPD27SFW software package contains the application source code designed to demonstrate the capabilities of the STEVAL-USBPD27S 27 W AC-DC USB-C and Power Delivery certified adapter with PPS functionality (TID: 5445).

The application firmware is designed to run on the mainstream ARM® Cortex®-M0+ 32-bit STM32G071KBU6N microcontroller integrated in the STEVAL-USBPD27S and embeds the USB-PD middleware stack coming from the STM32CubeG0 firmware package, allowing the solution to be certified according to the USB Type-C 2.0 and Power Delivery 3.1 specifications. The STM32G071KBU6N is certified as PD controller (TID: 5444).

The STSW-USBPD27SFW embeds three proprietary software IPs (as compiled libraries) that allow the STM32G071KBU6N microcontroller to manage the synchronous rectification (SR) mechanism and to operate on the output voltage/current to cope with the Programmable Power Supply feature. Thus, the microcontroller acts as synchronous rectification manager and USB Power Delivery controller, maximizing the power conversion efficiency and reducing the system-level power consumption and the BOM components. These requirements, together with the microcontroller low power mode, allow a low standby power consumption in line with the energy efficiency regulation (CoC Tier 2 and DoE Level VI).

The application firmware enables the adapter to deliver two fixed PDOs (5 V at 5 A, 9 V at 3 A), properly managing the constant voltage (CV) mechanism, and two APDOs (5 V Prog at 5 A and 9 V Prog at 3 A), adjusting the output voltage with 20 mV steps in CV mode and the output current with 50 mA steps in constant current (CC) mode.

1 Overview

The STSW-USBPD27SFW software package features:

- USB-PD middleware stack based on STM32CubeG0 STM32Cube MCU Package for STM32G0 series running on the ARM® Cortex®-M0+ 32-bit STM32G071KBU6N microcontroller
- Proprietary IPs included: Synchronous Rectification, V_{BUS} Control Algorithm (for PPS management) and Low Power Manager

RELATED LINKS

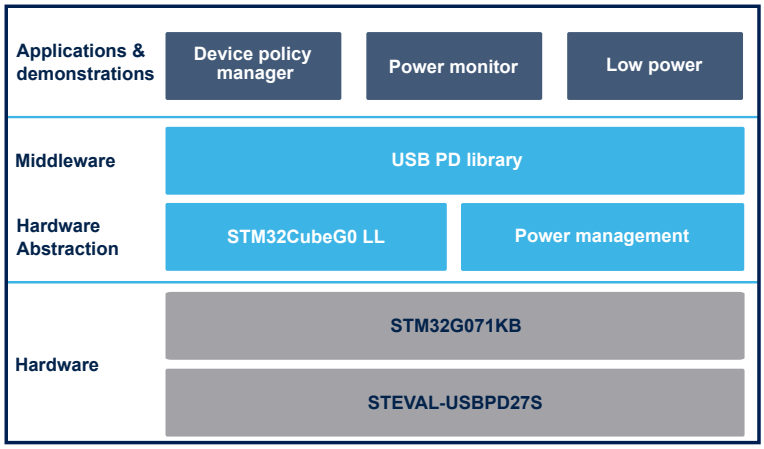
UM2552:" Managing USB power delivery systems with STM32 microcontrollers"

Visit the wiki page for relevant guides and resources regarding USB Power Delivery

2 Architecture

The STSW-USBPD27SFW architecture is organized in different levels, as shown in the following figure.

Figure 1. STSW-USBPD27SFW software architecture



1. **Hardware abstraction**
   - **STM32CubeG0 LL** - low level device libraries specific for the STM32G0 microcontroller
   - **Power Management** - a set of modules, libraries and blocks to manage the power and the bus. The main functionalities are:
     - **V_{BUS} Control** block
     - **V_{CONN} Control** block
     - **PPS compiled library**
     - **SR compiled library**
2. **Middleware**
   - **USB PD Library** - its core is provided as a compiled library, containing the ST USB-PD middleware stack main blocks:
     - **Policy Engine** - to implement the local policy for a specific USB power delivery port
     - **Protocol Layer** - to enable messages to be exchanged between a source port and a sink port
     - **Physical Layer** - to handle transmission and reception of bits on the wire and data transmission
     - **USB-C port Control** - to handle the type C detection state machines
3. **Applications & demonstrations**
   - **Device Policy Manager** - to manage USB power delivery resources within the device across one or more ports based on the device local policy
   - **Power Monitor** - to monitor, at higher level, the bus status acquiring the voltage-current pair and reaching events (notifications and faults)
   - **Low Power** - to implement low-power strategies to allow the solution to safely turn off and turn on MCU execution saving power without losing the application control

The software architecture exploits the characteristics of FreeRTOS that embeds a scheduler and a task organization.

The application project, containing source files, linker configuration files, microcontroller startup file, etc., has been provided for the following IDEs: STM32CubeIDE, EWARM (IAR) and µVision (Keil).

RELATED LINKS

UM2552:" Managing USB power delivery systems with STM32 microcontrollers"

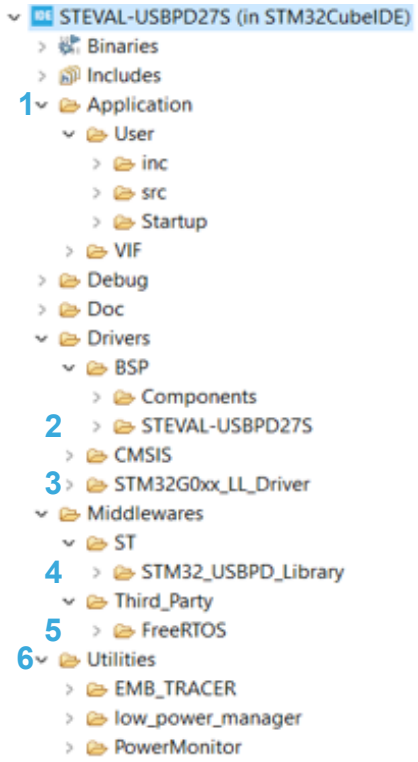
4 Workspaces on page 15

3 Project folder structure

The following figure shows the project main file organization and the related folder structure.

