FUSB15201 Dual Port USB Type-C/PD Controller Software Programming Guide

FUSB15201EVBSPG

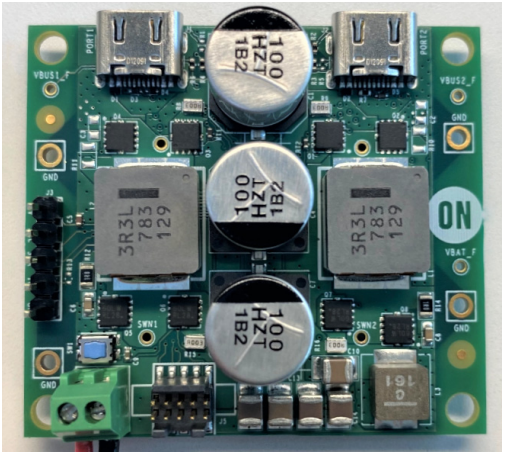

Figure 1. FUSB15201 Evaluation Board

Introduction

The FUSB15201 firmware codebase is a highly optimized Type−C/PD controller driver that supports an integrated ARM® Cortex−M0+ processor. Together with the FUSB15201 EVB, it provides customer a complete platform to evaluate the Type−C/PD solution.

The firmware codebase provides flexibility in adding or supporting new PD messages or adding additional Type−C state flow. It also allows easy modification of the hardware specific characteristic because of its Type−C/PD platform agnostic design architecture. When supplied with desired configuration, the codebase can be used in no time. The code organization offers modularity as it separates code sources for application, hardware abstraction layer, platform dependent source and Type−C/PD core.

Default configuration supports the FUSB15201 Type−C/PD device characteristic listed in Table 1. The PD core feature is configurable using project build options or by modifying the vendor info file. The codebase includes a sample eclipse project that can be ran using the eclipse based **onsemi** IDE that can be downloaded here. This allows a faster bring−up for full evaluation of Type−C/PD standalone controller

Table 1. FUSB15201 SUPPORTED DEVICE CHARACTERISTIC

Feature		Firmware (ARM Cortex−M0+)
Type−C	Source	Yes
PD	Provider	Yes

Quick Start

For the fastest bring−up, take the following steps.

1. Get the latest code release and ensure codebase has the directory structure as shown in code organization section.
2. From the **onsemi** IDE, Open File → Open Projects from File System.
   From Import Source, navigate directory to fw_fusb15200 → IDE → FUSB15201 → usbpd then select ON_IDE.
3. The project will open in **onsemi** IDE. The default project configuration allows customer to build and load the built binary to EVB for standalone evaluation with Source/Provider characteristic.

Code Organization

The code organization layout will help any firmware developer to get familiar with the code base quickly.

fw_fusb15200

−−−Application/FUSB15201/usbpd
−−−CMSIS
−−−Device
−−−Drivers
−−−External
−−−IDE/FUSB15201/usbpd/ON_IDE
−−−SVD

Table 2. FUSB15201 FIRMWARE CORE COMPONENTS

Component	Description
Application/FUSB15201/usbpd	This component contains custom specific source files including the sample Device Policy Manager. The vendor info file is also located in this folder.
Device	Platform specific source files
Drivers	Hardware abstraction Layer source files
External	Type−C/PD state machine
IDE/FUSB15201/usbpd/ON_IDE	Sample Eclipse based project

Build Configuration Option for FUSB15201 Reference Firmware

The reference firmware and its default configuration support the FUSB15201 EVB platform for a complete evaluation of Type−C/PD solution. Following the instruction in Quick Start, a firmware binary can be quickly built and loaded into the EVB. Listed in Table 3 is the FUSB15201 Supported Feature. If application requires disabling feature from the table below, simply change value from 1 to 0 as shown below.

Table 3. FUSB15201 SUPPORTED BUILD CONFIGURATION

Symbol	Value	Description
CONFIG_BC1P2_CDP	0	Disabled support for BC1P2 CDP
CONFIG_BC1P2_CSM	0	Disabled support for BC1P2_CSM
CONFIG_BC1P2_DCP	1	Enabled support for BC1P2_DCP
CONFIG_BC1P2_DCP_ADDIV	1	Enabled support for BC1P2_DCP_ADDIV
CONFIG_DRP	0	Disabled support for Dual Port Role
CONFIG_EXTMSG	1	Enabled support for extended message length
CONFIG_LEGACY_CHARGING	1	Enabled support for legacy charging
CONFIG_LOG	0	Disabled support for logging
CONFIG_SINK	0	Disabled support for sink characteristic
CONFIG_SLEEP	1	Enabled support for deep sleep
CONFIG_SRC	1	Enabled support for source characteristic
CONFIG_VDM	1	Enabled support for Vendor Define Message
FUSB15200		Define FUSB15200
HAL_USE_ASSERT		Define assertion of size check

Platform

The FUSB15201 platform integrates an ARM Cortex−M0+ processor with Nested Vector Interrupt Controller, Wake−up Interrupt Controller and Debug Access Port. The codebase includes peripheral drivers and supports for multiple external source interrupts for peripheral devices supported by the ARM Cortex−M0+ processor.

