Cypress EZ-PD™ CCGx SDK User Guide

Revision 3.0.0

Doc. No. 002-12541 Rev. *A

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
Phone (USA): 800.858.1810
Phone (Intl): 408.943.2600
www.cypress.com

Contents

1. Introduction

1.1 USB Type-C and Power Delivery

USB Type-C is the new USB-IF standard that solves several challenges faced by today's Type-A and Type-B cables and connectors. USB Type-C uses a slimmer connector (measuring only 2.4 mm in height) to enable increasing miniaturization of consumer and industrial products. The USB Type-C standard is gaining rapid support by enabling small form-factor, easy-to-use connectors, and cables that can transmit multiple protocols. In addition, it offers power delivery up to 100 W – a significant improvement over the 7.5 W for previous standards.

1.1.1 USB Type-C Highlights

- New reversible connector measuring only 2.4 mm in height.
- Compliant with USB Power Delivery Specification, providing up to 100 W.
- Double the bandwidth of USB 3.0, increasing to 10 Gbps with SuperSpeedPlus USB 3.1.
- Combines multiple protocols in a single cable, including DisplayPort™, PCIe®, or Thunderbolt™.

1.2 EZ-PD™ Type-C Controllers

Cypress offers the EZ-PD line of Type-C controllers, which currently include four product families:

- EZ-PD™ CCG1: Industry's First Programmable Type-C Port Controller
- EZ-PD™ CCG2: Industry's Smallest Programmable Type-C Port Controller
- EZ-PD™ CCG3: Industry's Most Integrated Type-C Port Controller
- EZ-PD™ CCG4: Industry's First Dual-Port Type-C Port Controller

Visit the Cypress Type-C Controller web page for more details on these product families and a feature comparison.

1.3 CCGx SDK

The CCGx Software Development Kit (SDK) is a software solution that allows users to harness the capabilities of the CCGx Type-C controllers.

This version of the CCGx SDK supports use of the CCG3 and CCG4 parts. The following applications are supported by the SDK:

- CCG3 and CCG4 based Dual Role solutions such as PD port controllers for notebooks and desktops.
- CCG3 based power adapter port controller solution.
- CCG3 based Display Port (DP) Dongle port controller solution.
- CCG3 based Charge through Dongle (CTD) port controller solution.

The SDK provides a firmware stack compatible with Type-C and USB-PD specifications, along with the necessary drivers and software interfaces required to implement applications using the CCG3 and CCG4 controllers.

The key features for CCGx notebook port controller solution are:

- Supports USB-PD protocol based on the PD 3.0 specification for CCG3 (CYPD3125-40LQXI) and CCG4PD3 (CYPD4126-40LQXI and CYPD4226-40LQXI) parts.
- Supports USB-PD protocol based on the PD 2.0 specification for CCG4PD2 parts (CYPD4125-40LQXI and CYPD4225-40LQXI).
- Supports integrated Rp, Rd resistors on CC1/2 pins.
- Supports dead battery termination.
- Integrated system-level ESD protection for exposed pins.
- Integrated bootloader to support firmware update over I^2C.
- Over-Voltage Protection (OVP) and Over-Current Protection (OCP). Only CCG3 based solution has internal OCP support.

The key features for CCG3 power adapter port controller solution are:

- USB-PD Protocol as per PD 2.0 specification.
- USB-PD power contract negotiation as provider.
- PFET/NFET selection for producer FETs based on status of GPIO 7.
- Capability of in-system firmware upgrade through the CC in UFP mode.
- Twin firmware images to allow continued system functionality in case of flashing firmware.
- Over-Voltage Protection (OVP) and Over-Current Protection (OCP).

The key features for CCG3 Display Port Dongle port controller solution are:

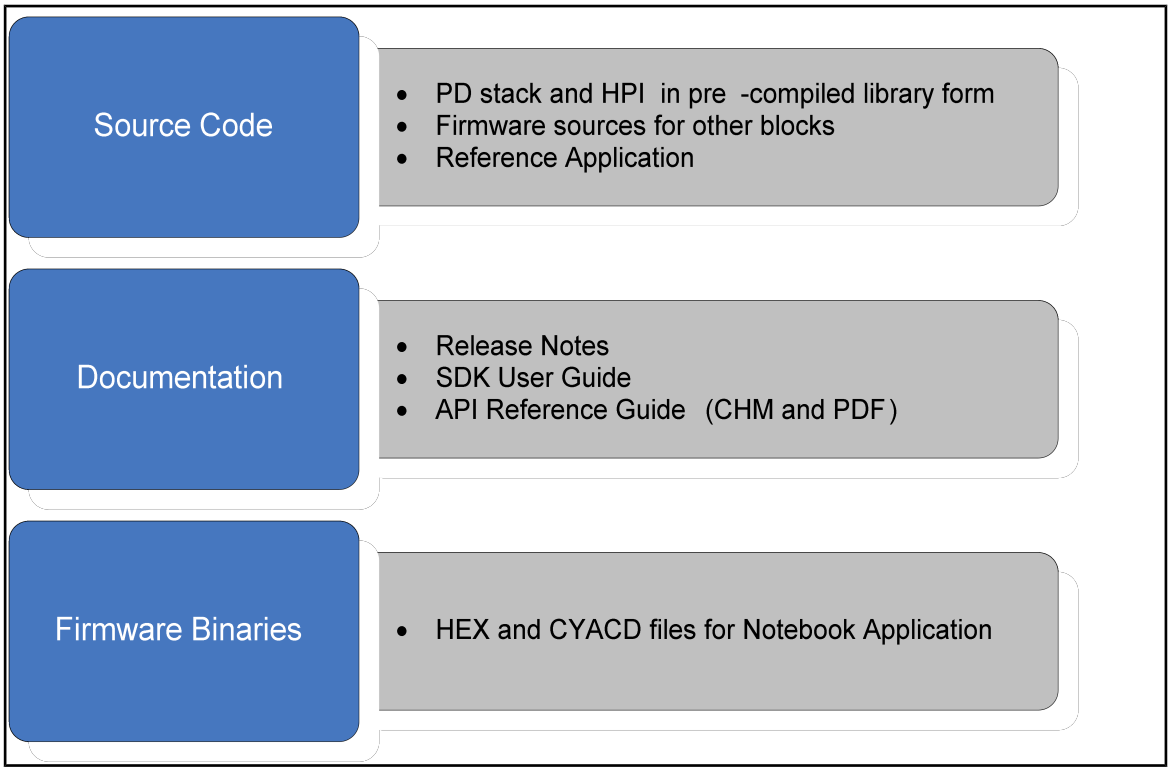
- USB-PD Protocol as per PD 2.0 specification.
- USB-PD power contract negotiation as consumer.
- Ability to function as Type-C to DP (4-Lane) or Type-C to HDMI/VGA/DVI (4-Lane) based on state of GPIO pin 23.
- Capability of in-system firmware upgrade over USB (HID Class) in firmware and bootloader mode.
- Over-Voltage Protection (OVP).
- Internal bill-board device enumeration.
- Dual firmware images to allow continued system functionality in case of flashing firmware.

The key features for CCG3 Charge Through Dongle port controller solution are:

- USB-PD Protocol as per PD 3.0 specification.
- USB-PD power contract negotiation as dual role port.
- Ability to function as Type-C to HDMI/VGA/DVI (2-Lane) Dongle.
- Capability to be producer when a power adapter is attached to down-stream CCG2 port controller.
- Capability of in-system firmware upgrade over USB (HID Class) in firmware and bootloader mode.
- Internal bill-board device enumeration.
- Dual firmware images to allow continued system functionality in case of flashing firmware.
- Capability of in-system firmware upgrade of the CCG2 port controller via the USB interface.

The CCGx SDK consists of several basic components as shown in Figure 1.

Figure 1: CCGx SDK Components



The SDK also includes reference projects implementing standard Type-C applications and documentation that guides the user in customizing existing applications or creating new applications.

SDK Directory Structure

At the top level, the following folders are present:

- **Documentation**: The Documentation folder contains the SDK documentation including release notes, user guide, and API reference guide.
- **Firmware**: The Firmware folder contains the firmware stack sources, reference projects, and pre-built firmware binaries.
  - **binaries**:
    - Pre-built CCG3 firmware binary for the CYPD3120-40LQXI part that matches the reference design documented in the CCG3 datasheet.
    - Pre-built firmware binaries for the CYPD3123-40LQXI part that control the upstream port of the Charge-Through Dongle reference application.
    - Pre-built CCG3 firmware binary for the CYPD3125-40LQXI part that matches the hardware design of the CY4531 kit.
    - Pre-built CCG3 firmware binary for the CYPD3135-40LQXI part that matches the reference design documented in the CCG3 datasheet.
    - Pre-built CCG4 firmware binaries for the CYPD4125-40LQXI, CYPD4126-40LQXI, CYPD4225-40LQXI and CYPD4226-40LQXI parts that matches the hardware design of the CY4541 kit.
  - **projects**: The projects folder contains the sources and PSoC Creator workspaces for the port controller designs based on CYPD3120-40LQXI, CYPD3123-40LQXI, CYPD3125-40LQXI, CYPD3135-40LQXI, CYPD4125-40LQXI CYPD4225-40LQXI, CYPD4126-40LQXI and CYPD4226-40LQXI.

- **lib**: The lib folder contains the USB-PD stack and Host Processor Interface (HPI) module in a pre-compiled library format.

  NOTE: This directory is made available for reference. Each reference project has a copy of the relevant libraries added to it locally.

- **src**: The src folder contains the sources for the CCGx firmware stack organized by the firmware module. Sources are provided for all modules except the core USB-PD stack and HPI. This directory is made available for reference. Each reference project has a copy of the src directory is added to it locally.

  NOTE: This directory is made available for reference. Each reference project has a copy of the src directory added to it locally.

Figure 2 shows the installed directory structure of the CCGx SDK, along with descriptions for all of the important folders.