Figure 2. Project folders and file organization

1. Application user files
2. BSP and modules
3. Low level drivers
4. USBPD stack library
5. FreeRTOS source code
6. Utilities collection

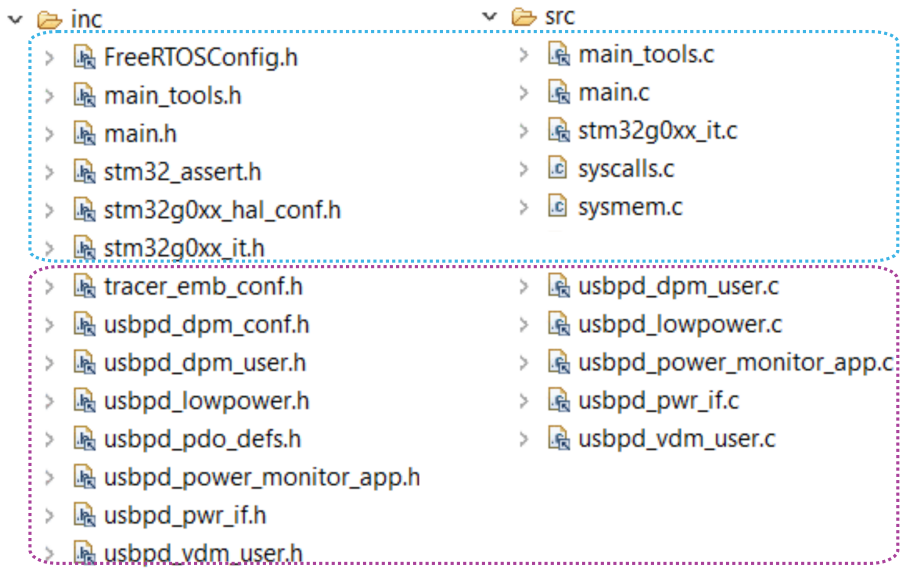


3.1 Application

The application user code is divided in:

- application files, containing main and system files (highlighted with dotted blue lines in the figure below)
- USBPD user files (highlighted with dotted purple lines in the figure below) dedicated to user settings and library configuration

Figure 3. **Application folders**



3.1.1 Main and system files

Table 1. **Main and system files**

File name	Description
Main.c / .h	Firmware application entry point that allows managing the hardware/firmware configuration and the boot sequence
Main_Tools.c / .h	Service file for the main user application. It contains the firmware version and description, error application handler and boot sequence messages
FreeRTOSConfig.h	FreeRTOS configuration header file
stm32g0xx_it.c / .h	Interrupt routine service file
system_stm32g0xx.c	System package file (1)

1. *For further details refer to STM32Cube.*

3.1.2 USBPD user files

These user application files configure the USBPD library provided with the STM32Cube MCU Package for STM32G0 and implement the required callbacks/handlers.

Table 2. **USBPD user files**

File name	Description
usbpd_dpm_conf.h	USBPD DPM configuration file used to enable the Type-C port, VID, PID, etc.
usbpd_dpm_user.c / .h	DPM user implementation files containing all notification callbacks from and required by USBPD stack library core
usbpd_pdo_defs.h	PDO definitions for both normal (max 3 A) and full featured (max 5 A) cables
usbpd_pwr_if.c / .h	Power interface files containing all APIs to turn on/off the V_{BUS} and V_{CONN}, to set a new profile and check the bus status
usbpd_vdm_user.c / .h	USBPD VDM user implementation files containing the VDM module initialization needed for cable discovery actions
usbpd_lowpower.c / .h	Low power monitor entry-point, implementing functions and callbacks for low power strategy fulfillment

File name	Description
usbpd_power_monitor_app.c / .h	Power monitor module entry-point to implement callbacks and application strategy to manage power, faults or critical conditions
tracer_emb_conf.h	Embedded Tracer configuration file

3.1.2.1 *USBPD DPM User*

The USBPD DPM user is split into:

1. functions called from the USBPD stack, used to configure and notify USB Type-C and Power delivery events

Table 3. USBPD stack callbacks

Function name	Description
`USBPD_DPM_UserInit`	DPM user initialization
`USBPD_DPM_UserExecute`	DPM user task to manage alerts
`USBPD_DPM_UserCableDetection`	Cable notification handler, managing attach/detach, cable type discovery
`USBPD_DPM_UserTimerCounter`	DPM timing management
`USBPD_DPM_WaitForTime`	Implementation of the delay used in the USBPD core; this application uses the osDelay provided by the CMSIS
`USBPD_DPM_SetupNewPower`	Interface for power requests coming from USBPD core
`USBPD_DPM_HardReset`	Hard reset state machine callback implementation, used to manage the BUS and related messaging during the HR procedure
`USBPD_DPM_EvaluatePowerRoleSwap`	Evaluation of the power role swap, always rejected in case of source
`USBPD_DPM_Notification`	Callback to handle the notification provided by the PE, i.e. when an explicit contract is reached
`USBPD_DPM_ExtendedMessageReceived`	Entry-point for incoming extended messages
`USBPD_DPM_GetDataInfo`	Entry-point to retrieve DPM data/configuration
`USBPD_DPM_SetDataInfo`	Entry-point to set DPM data/configuration
`USBPD_DPM_EvaluateRequest`	Ensure local policy evaluating requests from the sink partner that can be rejected or accepted
`USBPD_DPM_EvaluateVconnSwap`	Evaluation of V_{CONN} swap requests
`USBPD_DPM_EvaluateDataRoleSwap`	Evaluation of data role swap requests
`USBPD_DPM_IsPowerReady`	Check if the BUS is stable

2. miscellaneous callbacks and functions, containing service functions and generic callbacks available as USBPD DPM APIs.