PD Device Policy Manager

The firmware codebase provides a reference code for the Device Policy Manager. The sample DPM can manage platform specific PD message requests and responses using event subscription and notification callback. It is abstracting HAL from Policy Engine or Type−C state machine direct access. The DPM manages a private structure that encapsulate TCPD (Type−C/PD) driver and Port structure.

PD Policy Engine Core

The PD Policy Engine state−machine is platform agnostic. Most PE functions are statically defined and are only accessible through the TCPD driver except for PE state−machine core function, PE state−machine enable/disable function, PE hard reset message interrupt handler. Type−C/PD core in this codebase can perform characteristic other than the ones listed in Table 1. These options are configurable as mentioned in previous section.

PE Function exposed to any platform.

```
void policy_pd_enable(struct port *port,
bool source)
```

This function enables the PE state−machine

```
void policy_pd_disable(struct port *port)
```

This function disables the PE state−machine

```
void policy_receive_hardreset(struct port
*port)
```

This function processes a hard−reset PD message received through PD controller interrupt bit.

```
void policy_engine(struct port *port)
```

This is the main function for the PE core state−machine

PD PE Message Handling

A few PE functions example is listed below. Full list of PD message handlers can be found in policy.c. These are function only accessible from wit

static void policy_state_source_get_sink_cap(struct port *port)

−PD provider request for sink capability

static void policy_state_source_give_sink_cap(struct port *port)

−PD provider respond to get sink capability request

static void policy_state_source_send_drswap(struct port *port)

−PD provider request for drswap

static void policy_state_source_evaluate_drswap(struct port *port)

−PD provider evaluate received drswap request

Type−C State Machine

Port detection on attached/detached of a Type−C device is handled inside the type−C state−machine core function. Like PD, this is also platform agnostic and all access to HAL is abstracted by the TCPD driver.

Observer Files

These files are shared between the Device Policy Manager and the Policy Engine.

`observer.c` contains the function definitions of event_subscribe, event_unsubscribe, event_notify.

`observer.h` has all the declaration of all event ID and structure definition of event context.

Registering Event Handlers

Event subscription/callback notification is used by Policy Engine and Device Policy Manager to pass on PD message request/response and/or platform specific behavior change to the Type−C/PD controller. An event is registered using the function `event_subscribe` following this format `event_subscribe(EVENT_ID, callback_handler)`. Device Policy Managers subscribes to applicable events and Policy Engine used these events to notify DPM using the function `event_notify`, following this format `event_notify(EVENT_ID, struct *tcpd_device, void *ctx)`. All events in Table 4 are defined in the observer.h. ID is declared in enumerated type and is using ##pre−processor to generate the EVENT ID with prepended EVENT_ to each ID in Table 4.

Event Handling

The PD event messaging between DPM and PE is handled with no assumption for DPM to always subscribe to the notification. This provides flexibility for DPM to only subscribe to events that are needed for the intended application.

Table 4. SUPPORTED EVENTS

Event ID	Description
TC_ATTACHED	Type−C device attached
TC_DETACHED	Type−C device detached
VBUS_REQ	VBUS value request for source
VBUS_SINK	VBUS value request for sink
VCONN_REQ	VCONN request to turn on/off sourcing
PD_DEVICE	PE notify PD device capable
PD_GET_SRC_CAP	PE notify source capability request
PD_GET_SNK_CAP	PE notify sink capability request
PD_GET_EXT_SRC_CAP	PE notify extended source capability request
PD_GET_EXT_SNK_CAP	PE notify extended sink capability request

Table 4. SUPPORTED EVENTS

Event ID	Description
PD_SNK_CAP_RECEIVED	PE notify sink capability message is received
EXT_SNK_CAP_RECEIVED	PE notify extended sink capability is received
PD_GET_BAT_CAP	PE notify get battery capability request
PD_GET_BAT_STAT	PE notify get battery status request
PD_BAT_CAP_RECEIVED	PE notify battery capability is received
PD_BAT_STAT_RECEIVED	PE notify battery status is received
PD_GET_MAN_INFO	PE notify get manufacturer info request
PD_SRC_EVAL_SNK_REQ	PE notify to evaluate sink request
PD_SNK_EVAL_SRC_CAP	PE notify to evaluate source capability
PD_CBL_ID_RECEIVED	PE notify cable ID is received on cable query
PD_GET_ALERT_REQ	PE notify to fill out alert request
PD_ALERT_RECEIVED	PE notify alert message is received
PPS_STATUS_RECIEVED	PE notify PPS status is received on PPS status request
PPS_STATUS_REQUEST	PE notify PPS status request
PPS_MONITOR	PE notify to activate PPS handling
PPS_ALARM	PE notify to set PPS alarm
ENTER_USB_REQUEST	Disabled in version 1.0.0.0.
ENTER_USB_RESPONSE	Disabled in version 1.0.0.0
ENTER_USB_RECEIVED	Disabled in version 1.0.0.0.
IDENTITY_RECEIVED	Not used
PD_STATUS	PE notify PD device status
MODE_ENTER_SUCCESS	Not used
MODE_EXIT_SUCCESS	Not used
MODE_VDM_ATTENTION	Not used
HARD_RESET	PE notify hard reset
UNSUPPORTED_ACCESSORY	PE notify for unsupported accessory attached
DEBUG_ACCESSORY	Not used
AUDIO_ACCESSORY	Not used
ILLEGAL_CBL	Not used
BIST_SHARED_TEST_MODE	PE notify BIST shared test
PD_NEW_CONTRACT	PE notify for new PD contract
DATA_RESET_ENTER	Not used
DATA_RESET_EXIT	Not used
PD_GET_FW_ID	PE notify get firmware ID request
PD_FW_INITIATE	PE notify to initiate firmware update
PD_INITIATE_RESP_SENT	PE notify firmware update response was sent