Figure 2:CCGx SDK Directory Structure

```
EZ-PD CCGX SDK
├──CCG2                                    CCG2 SDK Directory
└──CCG3-CCG4                               CCG3 and CCG4 SDK Directory
    ├──Documentation                       Documentation: User guide, API guide etc.
    └──Firmware
        ├──binaries                        Pre-built firmware binaries
        │   ├──CYPD2122-24LQXI_ctd_ds      Binary for CYPD2122-24LQXI CTD downstream port controller
        │   ├──CYPD3120-40LQXI_dp_dongle   Binary for CYPD3120-40LQXI DP dongle port controller
        │   ├──CYPD3123-40LQXI_ctd_us      Binary for CYPD3123-40LQXI CTD upstream port controller
        │   ├──CYPD3125-40LQXI_notebook    Binary for CYPD3125-40LQXI notebook port controller
        │   ├──CYPD3135-40LQXI_power_adapter Binary for CYPD3135-40LQXI power adapter port controller
        │   ├──CYPD4125-40LQXI_notebook    Binary for CYPD4125-40LQXI notebook port controller
        │   ├──CYPD4126-40LQXI_notebook    Binary for CYPD4126-40LQXI notebook port controller
        │   ├──CYPD4225-40LQXI_notebook    Binary for CYPD4225-40LQXI notebook port controller
        │   └──CYPD4226-40LQXI_notebook    Binary for CYPD4226-40LQXI notebook port controller
        │
        ├──lib                             PD stack and HPI libraries (Reference only)
        │   ├──ccg3                        Libraries for CYPD31xx-40LQXI
        │   ├──ccg4_dualport               Libraries for CYPD42x5-40LQXI
        │   ├──ccg4_singleport             Libraries for CYPD41x5-40LQXI
        │   ├──ccg4pd3_dualport            Libraries for CYPD42x6-40LQXI
        │   └──ccg4pd3_singleport          Libraries for CYPD41x6-40LQXI
        │
        ├──projects                        Reference Projects
        │   ├──CYPD3120-40LQXI_dp_dongle   DP dongle project for CYPD3120-40LQXI part
        │   ├──CYPD3123-40LQXI_ctd_us      CTD upstream project for CYPD3123-40LQXI part
        │   ├──CYPD3125-40LQXI_notebook    Notebook project for CYPD3125-40LQXI part
        │   ├──CYPD3135-440LQXI_power_adapter Power adapter project for CYPD3135-40LQXI part
        │   ├──CYPD4125-40LQXI_notebook    Notebook project for CYPD4125-40LQXI part
        │   ├──CYPD4126-40LQXI_notebook    Notebook project for CYPD4126-40LQXI part
        │   ├──CYPD4225-40LQXI_notebook    Notebook project for CYPD4225-40LQXI part
        │   └──CYPD4226-40LQXI_notebook    Notebook project for CYPD4226-40LQXI part
        │
        └──src                             Firmware Stack Sources (Reference only)
            ├──app                         Application layer
            ├──ar_slave                    Alpine Ridge I2C slave interface
            ├──hpiss                       HPI protocol headers
            ├──pd_common                   PD stack headers
            ├──pd_hal                      PD block low level drivers
            ├──scb                         Serial communication block driver
            ├──system                      Low level drivers, firmware update, timer
            └──usb                         USB drivers (CCG3 only)
```

2. SDK Installation

2.1 SDK Installation

Once installed, the directory structure will be as shown in Figure 2Error: Reference source not found.

2.1.1 Copy the Firmware files

The firmware sources and reference projects are installed Read-Only under the Windows “Program Files” folder by default. Compiling the projects while they are located in “Program Files” may fail if User Account Control (UAC) is activated in the system. Also, it is desirable to leave the original source files untouched in case you wish to revert to a clean copy to undo any source changes you may have made.

PSoC Creator allows creating copies of the reference projects and workspaces via link in Start Page or using Code Examples Dialog. Refer to Section 3.1 for more details.

You can also make a copy of all the files under the “Firmware” folder, to create a working copy that you can modify. Make sure that the copied files are not read-only.

2.2 SDK Limitations

1. The CYPD3120-40LQXI_dp_dongle, CYPD3123-40LQXI_ctd_us and CYPD3135-40LQXI_power_adapter projects provided for CCG3 are firmware only projects. There are no hardware reference designs with schematic and BOM for these applications available yet for end customers. The pin assignments used in the CCG3 devices for these example projects closely match the reference application diagrams in the CCG3 datasheet. Cypress is working on making reference designs with schematic and BOM for these example projects available in the future.
2. CYPD3123-40LQXI parts are not yet available and so CYPD3120-40LQXI parts can be used for the same. Please note that the boot-loader is different for the project and shall require firmware download via SWD before USB flashing can be enabled.
3. CYPD4126-40LQXI and CYPD4226-40LQXI parts are pin compatible with CYPD4125-40LQXI and CYPD4225-40LQXI respectively and CY4541 kit (with the silicon replaced) can be used for evaluation.

2.3 Tool Dependencies

2.3.1 PSoC Creator

Cypress’s Type-C controllers are based on Cypress’s PSoC® 4 programmable system-on-chip architecture, which includes programmable analog and digital blocks, an ARM® Cortex®-M0 core, and internal flash memory.

The PSoC Creator IDE is used for configuring the CCG3 and CCG4 devices, to develop and compile the firmware applications and optionally to program the devices using SWD.

This version of the SDK requires PSoC Creator 3.3 DP1 (build 9674) or higher.

This version of PSoC Creator can be installed and used on a computer along with previous versions of PSoC Creator.

The PSoC Creator release includes the GNU ARM compiler tools required to compile the CCGx firmware applications.

2.3.2 EZ-PD Configuration Utility

The CCGx devices are shipped with a pre-programmed bootloader that allows the firmware on the device to be updated through an I^2C interface, the CC channel or the USB interface, which is part of the Type-C interface.

The EZ-PD Configuration Utility is a Windows-based application, which can be used to program the CCGx devices on Cypress-provided kits (DVKs and EVKs) through the bootloader interface.

The EZ-PD Configuration Utility relies on a Cypress USB controller, which can connect to the CCGx device through I^2C for programming. Therefore, it will only work with the Cypress-provided kits or other hardware, which includes the Cypress USB – I^2C bridge devices.

The EZ-PD Configuration Utility is also used for creating custom configurations for the CCGx firmware application, which includes aspects such as the supported power profiles, protection schemes, and so on.

This version of the CCGx SDK requires the latest EZ-PD Configuration Utility Beta version, which includes support for programming and configuring the CCG3 and CCG4 devices.

2.4 Hardware Dependencies

Cypress provides a set of kits, which can be used to test and evaluate the SDK functionality.

- The CY4531 EZ-PD™ CCG3 Evaluation Kit can be used to evaluate this version of the SDK with the CCG3 devices
- The CY4541 EZ-PD™ CCG4 Evaluation Kit can be used to evaluate this version of the SDK with the CCG4 devices.
- Schematics and hardware design guidelines for the Charge-Through Dongle application will be released soon by Cypress.

3. Getting Started with CCGx

3.1 Using the Reference Projects

As Error: Reference source not foundFigure 1Figure 2shows, the SDK includes reference projects for the target applications that can be used to obtain a jump-start in the process of developing a CCGx application.

This version of SDK provides the following reference projects for the following applications:

1. **CYPD3120-40LQXI_dp_dongle**: This project implements a single Type-C port controller for DP dongle application using the CYPD3120-40LQXI device. This project supports PD 2.0 specification.
2. **CYPD3123-40LQXI_ctd_us**: This project implements the upstream PD port controller for a PD 3.0 enabled Charge-Through Dongle application.
3. **CYPD3125-40LQXI_notebook**: This project implements a single Type-C port controller for notebooks using the CYPD3125-40LQXI device. This project supports PD 3.0 specification.
4. **CYPD3135-40LQXI_power_adapter**: This project implements a single Type-C port controller for power adapters using the CYPD3135-40LQXI device. This project supports PD 2.0 specification.
5. **CYPD4125-40LQXI_notebook**: This project implements a single Type-C port controller for notebooks using the CYPD4125-40LQXI device. This device supports PD 2.0 specification. The implementation supports all of the features of the dual-port project with only one difference - the target device supports only a single port.
6. **CYPD4225-40LQXI_notebook**: This project implements a dual Type-C port controller for notebooks using the CYPD4225-40LQXI device. This device supports PD 2.0 specification. Each of the ports on the device can be configured and function independently with full capabilities.
7. **CYPD4126-40LQXI_notebook**: This project implements a single Type-C port controller for notebooks using the CYPD4126-40LQXI device. This device is similar to CYPD4125-40LQXI with PD 3.0 specification support.
8. **CYPD4226-40LQXI_notebook**: This project implements a dual Type-C port controller for notebooks using the CYPD4225-40LQXI device. This device is similar to CYPD4125-40LQXI with PD 3.0 specification support.
9. Each reference project is provided in the form of a PSoC Creator workspace. The workspace can be opened using the PSoC Creator and the projects can be customized and compiled.

Note: These projects are designed to work with specific devices mentioned above. Changing the target part number using Device Selector will cause the firmware build to fail.

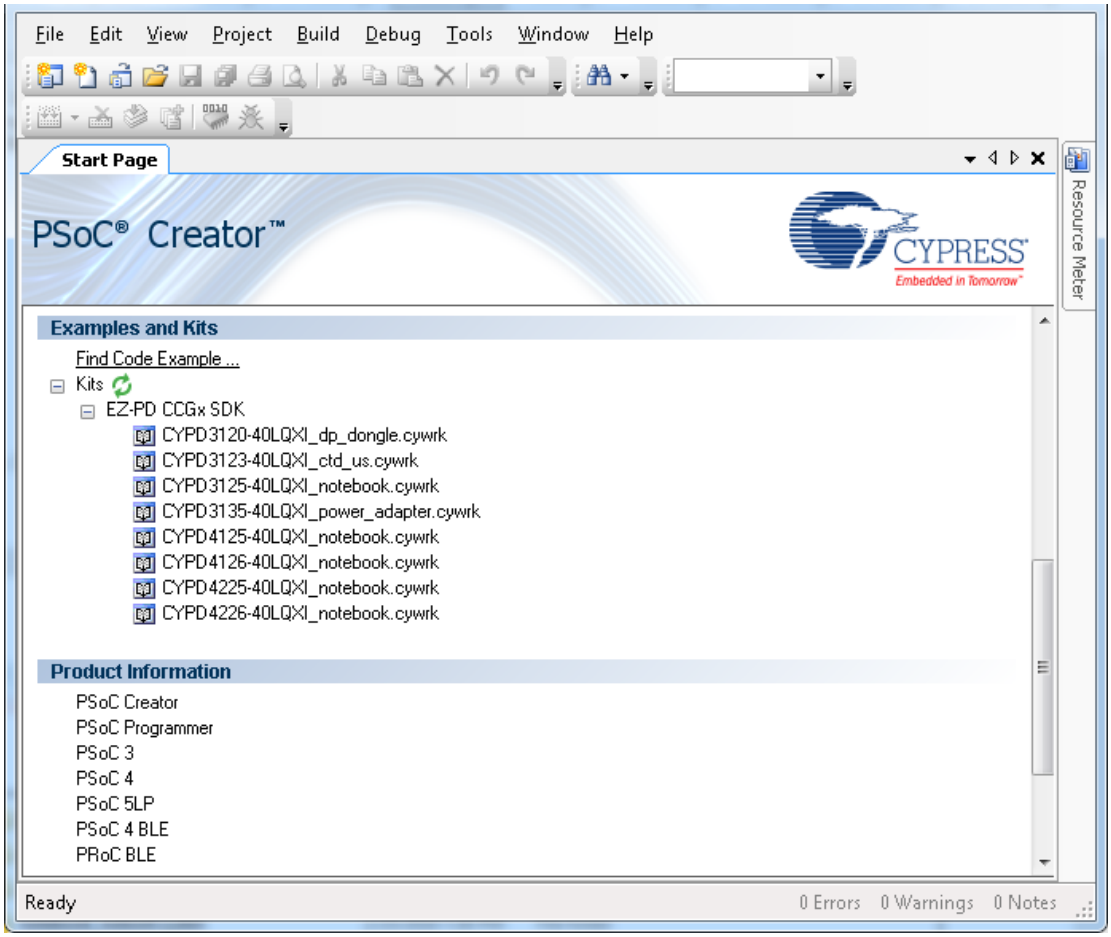
3.1.1 Copying the Project with PSoC Creator

PSoC Creator allows SDK example projects to be copied to a different location without affecting the original installed files. There are mainly two ways of doing this: using the Start Page to copy the workspaces and using the Code Examples.