Table 4. Miscellaneous callbacks and functions

Function name	Description
USBPD_DPM_Calib_GetFromFlash	To get calibration parameters from the Flash memory
USBPD_DPM_Calib_Align	To set the calibration parameter for the power layer
DPM_TurnOffPower	To turn the power off
DPM_TurnOnPower	To turn the power on
DPM_PWRMON_ThresholdOutOfRangeCallback	Callback to notify out of range condition to the DPM, used for overcurrent condition directly caught from the current/voltage sensing device
DPM_PWRMON_ProtectionAlertCallback	Callback to manage alerts from the protection device

3. wrapper to PE messaging functions, which a set of functions allowing the user application to send specific USBPD control or data messages to the port partner (i.e. USBPD_DPM_RequestGotoMin to go to the minimum to pair port).

3.1.2.2 *USBPD Low Power*

The USBPD user section also contains the *usbpd_lowpower* files, which include all the interface functions and some utilities, and allow low power operation through the Low Power Manager utility IP.

Table 5. Low Power APIs

Function name	Description
USBPD_LOWPOWER_SetCbs	To store a callback into the system Clock Configuration function, which is a mandatory step when the MCU exits the Stop 0 mode
USBPD_LOWPOWER_LPTIM_IRQ_cb	Callback called by the IRQ of the low power mode timer
USBPD_LOWPOWER_WaitToEnter	Utility function to allow the user to specify a time before entering low power mode permitting, for instance, the system to run for a specified time
USBPD_LOWPOWER_CanEnter	Utility function to know whether low power mode operation is allowed

All the information managed by these functions, as well as other collateral relative data, are stored in the OnLineSetup variable that holds time-related information.

The core function is USBPD_LPM_vApplicationSleep which joins FreeRTOS OS functionalities with the Low Power Manager utility module and is called by the OS each time the scheduler has to wait for task execution.

Before starting low power operation, the following checks are performed:

- the idle time must be greater than the threshold specified by the user
- the waiting time (set by the user via the USBPD_LOWPOWER_WaitToEnter call) must be expired

If the checks are successful, the SysTick timer is stopped and low power mode is activated. A timer (LPM_TIM) works with the scheduler through a dedicated interrupt.

Two events can wake the system up: an idle time expiration occurred (LPM_TIM IRQ), a USB Type-C/PD event (i.e., attach/detach, message incoming) or one of the several interrupts related to the normal application operations (i.e., fault conditions) are identified. To speed the operation up, the LPM_TIM IRQ_Handler function updates a flag, available in the OnLineSetup variable, while the low power manager utility module handles the timing aspects.

After any interrupt, the system wakes up and the LPM_OnExit_UpdateTime is called. If the system is woken up by the *LPM_TIM* IRQ (i.e., the programmed idle time has expired), no other timing update must be performed. Otherwise, the *LP_TIM* counter is used to get a timing value as much close as possible to the real wakeup time. The LPM_OnExit_UpdateTime function immediately stores the *LP_TIM* counter value.

Two main strategies have been implemented:

- when no sink is attached to the STEVAL-USBPD27S Type-C port, the microcontroller enters in Stop mode, after a specified amount of time, allowing the system to reach the lowest standby consumption. If no sink is attached, the power stage is driven to a voltage level ensuring the lowest consumption. This action as well as MCU Stop mode management are performed by the `LPM_EnterStopMode` and `LPM_ExitStopMode` functions
- when the source has a pair device connected and an explicit contract is achieved, the microcontroller enters in Sleep mode until a new event occurs

The `USBPD_LOWPOWER_WaitToEnter` function is also used to avoid undesired system behavior, keeping it running when needed. If a new message is received, an interrupt wakes the system up, to process the information received and its DPM sends a message to the sink; in this context, the system must not enter in low power mode until it completes message exchanges and power settings.

Table 6. USBPD_LOWPOWER_WaitToEnter function default values

Name	Value [ms]	Use
`TIME_TO_WAIT_LPM_STARTUP`	5000	At startup or after System Reset After any Attach/Detach event
`TIME_TO_WAIT_AFTER_CHANGE`	2000	After all application interrupts (except *LPM_TIM*)
`TIME_TO_WAIT_LPM`	1000	After any Explicit Contract is reached

3.1.2.3 *USBPD Power Monitor*

The power monitor application defines the callbacks necessary in the Power Monitor module. All static functions are stored in a custom array file and transmitted to the module during the initialization.

```
PM_Callback_Typedef PM_Callbacks = {
            .PM_ReadData               = PM_ReadData_Handler,
            .PM_NotifyData             = PM_NotifyData_Handler,
            .PM_CheckStatus            = PM_CheckStatus_Handler,
            .PM_FaultCondition         = PM_FaultCondition_Handler,
            .PM_CriticalCondition      = PM_CriticalCondition_Handler,
};
```

The first two functions (`PM_ReadData_Handler` and `PM_NotifyData_Handler`) are related to the Control Task running each 2 ms with a strong deadline. The other functions are used for Monitor Task which checks the system status, receives events (without latency) and notifies fault and critical conditions at user level. In this module, no direct action on the power is performed.

Table 7. Power monitor module callbacks and tasks

Task	Callback	Description
Control Task (task at 2 ms)	`PM_ReadData_Handler`	Reads V_{BUS} and I_{BUS} data from SPI and
	`PM_NotifyData_Handler`	Notifies the data to the module (called after reading)
Monitor Task (task at 100 ms and events)	`PM_CheckStatus_Handler`	Checks the status callback and the whole system, implements the overcurrent/ PGood control strategy and returns *ERR* or *OK*. This callback is designed to perform periodic actions
	`PM_FaultCondition_Handler`	Fault condition callback, called in case of *ERR* and to solve and manage faults. If this function returns *ERR*, the critical condition is notified. The system is moved into safe mode (no V_{BUS}) and the USBPD connection is blocked for 8 seconds.
	`PM_CriticalCondition_Handler`	Critical condition callback to put the system in safe mode and require a power cycle to start again

3.1.3 Vendor information file (VIF)

The vendor information file (VIF) used to test the solution has been also included in the application folder.
The file contains all the board setup information to correctly run the test activity with USB-IF compliance test tools.
VIF has been included in the software package to ease testing the solution for further customization.