Adding new Events

While the events that are already defined may be adequate in the supported platform, if application requires more events subscription/notification between Policy Engine and Device Policy Manager, additional events can be added as needed. To add new event add a definition to the enum type Events_t in observer.h.

```
#define CREATE_EVENT_LIST(EVENT)     \
        EVENT(TC_ATTACHED)           \
        EVENT(TC_DETACHED)           \
        ..
        ..
        EVENT(NEW_EVENT_TO_BE_ADDED)
```

Changing Vendor Info File

Device Vendor information is in vif.info.h, if change is required follow the applicable steps below:

1. If info is already available in the header file, simply modify the default value to the new value.
2. If info is not yet defined in the header file, modify header file by adding the new information to applicable port, add entry to PORT_VIF_T which represent the newly added information in vif_info.c.

Examples.

1. Changing max current in PDO 4 from 20 V / 3 A to 20 V / 3.25 A in Port A

Current PDO values: max current = 300 (3 A)
max voltage = 400 (20 V)

```
#define PORT_A_SRC_PDO_VOLTAGE_4 400            // 20000 mV
#define PORT_A_SRC_PDO_MAX_CURRENT_4 300        // 3000 mA
```

New PDO values: max current = 325 (3.25 A)

```
#define PORT_A_SRC_PDO_VOLTAGE_4 400            // 20000 mV
#define PORT_A_SRC_PDO_MAX_CURRENT_4 325        // 3250 mA
```

2. Removing support for chunk extended message in Port A

Current value

```
#define PORT_A_CHUNKING_IMPLEMENTED_SOP         1
```

New Value

```
#define PORT_A_CHUNKING_IMPLEMENTED_SOP         0
```

3. Adding new entry in the vif_info.h in Port A
   a. `#define PORT_A_NEW_ENTRY 1`
   b. Then add an entry in PORT_VIF_T in Device/FUSB15200/vif_info.c

Changing Build Configuration

To change of the default build configuration, the project property must change. This configuration change applies to both Port A/B

Example:

1. Removing support for Extended Message Length. NOTE: removing this build config will disable the code at build time.
   Go to Project Properties → C/C++ General → Path and Symbol → Symbol tab under GNU C, change the value of CONFIG_EXTMSG from 1 to 0.
2. Removing support for VDM.
   In the same window, or by going to Project Properties → C/C++ General → Path and Symbol → Symbol tab under GNU C, change the CONFIG_VDM from 1 to 0.

Type−C/PD Core Version

The firmware uses a 4−digit version number. This codebase initial version is v1.0.0.0

Table 5.

Digit Representation		
W	1	Major version of the firmware code base – Type−C/PD related change
X	0	Minor version of the firmware code base – Type−C/PD related change
Y	0	Internal use
Z	0	Internal use

TCPD Driver

This driver provides abstraction to/from PE and Type−C state−machine. Neither PD nor Type−C can directly change the platform specific behavior. TCPD driver implements access to the Port HAL and other supported peripheral.

Ex. PE needs to change VBUS value for a contract being negotiated, the following code would be executed

PE −> port_vbus_src(struct port *port,,,) −> TCPD
port−>dev−>driv−>set_pd_source(port−>dev, true) − TCPD
fusb15201_set_pd_source(struct tcpd_device *dev, bool src_en) −> DPM
TCPORT_DRIVER.pd.Source(dpm−>tcpd, src_en); −> HAL Port Driver

PUBLICATION ORDERING INFORMATION

LITERATURE FULFILLMENT:
Email Requests to: orderlit@onsemi.com

onsemi Website: www.onsemi.com

TECHNICAL SUPPORT
North American Technical Support:
Voice Mail: 1 800−282−9855 Toll Free USA/Canada
Phone: 011 421 33 790 2910

Europe, Middle East and Africa Technical Support:
Phone: 00421 33 790 2910
For additional information, please contact your local Sales Representative