From Start Page

The SDK example projects are listed under ***Kits->EZ-PD CCGx SDK*** on the Start Page. Click on the workspace name to copy it. When copying the workspace, the complete workspace directory along with all the projects associated with the workspace are copied to the selected destination location. PSoC Creator automatically opens the copied workspace after completing the copy. Figure 3 shows the startup page for Creator.

Figure 3: PSoC Creator Startup Page



Note: If Start Page is not open, it can be accessed via ***View->Other Windows->Start Page***.

From Code Examples

PSoC Creator lists the SDK examples under the Code Examples Dialog. The dialog can be accessed via ***File->Code Example ...*** or during new Project creation.

Figure 4 shows the example projects from Code Examples Dialog. The required Device Family can be selected to narrow down the list of examples and the required project can be copied by clicking the ***Create Project*** button. When using Code Examples Dialog to copy the project, the complete project directory and the selected project shall be copied. It should be noted that the workspace and the files outside the project directory are not copied. All files required for the SDK examples projects are under the project directory and so shall not have any impact.

Note: The ***noboot*** project shall not be present in the newly created workspace and shall require to be explicitly added to the workspace. But, since the ***noboot*** project is under the example project directory, it is also copied when the example project is created.

Create Project option also allows to create a new project based on existing example projects. Figure 5 shows how to create a new Project based on example projects.

Note: When copying via Create Project option, a wrong device part number may get selected. In this case, the user is expected to change the part number to correct one using ***Device Selector Dialog***.

Figure 4: PSoC Creator Code Examples Dialog

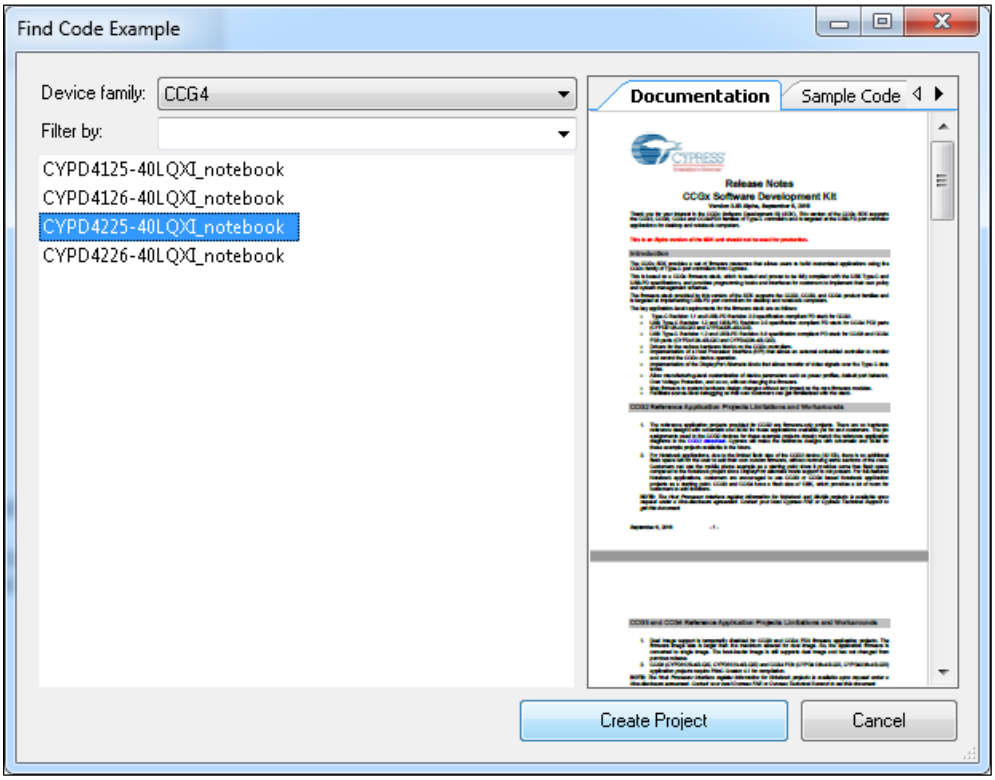
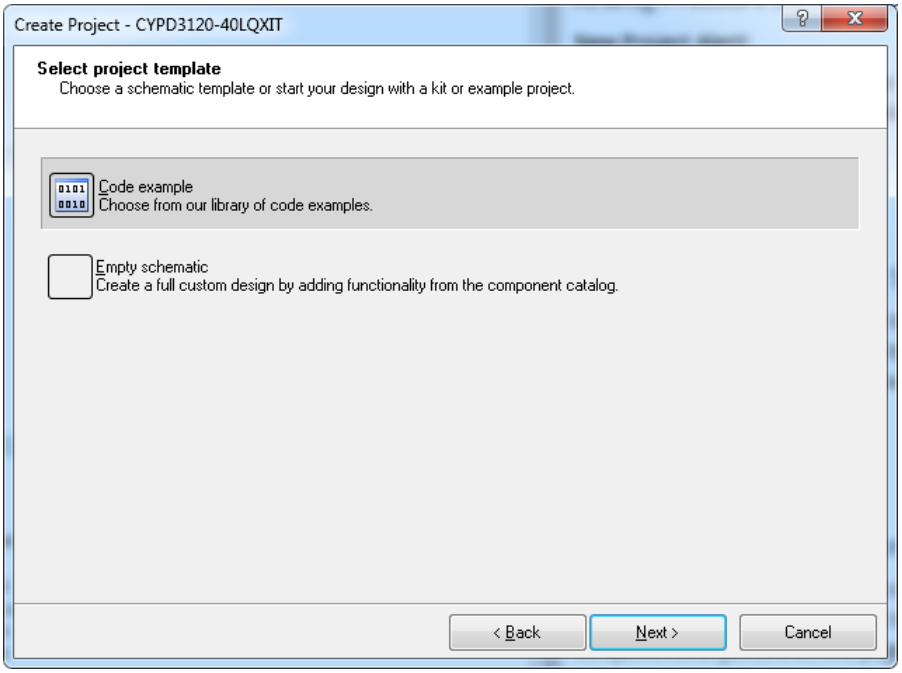


Figure 5: PSoC Creator Create Project Dialog

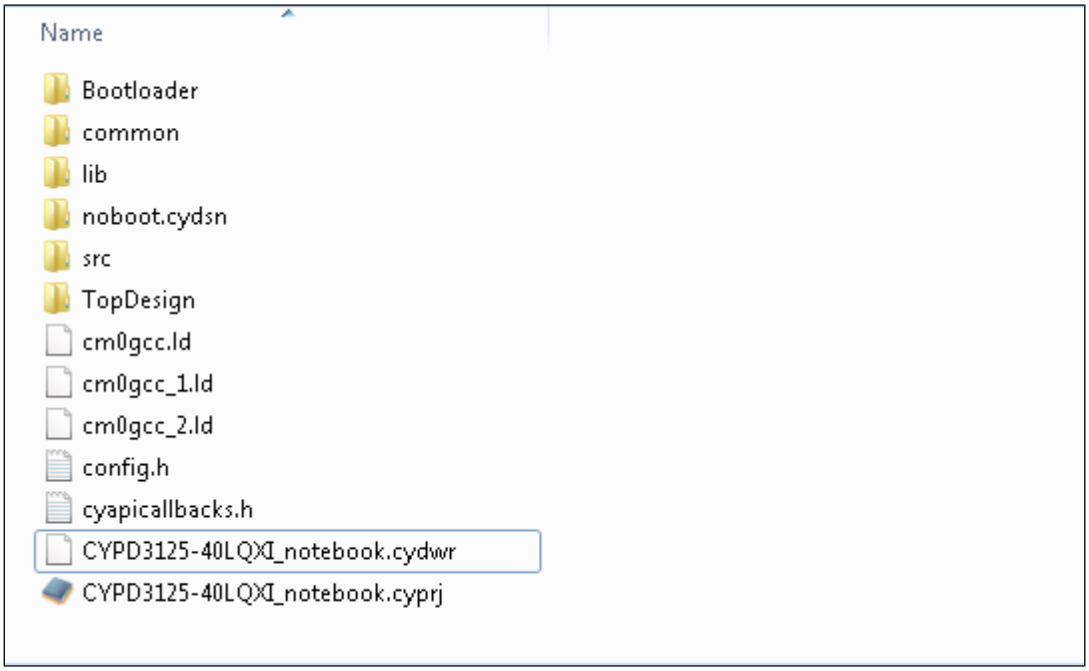


3.1.2 Compiling the Project with PSoC Creator

This section walks you through the procedure to open the reference projects and build them using PSoC Creator. The CYPD4225-40LQXI project is used as an illustration in the following descriptions.

1. Navigate to the project folder using Windows Explorer. The project folder contents will look as shown in Figure 6.
2. The CYPD4225-40LQXI_*notebook.cywrk* file is the PSoC Creator workspace file that can be opened using the PSoC Creator IDE. If you have installed multiple Creator versions, ensure that the appropriate PSoC Creator version is used to open the workspace.

Figure 6: Contents of the Reference Project Folder



3. Once you open the workspace, note that there are two projects:
   a. *CYPD4225-40LQXI_notebook.cydsn*: This is the main firmware project for the application. This application is designed to work on top of the bootloader pre-programmed on the CCG4 device. It also creates two copies of the firmware binary that will be stored in different banks (regions) of the CCG4 device flash so that the system can implement a fail-safe firmware upgrade mechanism. More details on this project are provided in later sections.
   b. *noboot.cydsn*: The main firmware project in CYPD4225-40LQXI_*notebook.cydsn* does not support runtime debugging through the SWD interface. The *noboot.cydsn* is a debug-enabled version of the same firmware application, which does not depend on the bootloader. As run-time debugging is only of interest to customers who have access to the SWD interface on the CCG4 device, this firmware overwrites the complete device flash and expects that the device will be programmed through SWD.

NOTE*:** If the project was copied using Code Examples, then only the main project shall be seen on the workspace. The **noboot.cydsn** project can be added to the workspace using ***Add Existing Project option.

4. The *CYPD4225-40LQXI_notebook.cydsn* project is set as the default project for the workspace. Choose the **Build notebook** menu option from the **Build** menu or the pop-up menu obtained by right-clicking on the project name.
5. Ensure that the compiler Toolchain is set to **ARM GCC 4.9-2015-q1-update**. This can be verified / modified in the Build settings for the project. The Build Settings Dialog can be opened by right clicking the corresponding project. Figure 7 shows the Build Settings Dialog.