RELATED LINKS

For further details on test results carried out with the USB-IF compliance tools, refer to AN5563: "STEVAL-USBPD27S performance"

3.2 Drivers

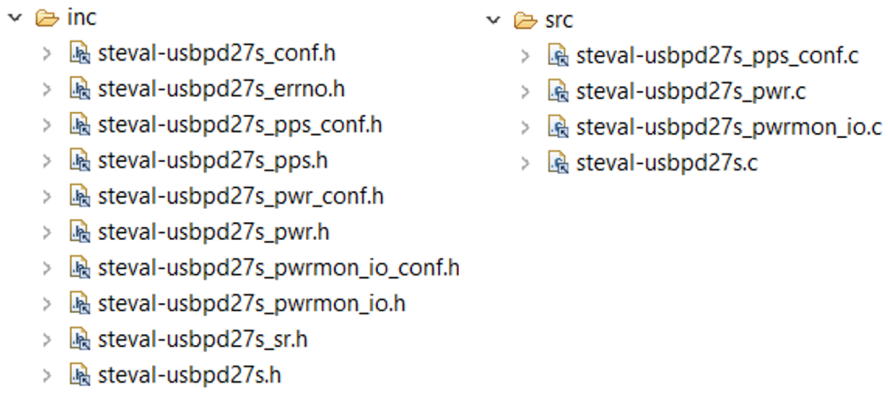
The Drivers folder includes:

- STM32G071KBU6N low level (LL) drivers to configure the peripheral devices such as I/O ports, interrupts, timers and communication
- STEVAL-USBPD27S board support package (BSP)
- other modules to better exploit the hardware platform functionality
- IP libraries specifically designed to support synchronous rectification (SR) and programmable power supply (PPS) both implemented on the board

3.2.1 STEVAL-USBPD27S BSP

Each module manages a specific feature of the system, creating a software abstraction of the available hardware.

Figure 4. STEVAL-USBPD27S BSP files



3.2.1.1 *Configuration files*

Table 8. STEVAL-USBPD27S BSP Configuration files

File name	Description
steval-usbpd27s.c / .h	Main entry file and common definition, responsible for the whole initialization of the layer, it includes the LED APIs too.
steval-usbpd27s_conf.h	Pin-out configuration and peripheral identification for the main file: the USBC peripheral, the GPIO used.
steval-usbpd27s_errno.h	Definition of the error type

3.2.1.2 *Power management (V_{BUS} and V_{CONN} control modules)*

The application firmware massively uses the power functions contained in this module to:

- configure the dedicated peripherals
- manage voltage and current and control the V_{BUS} and the V_{CONN}
- monitor the power managed in the V_{BUS} path
- collect the interrupts from the protection mechanisms

These functions allow to the entire solution to be compliant with the USB PD specification.

Table 9. STEVAL-USBPD27S BSP Power files

File name	Description
steval-usbpd27s_pwr.c /.h	Used for power management and split in four sections: V_{BUS}, V_{CONN}, Monitor and Protection
steval-usbpd27s_pwr_conf.h	Pin-out configuration and peripheral identification of the power management file
steval-usbpd27s_pwrmon_io.c / .h	Voltage/current sensing driver in/out implementation functions
steval-usbpd27s_pwrmon_io_conf.h	Voltage/current sensing driver pin-out configuration and I^2C peripheral identification

3.2.1.3 *SR module*

The STEVAL-USBPD27S exploits the ST adaptive synchronous rectification IP algorithm to drive the mechanism developed on the hardware primary side.

Table 10. STEVAL-USBPD27S BSP Synchronous Rectification files

File name	Description
steval-usbpd27s_sr.h	This file contains the API that interacts with the SR module

To correctly run the SR algorithm, this module manages the following functions:

- `BSP_SR_Init`: to initialize the hardware needed and the associated software structures
- `BSP_SR_Enable`: to enable the A-SR algorithm
- `BSP_SR_Disable`: to disable the A-SR algorithm
- `BSP_SR_Update`: to periodically update the A-SR parameter to cope with the different load conditions

The A-SR algorithm is enabled or disabled by two related functions, when a sink is attached or detached, respectively. These actions are performed by the `USBPD_DPM_UserCableDetection` callback in the usbpd_dpm_user.c file.

The `BSP_SR_Update` function takes as input the current flowing into the sink, provided by the Power Monitor module, to update A-SR parameters according to the actual load. It is called by the `PM_Monitor_Task` function.

RELATED LINKS

For further details on adaptive synchronous rectification algorithm, refer to AN5499

3.2.1.4 *V_{BUS} Control Algorithm for PPS management (PPS module)*

Thanks to a dedicated control algorithm and a PI controller, provided in a compiled library; the STEVAL-USBPD27S manages the incoming requests, controlling the V_{BUS} output to be in line with USB PD and PPS specifications.

Table 11. STEVAL-USBPD27S BSP PPS files

File name	Description
steval-usbpd27s_pps.h	This file contains the API that interacts with the PPS module

File name	Description
steval-usbpd27s_pps_conf.c / .h	This file contains the parameters and the structure to configure the PPS module

The PPS module contains the following APIs:

- `BSP_PPS_Init`: to initialize the hardware needed and the associated software structures
- `BSP_PPS_Reset`: to re-initialize the hardware needed and the associated software structures
- `BSP_PPS_VoltageFixed`: to notify a fixed request
- `BSP_PPS_VoltageAPDO`: to notify a programmable request
- `BSP_PPS_VoltageControl`: control function called at 2 ms (from the control task), with the V_{BUS}/I_{BUS} parameters
- `BSP_PPS_GetPGood`: returns *Ok* if the V_{BUS} is valid, *Fail* otherwise and *Not_valid* during boot, transition and in Current Limit mode.
- `BSP_PPS_GetOMF`: to get the operating mode flag (OMF) condition
- `BSP_PPS_RegisterCallback`: to register the callback

You can configure all the involved parameters (timing, delay, counting configuration, etc.) contained in the .h conf file.