Figure 7: Project Build Settings Dialog

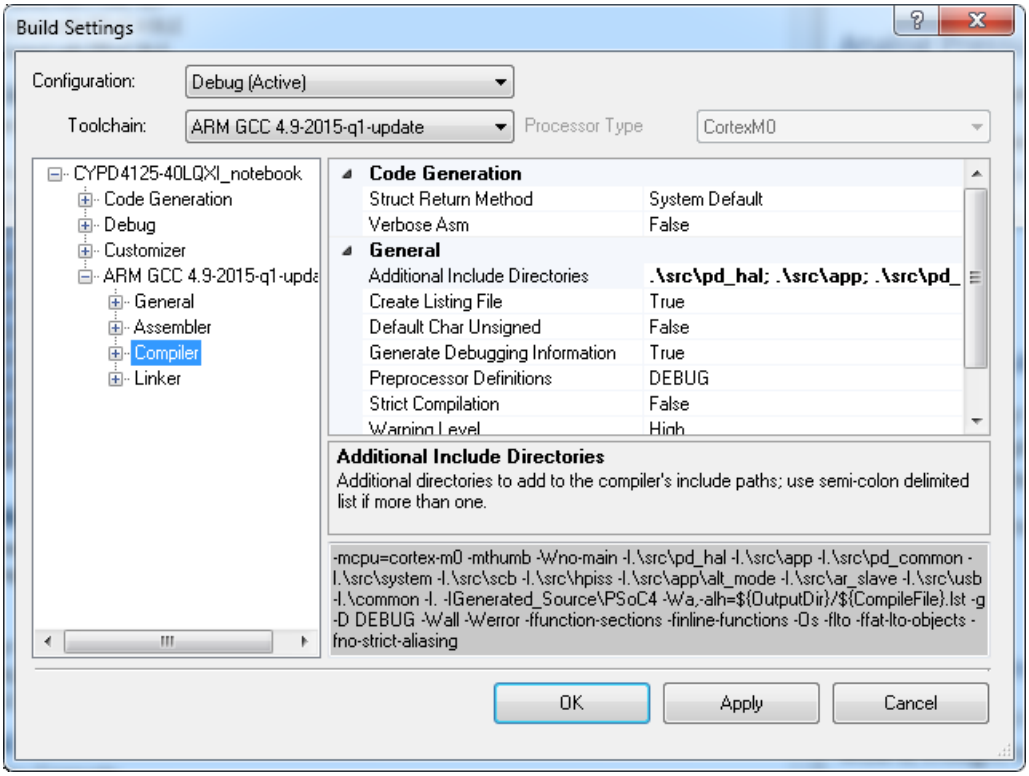
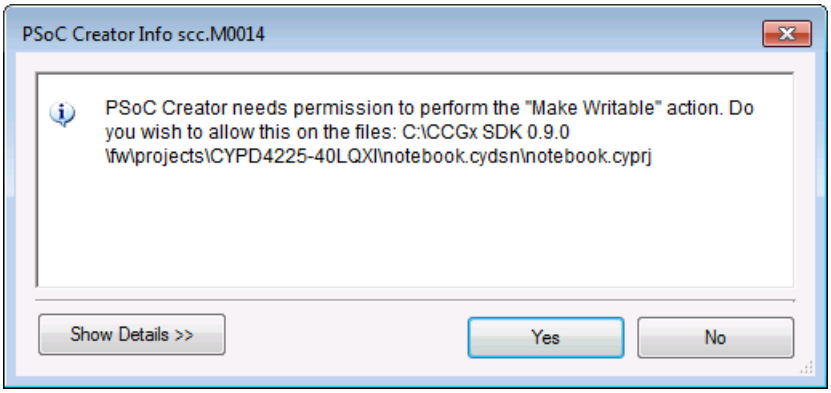
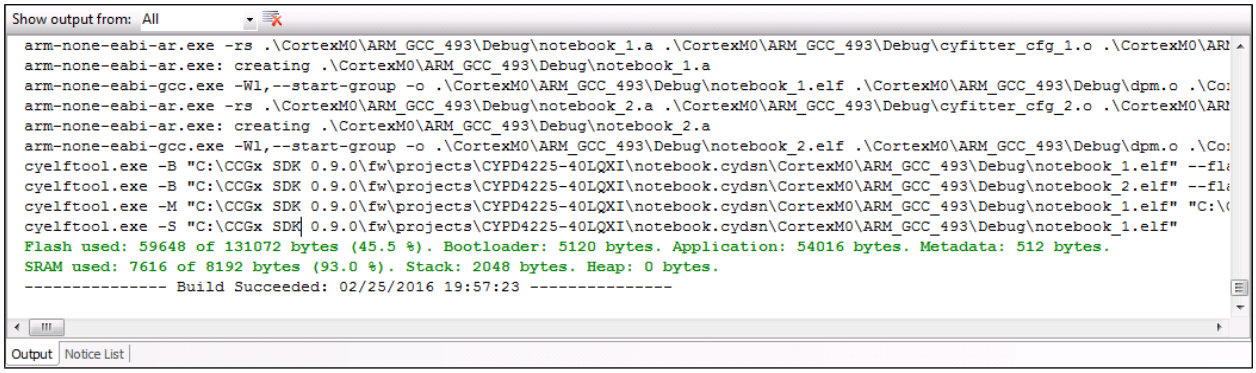


Figure 8: Pop-up Window for Project Write Permissions



6. You may receive a pop-up window asking for permission to make the project file writeable, as shown in Figure 8. Select **Yes** to allow the project build to go through. The complete build process may take about two to three minutes. The output window at the bottom of the IDE will look as shown in Figure 9 at the end of the build process.

Figure 9: Output Window after the Build is Complete



7. Now, navigate to the project folder using Windows Explorer to locate the compiled firmware binaries. Navigate to the CYPD4225-40LQXI_`notebook.cydsn\CortexM0\ARM_GCC_493\Debug` folder for the output files. The following three files are the most important output files generated by the build process:

   a. *CYPD4225-40LQXI_notebook.hex*: This is an SWD programmable binary file in the Intel Hex format that combines the bootloader as well as both copies of the notebook port controller firmware application.

   b. *CYPD4225-40LQXI_notebook_1.cyacd*: This binary file contains the notebook firmware application to be placed in the lower memory bank (region) of the CCG4 device. The format of the file is documented here. The EZ-PD Configuration Utility accepts firmware binaries in the cyacd format and programs them to the CCG4 device using the I^2C slave interface provided by the bootloader (or firmware itself).

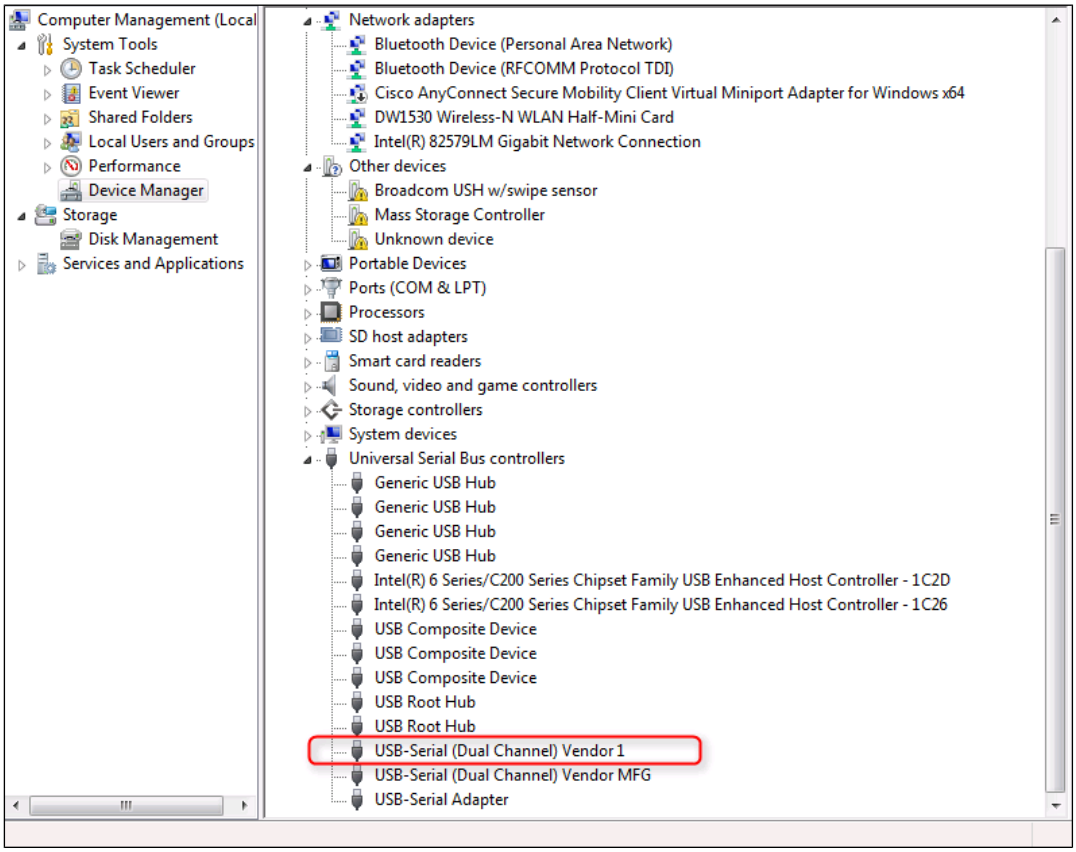
   c. *CYPD4225-40LQXI_notebook_2.cyacd*: This binary file contains the notebook firmware application to be placed in the upper memory bank of the CCG4 device.

Programming CCGx using the EZ-PD Configuration Utility

This section provides step-by-step instructions for updating the firmware with the EZ-PD Configuration Utility, using the CY4541 kit for illustration. However, this procedure remains the similar for other supported CCGx device families. CCG3 Power adapter requires CC line programming and needs to be connected to the Type-C port via CY4541 kit, CY4531 kit. CCG3 DP Dongle can be connected directly to the PC via Type-C port or via CY4541 kit or CY4531 kit. Refer to the Ez-PD Configuration Utility User Manual for detailed instructions for all applications.

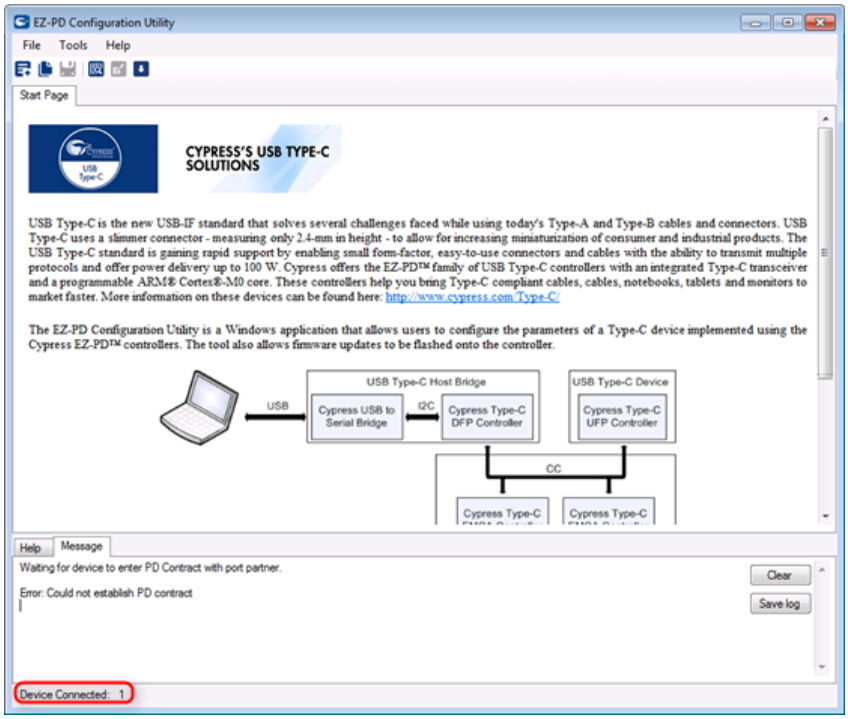
1. Power up the CY4541 kit (the base board of the kit needs to be externally powered) and connect the CCG4 daughter board to the host computer using the USB Mini-B to A cable provided with the kit.
2. Wait for driver detection and binding for the USB-Serial controller on the CY4541 kit. The driver for this controller can be obtained by searching on Windows Update. Once the driver binding is successful, a “USB-Serial (Dual Channel) Vendor 1” device will be listed under ‘Universal Serial Bus Controllers’ in the **Device Manager** window. See Figure 10 for the expected device listing.

Figure 10: Device Manager View Showing USB-Serial Bridge Device



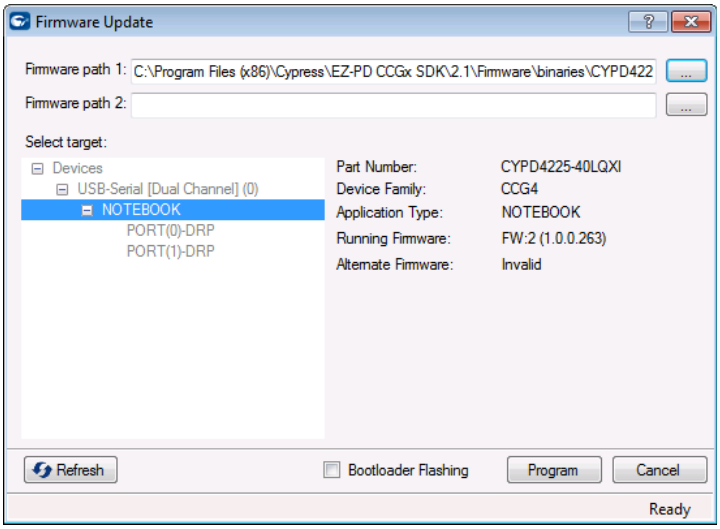
3. If the automatic driver installation does not succeed, you can download and use the Cypress USB Serial Windows Driver Installer. Refer to the USB-Serial Windows Driver Installation Guide document too.
4. Open the EZ-PD Configuration Utility GUI. If the device driver binding is successful, the GUI should report one device connected on the lower border of the UI as shown in Figure 11.

Figure 11: Configuration Utility Detecting the Connected CCG4 EVK



5. Go to **Tools** > **Firmware Update**. The utility detects and identifies the device at this stage. A firmware update dialog appears at the end of this process (see Figure 12).
6. When you click on the **Notebook** node in the device tree, the UI displays information about the CCGx device and the current firmware running on it.

Figure 12: Firmware Update Dialog

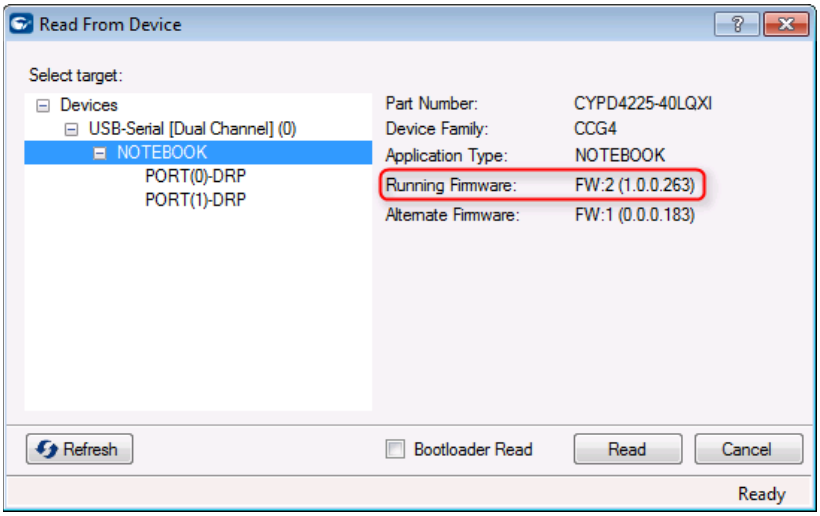


7. Navigate to the folder containing the firmware binaries generated during the firmware build, and select the CYPD4225-40LQXI_*notebook_1.cyacd* and CYPD4225-40LQXI_*notebook_2.cyacd* files in the two firmware path options in the dialog.

   ***Note**:* The firmware is designed such that notebook_1 image will allow notebook_2 to be updated and vice versa. Selecting both files will allow the utility to update using the appropriate file based on the currently running firmware.

8. On the CCGx device, it is possible to update the firmware while the device is in PD contract. Hence, the utility does not automatically reset the device at the end of the firmware update sequence. You must do a manual reset or power cycle to activate the newly programmed firmware.
9. After the device is reset, use either the **Tools** > **Read from Device** or **Tools** > **Firmware Update** option to bring up a dialog that can show the current firmware version. If the firmware project from the SDK is used without any changes, the new running firmware version should match the version of firmware downloaded, as shown in Figure 13. The running firmware can be FW1 or FW2 depending on the firmware version, which was running prior to the update operation.
10. The CCGx bootloader is designed such that it loads the last programmed firmware binary (irrespective of the firmware version). If desired, you can repeat steps 4 through 8 to update the firmware in the second bank too.

Figure 13: Firmware Versions after Update Operation



3.2 Updating CCGx Configuration

The CCGx firmware design uses a configuration table, which specifies several parameters that control device functionality. These parameters include:

- The VDM responses sent by the device for DISCOVER_ID, DISCOVER_SVID, and DISCOVER_MODE requests when it is functioning as a UFP.
- The power profiles supported by the device as a provider and as a consumer.
- The various alternate mode configurations currently supported by CCGx as a DFP (only DisplayPort alternate mode is supported currently).
- The port roles supported by the device (Source/Sink/Dual Role).
- Enable/disable flags and parameters that control the overvoltage and overcurrent protection schemes implemented by CCGx.

These parameters are stored in a configuration table so that they can be updated/customized without updating the firmware.

Each copy of the CCGx notebook firmware (notebook_1 and notebook_2) contains its own embedded configuration table. For safety reasons, the firmware does not update the current configuration. You can update the configuration for the alternate image and then switch control to it, to run with the new configuration.

Configuration using the EZ-PD Configuration Utility

The utility provides an interactive GUI through which all of the contents of the configuration table can be updated.

Figure 14: PD Port Configuration using EZ-PD Configuration Utility

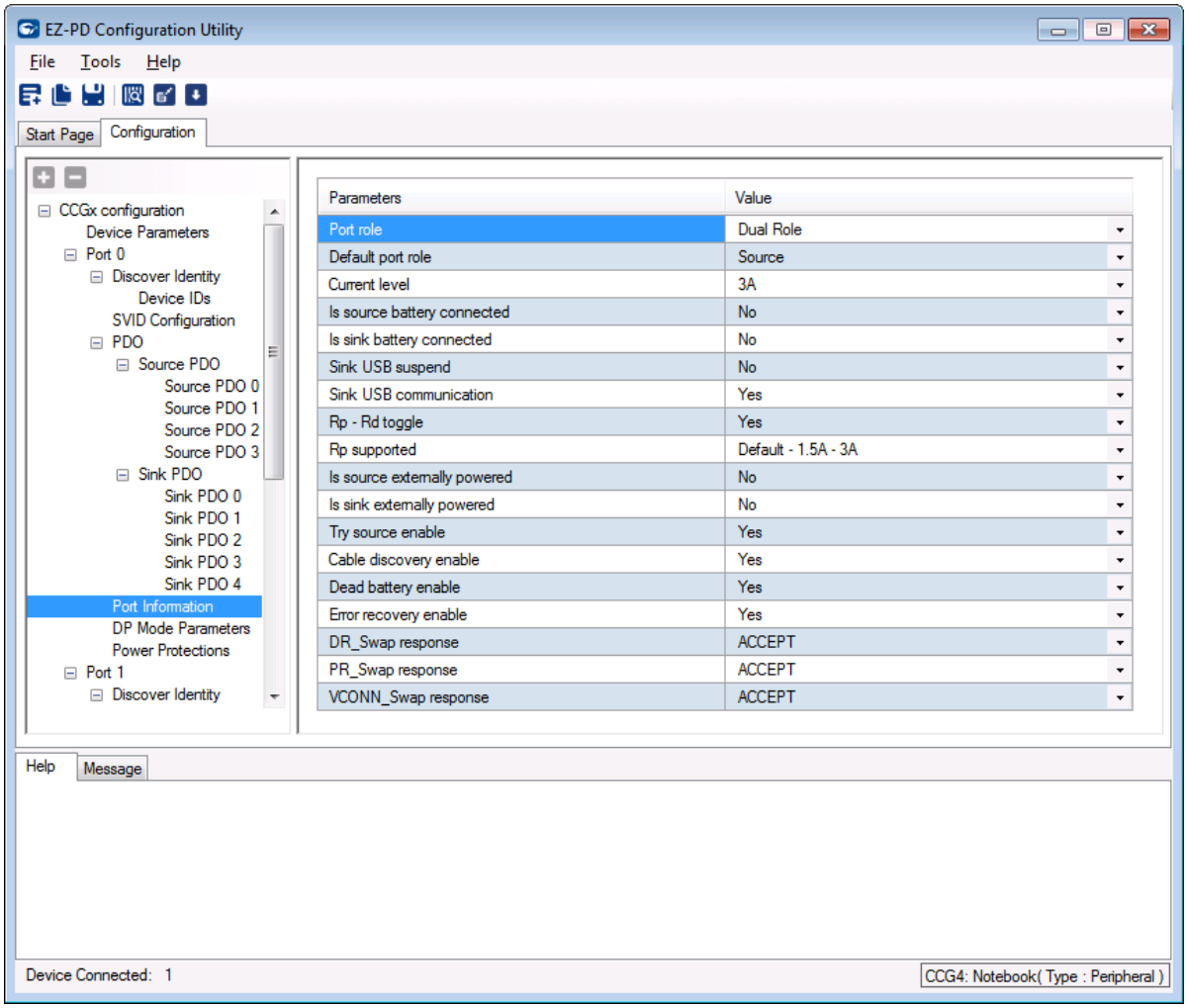


Figure 14 shows a snapshot of the UI screens used for configuring the CCGx notebook solution as an example. The entire device configuration is completed by navigating through all of the nodes shown on the left side window of the UI.

Refer to the Configuration Utility User Manual for a description of the various configuration screens provided by the utility. Each UI screen also provides tool-tips that guide you through the process of defining the configuration.