Table 12. PPS configuration parameters

Parameter name	Description
`ControlDelay`	Control timing (ms)
`ControlDisableCount`	Default tick delay to start controlling
`ControlDisableCountMax`	Max. tick delay to start controlling
`MobileAvgOrder`	Order of the mobile average
`RsinkCheckWait`	Tick waiting for stable R_{sink}
`IBusCCThreshold`	I_{BUS} CC threshold
Vbus configuration and tolerance	
`VbusDeltaEnabler`	Generic V_{BUS} delta to enable the control (mV)
`VBusDeltaOMF`	In Current Limit mode, V_{BUS} delta to enable the control (mV)
`VbusTolerancePerc`	Tolerance percentage
Thresholds	
`VBusUVThreshold`	Undervoltage V_{BUS} threshold (mV)
`VBusUVThresholdValidity`	Undervoltage V_{BUS} threshold to validate check (mV)
`VBusOVThreshold`	Overvoltage V_{BUS} threshold (mV)
V_{ref} and Power	
`VrefNominal`	Nominal V_{ref} (mV) (default 5000 mV)
`PowerNominal`	Nominal power (mW)
`PowerLimitation`	Power tolerance (mW)
PI Control	
`PIParamKp`	Kp parameter
`PIParamKi`	Ki parameter
`PIParamLowerLimit`	Lower limit value
`PIParamUpperLimit`	Upper limit value

Parameter name	Description
Duty configuration	**Description**
DutyMax	Max. duty cycle
Duty5Vnominal	Nominal 5 V duty cycle
Duty9Vnominal	Nominal 9 V duty cycle
DutyStepValue	V_{BUS} step value (mV) on duty calculation
DutyStepCount	Number of the step value
DutySteps	Duty step table

The control loop reacts to all the external changes maintaining the correct BUS level to cope with all application needs even managing PPS profiles when, especially in Current Limit mode, the algorithm must ensure 20 mV and 50 mA of granularity on BUS voltage and current, respectively.

RELATED LINKS

For further details on ST VBUS control algorithm for PPS management, refer to AN5562: "VBUS control algorithm compliant with USB Type-C and Power Delivery specifications"

3.2.2 CMSIS

The Cortex Microcontroller Software Interface Standard (CMSIS) driver library contained in this directory is the hardware abstraction layer for microcontrollers based on Arm® Cortex® processors. It enables device support and software interfaces for the processor and its peripherals.

For this software package, CMSIS provides the RTOS services as wrapper of FreeRTOS.

RELATED LINKS

For further details on CMSIS Software Packs available in STM32CubeMX, refer to UM1718:"STM32CubeMX for STM32 configuration and initialization C code generation"

3.2.3 STM32G071KB low level (LL) drivers

The LL drivers contain APIs for a set of STM32 peripherals.

LL APIs are hardware services that reflect the hardware capabilities and provide operation functions that must be called by following the programming model described in the product line reference manual.

RELATED LINKS

For further details on STM32G0 LL Drivers, refer to UM2319: "Description of STM32G0 HAL and low-layer drivers"

3.3 Middleware

3.3.1 USB-PD library

USB-PD library is the Middleware stack hosted in the STM32CubeG0 MCU expansion package. It consists of libraries, drivers, sources, APIs and application examples running on any STM32 32-bit microcontrollers.

RELATED LINKS

For further details on USB PD on STM32G0 microcontroller, refer to UM2552:" Managing USB power delivery systems with STM32 microcontrollers"

3.3.2 FreeRTOS

FreeRTOS is a widely known real-time operating system (RTOS) for microcontrollers and small microprocessors. It offers many APIs to work with tasks, queues, semaphores, etc. as well as scheduler functions.

The application is set on several tasks defined in different layers that perform specific functions: at USBPD stack level, the most relevant tasks are Type-C cable detection and Policy Engine states machine management.

A further task has been created to manage alerts at DPM level.

Other two tasks are defined in the Power Monitor module to implement the V_{BUS} control and system monitoring.

3.4 Utilities

The Utilities folder contains the Power Monitor, the Low Power Manager and the Embedded Tracer modules.

3.4.1 Power monitor

Table 13. Power monitor files

File name	Description
usbpd_power_monitor.c / .h	Power monitor module

This FreeRTOS-based module is contained in the usbpd_power_monitor file pair. It monitors power and is connected to other layers through callbacks.

Users can customize the actions performed using usbpd_power_monitor_app file pair.

There are two main tasks running:

- `PM_Control_Task`: a high priority task that reads data operation (`PM_ReadData_Handler` callback) and, consequently, notifies the data collected (`PM_NotifyData_Handler` callback).

  The data collected are the BUS voltage and current; once read, they are compensated by a dedicated device through I²C bus. The device calculates the calibrated values for overall system errors using calibration parameters (gain and offset). Then, it calculates the calibrated values and stores them into the corresponding dedicated field host in the `PM_Handle` (ready to be shared with the other part of the system).
- `PM_Monitor_Task`: runs periodically or on event-driven actions, to get USBPD status notifications (i.e., attach/detach events, explicit contract) or faults (i.e., overcurrent, overvoltage).

When running periodically, it recalls the `PM_CheckStatus_Handler` external function to perform a system check: in case of error, a fault event is generated.

3.4.2 Low power manager

Table 14. Low power manager files

File name	Description
low_power_manager.c / .h	Low power manager module

This module is contained in the low_power_manager file pair and manages the MCU low power operation according to application needs.

By default, the microcontroller is in Run mode after a system or power reset. Several low power modes are available when the CPU does not need to be kept running, for example when waiting for an external event.

Three low power modes (LPMs) have been selected for USB-PD applications: Sleep, Stop 0 and Standby.