Table 1: List of CCG3/CCG4 Notebook Configuration Parameters

Configuration Parameter	Default Value	Change Allowed
Device Parameters		
Manufacturer Info	“Cypress”	Can be changed. Valid only for PD 3.0 configurations.
Device IDs		
USB host support	Yes	Not allowed
USB device support	No	Not allowed
Modal operation supported	No	This gets changed when an SVID is added
USB Vendor Id	0x04B4	Can be changed
Product type	Peripheral	Not allowed
USB-ID compliance test ID	0	Can be changed
USB Product ID	F640 for CCG3 F6C0 for CCG4	Can be changed
Bcd device	0	Can be changed
SVID Configuration		
SVID	None	Can be changed
Mode	None	Can be changed
Source PDOs		
Source PDO 0	5 V @ 3A	Not recommended
Source PDO 1	9 V @ 3A	Can be changed based on hardware capabilities
Source PDO 2	15 V @ 3A	Can be changed based on hardware capabilities
Source PDO 3	20 V @ 3A	Can be changed based on hardware capabilities
Sink PDOs		
Sink PDO 0	5 V @ 0.9 A	Current can be changed
Sink PDO 1	Variable PDO 7 V to 21 V @0.9 A	Can be changed based on hardware capabilities
Port Information		
Port role	Dual role	Can be changed
Default port role	Source	Can be changed
Current level	3A	Can be changed
Is source battery connected	No	Can be changed based on hardware capabilities
Is sink battery connected	No	Can be changed based on hardware capabilities
Sink USB suspend	No	Not recommended

Configuration Parameter	Default Value	Change Allowed
Sink USB communication	Yes	Can be changed
Rp-Rd Toggle	Yes	Not recommended if port role is “Dual role”
Rp supported	Default, 1.5A and 3.0A	Can be changed
Is source externally powered	No	Can be changed
Is sink externally powered	No	Can be changed
DRP preferred role	Source	Can be changed
Cable discovery enable	Yes	Can be changed
Dead battery enable	Yes	Can be changed
Error recovery enable	Yes	Not recommended
DR_SWAP response	ACCEPT	Can be changed
PR_SWAP response	ACCEPT	Can be changed
VCONN_SWAP response	ACCEPT	Can be changed
FRS Enable	FRS receive	Can be changed. Valid only for PD 3.0 configurations.
DP Mode Parameters		
Modes supported	CDEF	Can be changed
Power Protections		
Over Voltage Protection	Enable	Not recommended
Over Current Protection	Enable	Not recommended
OVP Threshold	20%	Can be changed in the range of 10 % to 50 %
OVP debounce period	1 us	Not allowed
OCP Threshold	CCG4: 0 CCG3: 20%	Not allowed for CCG4. Can be changed in the range of 10 % to 50 % for CCG3.
OCP debounce period	CCG4: 0 ms CCG3: 1 ms	Not allowed for CCG4. Can be changed for CCG3.
OCP off time	0	Not allowed
OCP retry count	0	Not allowed
OCP sample period	0	Not allowed
User Parameters		
Parameters 1 to 8	00	Can be changed to any value

Table 1 shows the list of configuration parameters supported by the CCGx notebook firmware, along with the default values and restrictions on changing them. Note that changing the Source PDO configuration is not recommended while using the EVKs, because the default settings correspond to the actual kit hardware configuration.

Table 2: List of CCG3 Power Adapter Configuration Parameters

Configuration Parameter	Default Value	Change Allowed
Device Parameters		
Manufacturer Info	“Cypress”	Can be changed. Valid only for PD 3.0 configurations.
Device IDs		
USB host support	No	Not allowed
USB device support	No	Not allowed
Modal operation supported	Yes	This gets changed when an SVID is added
USB Vendor Id	0x04B4	Can be changed
Product type	Undefined	Not allowed
USB-ID compliance test ID	0	Can be changed
USB Product ID	F640	Can be changed
Bcd device	0	Can be changed
SVID Configuration		
SVID	0x04B4	Not allowed
Mode	0x00000001	Not recommended
Source PDOs		
Source PDO 0	5 V @ 3A	Not recommended
Source PDO 1	9 V @ 3A	Can be changed based on hardware capabilities
Source PDO 2	15 V @ 3A	Can be changed based on hardware capabilities
Source PDO 3	20 V @ 3A	Can be changed based on hardware capabilities
Port Information		
Port role	Source	Not allowed
Default port role	Source	Not allowed
Current level	3A	Can be changed
Is source battery connected	No	Not allowed
Is sink battery connected	No	Not allowed
Sink USB suspend	No	Not recommended
Sink USB communication	No	Not recommended
Rp-Rd Toggle	No	Not allowed
Rp supported	Default, 1.5A and 3.0A	Can be changed
Is source externally powered	Yes	Not allowed
Is sink externally powered	No	Not allowed
DRP preferred role	Source	Not allowed
Cable discovery enable	Yes	Can be changed
Dead battery enable	No	Not allowed
Error recovery enable	Yes	Not recommended

Configuration Parameter	Default Value	Change Allowed
DR_SWAP response	ACCEPT	Not recommended
PR_SWAP response	REJECT	Can be changed
VCONN_SWAP response	REJECT	Can be changed
FRS enable	None	Not recommended
Power Protections		
Over Voltage Protection	Enabled	Not recommended
Over Current Protection	Polling Method	Not recommended
OVP Threshold	20%	Can be changed in the range of 10 % to 50 %
OVP debounce period	1 us	Not allowed
OCP Threshold	20%	Can be changed in range of 10% to 50 %
OCP debounce period	1 ms	Can be changed
OCP off time	100 ms	Can be changed
OCP retry count	3	Can be changed
OCP sample period	0	Not allowed
User Parameters		
Parameters 1 to 8	00	Can be changed to any value

Table 2 shows the list of configuration parameters supported by the CCG3 power adapter firmware, along with the default values and restrictions on changing them.

Table 3: List of CCG3 DP Dongle Configuration Parameters

Configuration Parameter	Default Value	Change Allowed
Device Parameters		
Manufacturer Info	“Cypress”	Can be changed. Valid only for PD 3.0 configurations.
Device IDs		
USB host support	No	Not allowed
USB device support	Yes	Not allowed
Modal operation supported	Yes	This gets changed when an SVID is added
USB Vendor Id	0x04B4	Can be changed
Product type	AMA	Not allowed
USB-ID compliance test ID	0	Can be changed
USB Product ID	3120	Can be changed
Bcd device	0	Can be changed
AMA VDO		
Hardware version	0	Can be changed
Firmware version	0	Can be changed
SSTX1 directionality support	Fixed	Can be changed
SSTX2 directionality support	Fixed	Can be changed

Configuration Parameter	Default Value	Change Allowed
SSRX1 directionality support	Fixed	Can be changed
SSRX2 directionality support	Fixed	Can be changed
VConn power	1W	Can be changed
VConn required	Yes	Can be changed
VBUS required	Yes	Can be changed
USB version	USB 2.0 Billboard Only	Can be changed
SVID Configuration		
SVID	FF01	Not recommended
Mode	00001405	Not recommended
Sink PDOs		
Sink PDO 0	5 V @ 0.9 A	Current can be changed
Port Information		
Port role	Sink	Not recommended
Default port role	Sink	Not recommended
Current level	Default	Not recommended
Is source battery connected	No	Not recommended
Is sink battery connected	No	Can be changed based on hardware capabilities
Sink USB suspend	No	Not recommended
Sink USB communication	Yes	Can be changed
Rp-Rd Toggle	No	Not recommended
Rp supported	None	Not recommended
Is source externally powered	No	Not recommended
Is sink externally powered	No	Not recommended
DRP preferred role	None	Not recommended
Cable discovery enable	No	Not allowed
Dead battery enable	Yes	Not recommended
Error recovery enable	Yes	Not recommended
DR_SWAP response	REJECT	Not recommended
PR_SWAP response	REJECT	Not recommended
VCONN_SWAP response	REJECT	Can be changed
FRS enable	None	Not recommended. Valid only for PS 3.0 configurations.
Billboard Parameters		
Billboard type	Internal BB device	Not recommended
Billboard enable	BB always enabled	Can be changed
Billboard programmer support	Programming interface	Not recommended

Configuration Parameter	Default Value	Change Allowed
	enabled	
Billboard timeout	600 s	Can be changed
Billboard power settings	Bus powered	Can be changed
Billboard container ID settings	Generate based on device UID	Can be changed
Billboard VConn power setting	1W	Can be changed
Billboard serial number setting	Generate based on device UID	Can be changed
Billboard Settings		
VID	04B4	Changes when the Device ID field is updated
PID	3120	Changes when the Device ID field is updated
Manufacturer	Cypress Semiconductor	Can be changed
Product	Type-C DP Dongle	Can be changed
Configuration	Billboard Configuration	Can be changed
Billboard interface	Billboard Interface	Can be changed
HID interface	Control Interface	Can be changed
Additional info URL	http://www.cypress.com/Type-C/	Can be changed
Preferred mode	0	Can be changed
Alternate Mode 0	“Type-C Alternate Mode”	Can be changed
DP Mode Parameters		
Modes supported	C	Can be changed
Preferred DP Mode	4-lane DisplayPort	Can be changed
Power Protections		
Over Voltage Protection	Enabled	Not recommended
Over Current Protection	Disable	Not recommended
OVP Threshold	20%	Can be changed in the range of 10 % to 50 %
OVP debounce period	0	Not allowed
OCP Threshold	0	Not recommended
OCP debounce period	0	Not recommended
OCP off time	0	Not allowed
OCP retry count	0	Not allowed
OCP sample period	0	Not allowed
User Parameters		
Parameters 1 to 8	00	Can be changed to any value

Table 3 shows the list of configuration parameters supported by the CCG3 DP Dongle firmware, along with the default values and restrictions on changing them.

Table 4: List of CCG3 Charge Through Dongle Upstream Configuration Parameters

Configuration Parameter	Default Value	Change Allowed
Device Parameters		
Manufacturer Info	“Cypress”	Can be changed. Valid only for PD 3.0 configurations.
Device IDs		
USB host support	No	Not allowed
USB device support	Yes	Not allowed
Modal operation supported	Yes	This gets changed when an SVID is added
USB Vendor Id	0x04B4	Can be changed
Product type	AMA	Not allowed
USB-ID compliance test ID	0	Can be changed
USB Product ID	0xF649	Can be changed
Bcd device	0	Can be changed
AMA VDO		
Hardware version	0	Can be changed
Firmware version	0	Can be changed
SSTX1 directionality support	Fixed	Can be changed
SSTX2 directionality support	Fixed	Can be changed
SSRX1 directionality support	Fixed	Can be changed
SSRX2 directionality support	Fixed	Can be changed
VConn power	1W	Can be changed
VConn required	Yes	Can be changed
VBUS required	Yes	Can be changed
USB version	[USB 3.1] Gen 1 and USB 2.0	Can be changed
SVID Configuration		
SVID	FF01	Not recommended
Mode	00001405	Not recommended
Source PDOs		
Source PDO 0	5 V @ 0.9 A	Not recommended
Sink PDOs		
Sink PDO 0	5 V @ 0.9 A	Not recommended
Port Information		
Port role	Dual Role	Not recommended
Default port role	Sink	Not recommended
Current level	Default	Not recommended

Configuration Parameter	Default Value	Change Allowed
Is source battery connected	No	Can be changed
Is sink battery connected	No	Can be changed based on hardware capabilities
Sink USB suspend	No	Not recommended
Sink USB communication	Yes	Can be changed
Rp-Rd Toggle	Yes	Can be changed
Rp supported	Default, 1.5A and 3.0A	Can be changed
Is source externally powered	Yes	Can be changed
Is sink externally powered	Yes	Can be changed
DRP preferred role	None	Can be changed
Cable discovery enable	No	Not allowed
Dead battery enable	Yes	Not recommended
Error recovery enable	Yes	Not recommended
DR_SWAP response	REJECT	Can be changed
PR_SWAP response	REJECT	Can be changed
VCONN_SWAP response	REJECT	Can be changed
FRS enable	FRS transmit	Can be changed. Valid only for PD 3.0 configurations.
Billboard Parameters		
Billboard type	Internal BB device	Not recommended
Billboard enable	BB always enabled	Can be changed
Billboard programmer support	Programming interface enabled	Not recommended
Billboard timeout	600 s	Can be changed
Billboard power settings	Bus powered	Can be changed
Billboard container ID settings	Generate based on device UID	Can be changed
Billboard VConn power setting	1W	Can be changed
Billboard serial number setting	Generate based on device UID	Can be changed
Billboard Settings		
VID	04B4	Changes when the Device ID field is updated
PID	F649	Changes when the Device ID field is updated
Manufacturer	Cypress Semiconductor	Can be changed
Product	Type-C DP Dongle	Can be changed
Configuration	Billboard Configuration	Can be changed

Configuration Parameter	Default Value	Change Allowed
Billboard interface	Billboard Interface	Can be changed
HID interface	Control Interface	Can be changed
Additional info URL	http://www.cypress.com/Type-C/	Can be changed
Preferred mode	0	Can be changed
Alternate Mode 0	“Type-C Alternate Mode”	Can be changed
DP Mode Parameters		
Modes supported	D	Can be changed
Preferred DP Mode	Multi-function DisplayPort	Can be changed
Power Protections		
Over Voltage Protection	Enabled	Not recommended
Over Current Protection	Disable	Can be changed
OVP Threshold	20%	Can be changed in the range of 10 % to 50 %
OVP debounce period	0	Not allowed
OCP Threshold	0	Can be changed
OCP debounce period	0	Can be changed
OCP off time	0	Not allowed
OCP retry count	0	Not allowed
OCP sample period	0	Not allowed
User Parameters		
Parameters 1 to 8	00	Can be changed to any value

Table 4 shows the list of configuration parameters supported by the CCG3 Charge Through Dongle upstream firmware, along with the default values and restrictions on changing them.