Inside the module, the `LPM_GetMode_t` typedef represents the allowed modes: `LPM_SleepMode`, `LPM_StopMode` and `LPM_OffMode`.

By default, the Sleep mode is used when the module is requested to put the application in low power condition. Other LPMs can be activated and deactivated with dedicated APIs: `LPM_SetStopMode` and `LPM_SetOffMode`. LPMs information is stored inside the `Modes` variable which is also responsible for storing the information of the LPM request source, identified by an ID, using the `LPM_Id_t` typedef.

This module core function is the `LPM_EnterLowPower` that prepares the timer resource used to wake the system up (if needed), enters low power mode and, after wakeup, updates all the system timings.

This function is called by the application layer. You have to implement the functions interfacing with the hardware like timings or MCU core setup as well as OS interface.

The functions are provided by usbpd_lowpower file pair in the application folder.

The low power manager module is based on a feature available in the FreeRTOS OS, that is the `vPortSuppressTicksAndSleep()` function that puts the MCU into LPMs when there is no FreeRTOS task performing. The system should wake up via a dedicated interrupt. This function stops the normal operations, the OS is clocked (suppresses the ticks) and enables the low power mode (sleep).

The scheduler knows when to run the next task; a tick-less operation is achieved through a dedicated timer, able to run in low power mode, triggering the next MCU wakeup, if there is no interrupt to be managed.

3.4.3 Embedded tracer

This module implements the embedded tracer based on the USBPD stack library.

Table 15. Embedded tracer files

File name	Description
tracer_emb.c / .h	Logical implementation
tracer_emb_hw.c / .h	Hardware support

RELATED LINKS

UM2552:" Managing USB power delivery systems with STM32 microcontrollers"

Visit the wiki page for relevant guides and resources regarding USB Power Delivery

3.4.4 Libraries

The compiled libraries included in the software package represents the three ST IPs featuring the STEVAL-USBPD27S solution:

- **USBPD Core Library** (available in the STM32CubeG0 package): hosting all the functions related to the USB-PD Middleware stack USB-PD Policy engine and Protocol layer.

  Path→ `$ROOT\Firmware\Middlewares\ST\STM32_USBPD_Library\Core\lib`

  – `USBPDCORE_PD3_FULL_CM0PLUS_wc32.a` → STM32CubeIDE and EWARM (wc32)

  – `USBPDCORE_PD3_FULL_CM0PLUS_Keil.lib` → µVision

- **PPS Library** v.1.0.0: including the main functions and control algorithms to allow the solution to control the V_{BUS} by properly managing the PPS feature.

  Path→`$ROOT\Firmware\Drivers\BSP\STEVAL-USBPD27S\lib`

  – *steval-usbpd27s_PPS.a* → STM32CubeIDE and EWARM (wc32)

  – *steval-usbpd27s_PPS_Keil.lib* → µVision

- **SR Library** v.1.0.0: containing the functions to drive the adaptive synchronous rectification through the mechanism implemented in the board.

  Path→`$ROOT\Firmware\Drivers\BSP\STEVAL-USBPD27S\lib`

  – *steval-usbpd27s_SR.a* → STM32CubeIDE and EWARM (wc32)

  – *steval-usbpd27s_SR_Keil.lib* → µVision

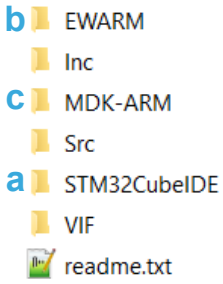
4 Workspaces

To customize and debug the application, the package supports three integrated development environments (IDEs): STMicroelectronics STM32CubeIDE, IAR EWARM and Keil µVision / MDK-ARM.

The project files are located in the application folder as shown below.

Figure 5. IDE folders

a. STM32CubeIDE– STMicroelectronics
b. EWARM – IAR
c. µVision / MDK-ARM – Keil



4.1 STM32CubeIDE

STM32CubeIDE is an advanced C/C++ development platform with peripheral configuration, code generation, code compilation, and debug features for STM32 microcontrollers and microprocessors.

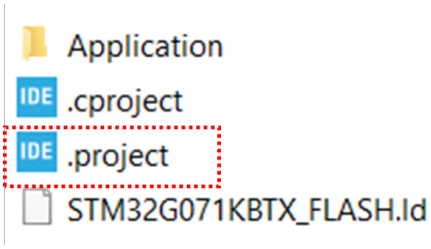
It is based on the Eclipse®/CDT framework and GCC toolchain for the development and GDB for the debugging.

To open the project, select the STM32CubeIDE folder and open the *.project* file.

The path is:

`$ROOT\Firmware\Projects\STEVAL-USBPD27S\Applications\USB_PD\STEVAL-USBPD27S\STM32CubeIDE`

Figure 6. STM32CubeIDE project location



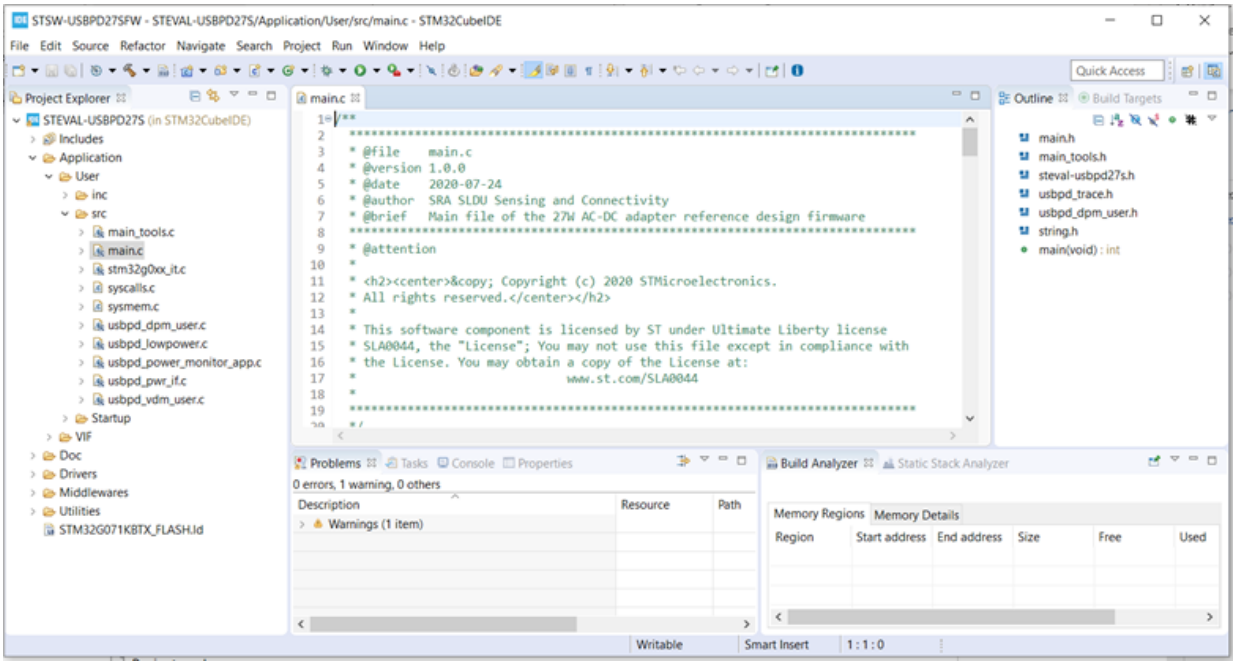
The conversion tool correctly imports the project in the STM32CubeIDE. Alternatively, you can open the environment and import the project selecting the previous folder.

If the project is successfully imported, the following message appears.

Figure 7. **Project successfully imported**

The IDE is then ready to evaluate the kit.

Figure 8. **Project imported in the STM32CubeIDE environment**



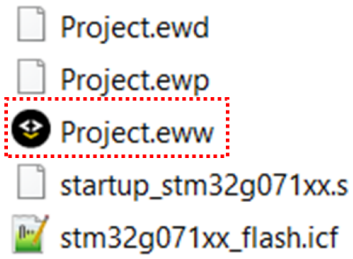
Note: The project has been tested with STM32CubeIDE v1.3.0.

4.2 EWARM - IAR

IAR Embedded Workbench is compliant with Arm embedded application binary interface (EABI) and Arm Cortex microcontroller software interface standard (CMSIS).

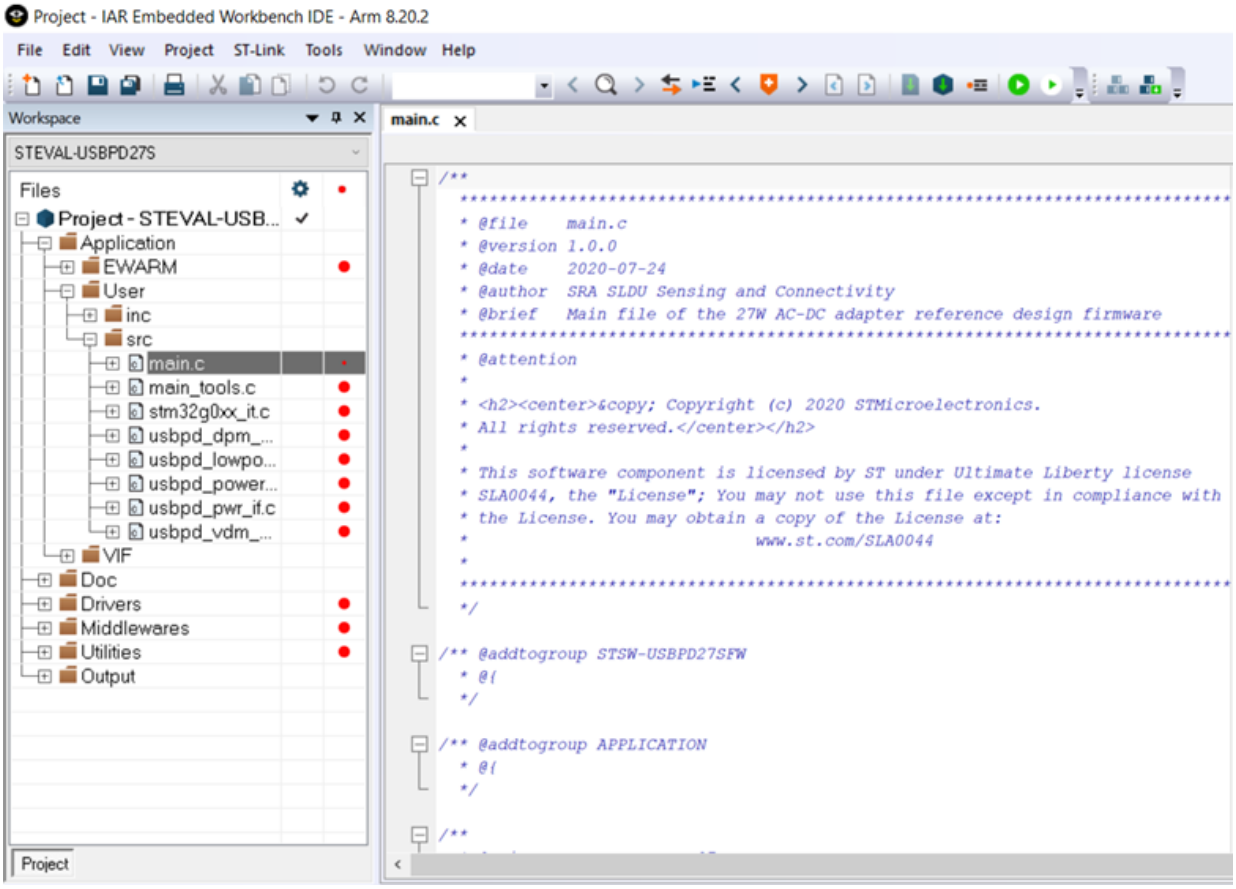
The path is: `$ROOT\Firmware\Projects\STEVAL-USBPD27S\Applications\USB_PD\STEVAL-USBPD27S\EWARM`.