The various fields under Port Information are inter-related, and should be updated to be mutually consistent. Table 5 shows the allowed variations and restrictions on various parameter values across different kinds of CCG applications.

Table 5: Port Information Variability across Applications

Parameter	DRP Application	DFP Application	UFP Application
Port role	Dual role	Source	Sink
Default port role	Variable	Source	Sink
Current level	Variable	Variable	Variable
Is source battery connected	Variable	Variable	No
Is sink battery connected	Variable	No	Variable
Sink USB suspend	Variable	No	Variable
Sink USB communication	Variable	No	Variable
Rp-Rd Toggle	Yes	No	No
Rp supported	Variable	Variable	None

Parameter	DRP Application	DFP Application	UFP Application
Is source externally powered	Variable	Variable	No
Is sink externally powered	Variable	No	Variable
Try source enable	Variable	No	No
Cable discovery enable	Variable	Variable	No
Dead battery enable	Variable	Variable	Variable
Error recovery enable	Variable	Variable	Variable
DR_SWAP response	Variable	Variable	Variable
PR_SWAP response	Variable	REJECT	REJECT
VCONN_SWAP response	Variable	Variable	Variable

Table 6 describes the extended source capabilities message parameters. Extended source capabilities message is supported by all port controllers with Source configuration either in Dual role or Source only mode. This message is used only when both port partners are in PD 3.0 contract.

Table 6: Extended Source Capabilities Parameters

Configuration Parameter	Default Value	Change Allowed
SCEDB Configuration		
XID	0	Can be changed
FW Version	0	Can be changed
HW Version	0	Can be changed
Voltage regulation load step slew rate (mA/us)	150	Not allowed
Voltage regulation load step magnitude (%IoC)	25	Can be changed
Holdup Time (ms)	3	Can be changed
LPS compliant	No	Can be changed
PS1 compliant	No	Can be changed
PS2 compliant	No	Can be changed
Low touch current EPS	No	Can be changed
Ground pin	Not supported	Can be changed
Touch temp	Default	Can be changed
Source Inputs-External supplies	No external supply	Can be changed
Source Inputs-Internal batteries	No	Can be changed
Hot swappable batteries	0	Can be changed
Fixed batteries	0	Can be changed
Peak Current		
Percentage overload (%)	0	Can be changed
Overload period (ms)	0	Can be changed

Configuration Parameter	Default Value	Change Allowed
Duty cycle (%)	0	Can be changed
Vbus voltage droop	No	Can be changed

After all the parameters are defined, click on the ‘Save’ button (or go to **File** > **Save As**) to save a copy of the configuration to the disk. The configuration is stored in the form of an XML file. The utility also generates two additional output files that help the user in deploying the configuration.

1. A *cyacd* file is generated, which can be used to program the new configuration data to the device. The EZ-PD utility itself uses the *cyacd* file for device programming.
2. A *.c* file is generated, which can be included in the firmware project to compile a new binary that embeds the desired configuration. More details on the use of this file are provided in later sections of this guide.

After the configuration is saved, use the **Tools** > **Configure Device** option to program the configuration to the device. As mentioned above, the configuration update applies to the current alternate firmware image. The utility issues a warning if the alternate firmware version is older than the current version, and you have the option of aborting the configuration update at this stage.

Resetting the device after a configuration update will cause the device to start running the alternate firmware, which was programmed with a new configuration.

4. Customizing the Firmware Application

As shown in section 3.2, a major part of the CCGx application functionality can be modified without having to change any of the firmware sources.

Any changes to the hardware design around the CCGx device will, however, require changes to the firmware sources implementing the application. This chapter walks through the process of updating the firmware implementation to work with a different hardware design.

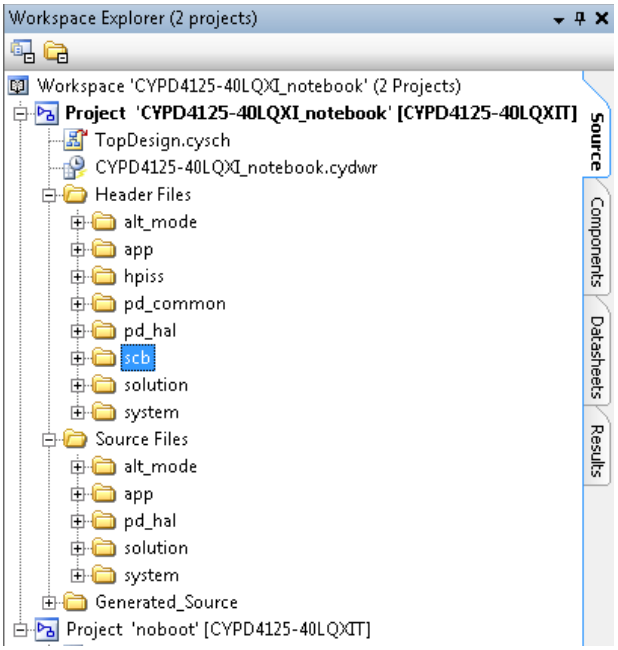
Note: As the firmware sources and reference projects are installed in the **Program Files** folder, it is not recommended that you make changes to the original installed version of these files. You can create a copy of the Firmware folder from the SDK installation, and use the copy for making any changes. This will ensure that you have a clean version of the files that you can revert to as well. Refer to section 3.1.1for more details.

Since the target application remains the same, it is expected that the changes are limited to aspects such as the mechanism for voltage selection, FET control, data path MUX/Switch control, and so on. This does not involve changes to the core functionality implemented by the CCGx device.

4.1 Solution Structure

The CCGx solution structure is shown in Figure 15. The figure uses the CYPD4125-40LQXI_notebook workspace as reference. The source and header files used in the solution are grouped into different folders.

- Figure 15: CCGx Notebook Solution Structure

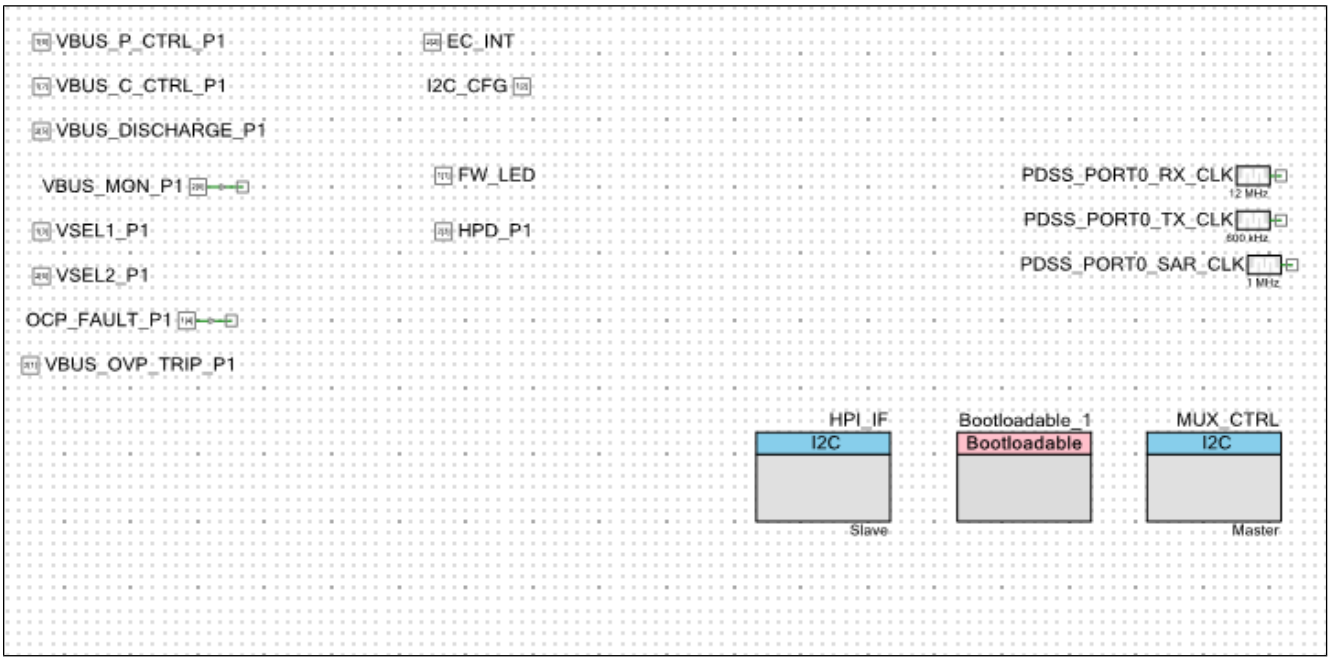


- Solution: The solution folders contain header and source files that provide user configurations, user hardware-specific functions and custom code modules. It is expected that these files will need to be changed to match the hardware design and requirements for all customer implementations. The solution-level sources include:
  - config.h: Header file that enables/disables firmware features and provides macros or function mappings for hardware-specific functions such as FET control and voltage selection.
  - alt_modes_config.h: Header file that selects the alternate modes that are supported by the firmware when CCGx is a Downstream Facing Port (DFP) or Upstream Facing Port (UFP).
  - stack_params.h: This header file defines a number of properties that are used to customize the PD stack operation.
  - config.c: This source file contains the default run-time configuration for the CCGx notebook application and has been generated using the EZ-PD Configuration Utility.
  - datamux_ctrl.c: This source file contains the functions that control the Type-C data switch that connects the Type-C data pins to the USB and DisplayPort controllers in the system.
  - main.c: This source file contains the main application entry point.
- app: The app folders contain header and source files that implement the device policy decisions such as power contract negotiation roles, port role management, power protection schemes, Vendor Defined Message (VDM) handling, and so on. The default implementation provided in the source form uses the configuration table and runtime customizations provided by the EC to handle these tasks. The files can be updated if there is a need to change the way policy decisions are implemented by the CCG firmware. The app source files include:
  - app.c: This is the top-level application source file that connects the PD stack to the alternate modes manager as well as the solution level code.
  - pdo.c: This source file implements the Power Data Object (PDO) and Request Data Object (RDO) handlers that define the power contract negotiation rules.
  - psource.c: This source file implements the power source-related state machines and tasks.
  - psink.c: This source file implements the power sink related state machines and tasks.
  - swap.c: This source file implements the various swap request handlers.
  - vdm.c: This source file implements the handlers for VDMs received by the CCG device.
- pd_hal: The pd_hal folders header and source files that implement the low level drivers for the PD stack. The header file definitions should not be modified as these are used by the PD stack library and conflicting definitions can result in undefined behavior. The pd_hal level sources include:
  - hal_ccgx.c: This source file implements Over Voltage Protection (OVP) and Over Current Protection (OCP) tasks, which are specific to the CCG device architecture.
- alt_mode: This folder contains header and source files that implement the alternate mode manager functions for when CCG is functioning as DFP and when CCG is functioning as UFP.
- hpiss: This folder contains the header files providing the serial communication block (SCB) driver and Host Processor Interface (HPI) protocol interfaces. The actual I^2C driver and HPI code is provided in library form. The header file definitions should not be modified as these are used by the HPI stack library and conflicting definitions can result in undefined behavior.
- pd_common: Since the PD stack is provided in the library form, the pd_common folder only contains header files that provide data structure definitions and function declarations for the PD stack. The header file definitions should not be modified as these are used by the PD stack library and conflicting definitions can result in undefined behavior.
- system: This folder contains the base system-level functionality such as GPIO, soft timer implementation, flash driver, and firmware upgrade handlers. The header file definitions should not be modified as these are used by the PD and HPI stack libraries and conflicting definitions can result in undefined behavior.