Figure 9. EWARM project location



To start the evaluation, double-click the *Project.eww* file and open it.

Figure 10. Project imported in the EWARM environment



*Note: The project was tested with EWARM v8.20.2. If there are several EWARM versions in the PC, open the correct IDE version and select the Project.eww file from the **[menu]>[open workspace]**.*

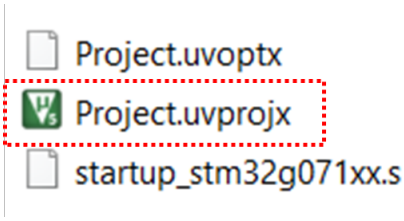
4.3 µVision/MDK-ARM – Keil

The µVision IDE and debugger is developed by Keil and supports the user in the development and debugging.

The path is:

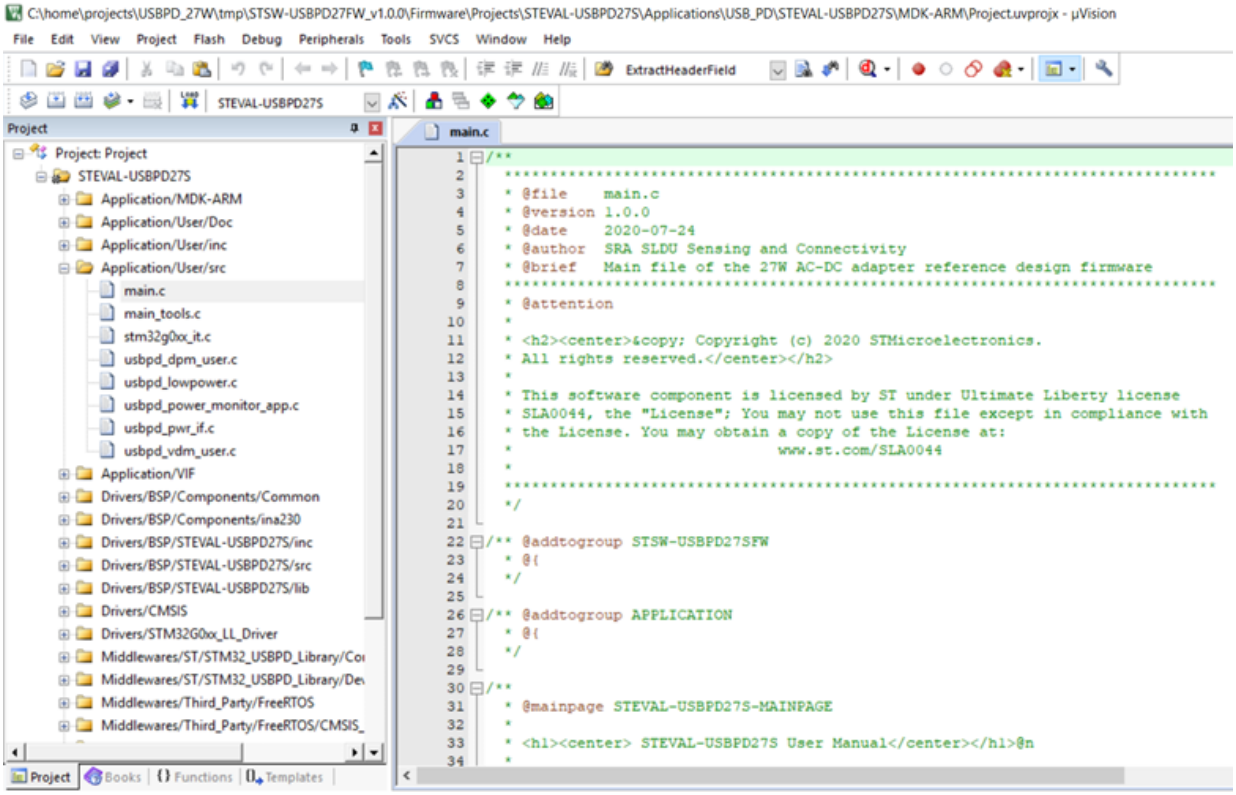
`$ROOT\Firmware\Projects\STEVAL-USBPD27S\Applications\USB_PD\STEVAL-USBPD27S\MDK-ARM`

Figure 11. µVision MDK-ARM project location



To start working with this development environment, double-click the *Project.uvprojx* file and open it.

Figure 12. Project imported in the µVision MDK-ARM environment



Note: The project has been tested with µVision v5.27.1.0 and MDK-ARM 5.31.

5 Licensing information

STSW-USBPD27SFW is delivered under the *Mix Ultimate Liberty+OSS+3rd-party V1 license*. The software components provided within this package come with different license agreements as listed in the following table.

Table 16. Software component license agreements

Software component	Owner	License
Cortex®-M CMSIS	Arm®	BSD 3-Clause
FreeRTOS™ Kernel	Copyright(C) 2017 Amazon.com, Inc. or its affiliates	MIT open source license
STM32WB HAL/LL APIs	STMicroelectronics International N.V.	BSD 3-Clause
STM32 USB-PD Library	STMicroelectronics International N.V.	Ultimate Liberty software license agreement (SLA0044)
STSW-USBPD27SFW	STMicroelectronics International N.V.	Software package license agreement (SLA0048)
STEVAL-USBPD27S BSP APIs	STMicroelectronics International N.V.	Ultimate Liberty software license agreement (SLA0044)
STEVAL-USBPD27S PPS Library	STMicroelectronics International N.V.	Production limited license agreement for ST materials (SLA0068)
STEVAL-USBPD27S Synchronous Rectification (SR) Library	STMicroelectronics International N.V.	Production limited license agreement for ST materials (SLA0068)

Revision history

Table 17. **Document revision history**

Date	Version	Changes
21-Oct-2020	1	Initial release.
06-Sep-2021	2	Updated Introduction and Section 1 Overview.

Contents

List of tables

List of figures