4.2 CCG4 Notebook

4.2.1 PSoC Creator Schematic

Figure 16: PSoC Creator Schematic for CCG4 Notebook



Most aspects of the hardware design around the CCG4 device are captured in the schematics associated with the PSoC Creator firmware project.

The PSoC Creator schematic can be found in the *TopDesign.cysch* file, which is part of each PSoC Creator project. Double-click on this file to open the schematic editor window (see Figure 16).

The schematic shows how internal resources of the CCG4 device are used in the design. This includes all the internal clocks used by the design, the various serial interfaces, and all the GPIO pins used to communicate with external elements.

The analog input pins of the CCG4 device are shown with a red wire connected to it on the right side. See the VBUS_MON_P1 signal for example.

Digital input pins are shown with a green wire connected to it on the right side. See the OCP_FAULT_P1 signal for example.

Digital output pins are shown with the corresponding pin mapping annotated on the left side. See the VBUS_P_CTRL_P1 signal for example.

Table 7 shows the various schematic elements used in the CCG4 notebook project. The selection of some of these elements is fixed due to the capabilities of the CCG4 device and the bootloader design. The table also points out the changes allowed in the schematic design.

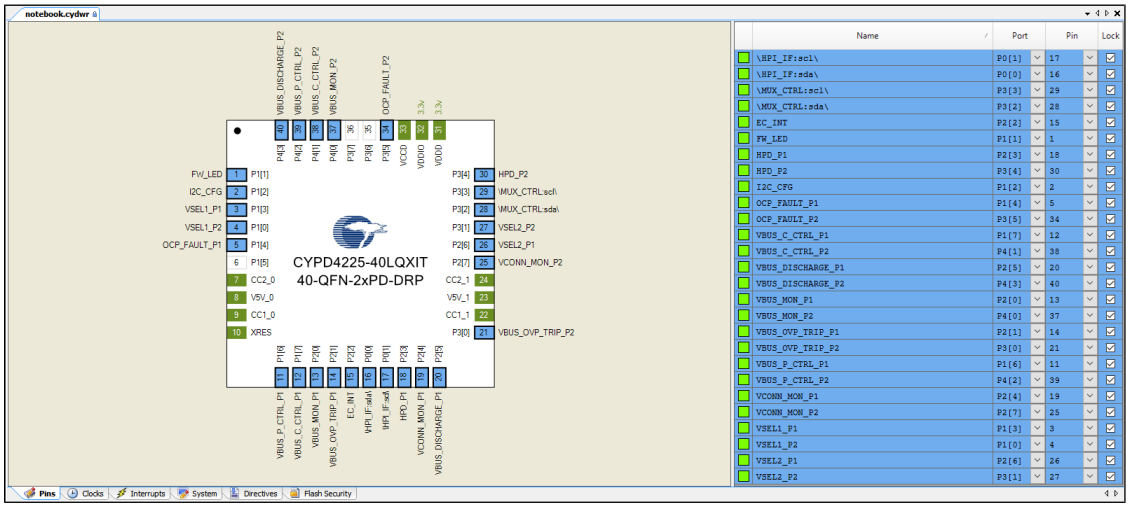
Table 7: Schematic Elements in CCG4 Notebook Design

Schematic Element	Description	Changes Allowed
Bootloadable_1	This is a software block, which interacts with the bootloader on the CCG4 device.	No changes should be made to this element.
HPI_IF	This is an I^2C slave block through which the CCG4 communicates with the Embedded Controller in the Notebook design.	No changes are allowed as the HPI_IF is also used by the bootloader which is fixed.
MUX_CTRL	This is an I^2C master block used by CCG4 to configure the Parade Type-C Interface switch on the CY4541 kit.	This block can be changed / replaced by other mechanisms (such as GPIOs), which can control the interface switch on the target design.
PDSS_PORT0_RX_CLK	This is an internal clock that is used for the RX portion of the USB-PD block.	No changes are allowed.
PDSS_PORT0_TX_CLK	This is an internal clock that is used for the TX portion of the USB-PD block.	No changes are allowed.
PDSS_PORT0_SAR_CLK	This is an internal clock that is used for the analog portion of the USB-PD block.	No changes are allowed.
PDSS_PORT1_RX_CLK	This is an internal clock that is used for the RX portion of the USB-PD block.	No changes are allowed.
PDSS_PORT1_TX_CLK	This is an internal clock that is used for the TX portion of the USB-PD block.	No changes are allowed.
PDSS_PORT1_SAR_CLK	This is an internal clock that is used for the analog portion of the USB-PD block.	No changes are allowed.
EC_INT	This is an output pin used to interrupt the Embedded Controller when there is a state change.	No changes are allowed as EC_INT is also used by boot-loader.
I2C_CFG	This is an input pin used to select the I2C slave address used on the HPI interface.	No changes are allowed as EC_INT is also used by boot-loader.
HPD_P1 HPD_P2	This is the Hotplug Detect output pin from CCG4 to the DisplayPort controller on the notebook.	This pin can be removed if DisplayPort is not used. If used, the pin mapping cannot be changed.

Schematic Element	Description	Changes Allowed
FW_LED	This is the firmware activity LED pin.	Actual control is via the GPIO module APIs. See the APP_FW_LED_ENABLE compile-time option for more information.
VSEL1_P1 VSEL2_P1 VSEL1_P2 VSEL2_P2	These are output pins used to select the source voltage to be provided on the Type-C port.	These can be changed based on the voltage selection mechanism in the target hardware.
VBUS_P_CTRL_P1 VBUS_P_CTRL_P2 VBUS_C_CTRL_P1 VBUS_C_CTRL_P2	Output pins used to control the provider and consumer FETs in the design.	These can be changed based on the FET control mechanism in the target hardware.
VBUS_DISCHARGE_P1 VBUS_DISCHARGE_P2	Output pins used to control the VBus discharge path in the design.	These can be changed based on the discharge control mechanism in the target hardware.
VBUS_MON_P1 VBUS_MON_P2	Input pins used to monitor the voltage on VBus.	No changes are allowed as the connectivity to the internal comparators is fixed.
OCP_FAULT_P1 OCP_FAULT_P2	Input pins that notify CCG4 that an overcurrent condition has been detected.	These can be removed if OCP fault detection circuitry is not available. If used, the names of the pins should not be changed. However, any available GPIO can be used for this purpose.
VBUS_OVP_TRIP_P1 VBUS_OVP_TRIP_P2	Output pins from CCG4 that are used for a fast turn-off of the VBus supply in case of overvoltage.	These can be removed if OVP trip functionality is not used. If used, the names of the signals and their pin mapping should not be changed.

Closely associated with the schematic is the Design Wide Resources (DWR) view, which maps each schematic element to a pin, clock, or hardware block on the CCG4 device. Open the CYPD4125-40LQXI_*notebook.cydwr* file to see the DWR settings for the project.

Figure 17. DWR Project Settings



As shown in Figure 17, the DWR view has several tabs, which configure aspects such as pin mapping, interrupt mapping, clock selection, flash security, and so on. It is recommended that you restrict any changes to the DWR to the pin mapping view. Do not change the clock, interrupt, system, or flash configurations. Even in the pin mapping editor, the changes should be subject to the constraints outlined in Table 7.

4.2.2 Updating Code to Match the Schematic

If you make changes in the schematic or pin mapping, you must make corresponding changes in the firmware code that manages these schematic elements.

All of the schematic-dependent code for the notebook application is implemented in the following files:

1. *CYPD4125-40LQXI_notebook.cydsn/config.h*: This file defines macros that perform hardware-dependent actions such as selecting source voltage and turning FETs ON/OFF. These are implemented as macros because all of these actions involve simple GPIO updates on the CY4541 kit. If required, add a source file, which implements more complex functions to perform these actions.

   Note: There is a similar config file in the noboot.cydsn project folder as well. If debugging is used, the schematic-dependent changes should be replicated there as well.

2. *common/datamux_ctrl.c*: This source file implements a pair of functions that control the Type-C interface switch on the board to select between USB and DisplayPort connections. The default implementation of these functions uses the MUX_CTRL I2C master block within CCG4.

Compile Time Options

The CCG4 Notebook port controller application supports a set of features that can be enabled/disabled using compile time options. These compile time options are set in the config.h header file that you can find under the solution folder, and are summarized in Table 8.

Table 8: Compile Time Options for CCG4 Notebook Application

Option	Description	Values
VBUS_OVP_ENABLE	Enable flag for the internal comparator-based Over Voltage Protection (OVP) scheme. Even if the OVP feature is enabled using this definition, it can be disabled at run-time using the configuration table.	1 for OVP enable 0 for OVP disable
VBUS_OCP_ENABLE	Enable flag for the external load switch based Over Current Protection (OCP) scheme.	1 for OCP enable 0 for OCP disable
VBUS_OVP_TRIP_ENABLE	Enable flag for a direct supply trip capability from CCG hardware on OVP event. Enabling this requires appropriate circuitry on the target hardware.	1 for OVP-TRIP enable 0 for OVP-TRIP disable
SYS_DEEPSLEEP_ENABLE	Enable flag for the low power module which keeps CCG in Deep Sleep mode at all possible times.	1 for low power enable 0 for low power disable
DFP_ALT_MODE_SUPP	Enable flag for Alternate mode support when CCG is a DFP.	1 for alternate mode enable 0 for alternate mode disable
DP_DFP_SUPP	Enable flag for DisplayPort support when CCG is a DFP.	1 for DisplayPort enable 0 for DisplayPort disable
APP_FW_LED_ENABLE	Enable flag for firmware activity LED indication. When enabled, the user LED blinks at 1 second intervals and the user switch cannot be used. Since the LED uses the SWD_IO GPIO, it is necessary to disable it if debugging via SWD. This LED can be used for development support but is recommended to be left in the OFF state to save power in production designs.	1 for LED enable 0 for LED disable

Source Voltage Selection

Refer to the APP_VBUS_SET_XX_PX macros in the *CYPD4125-40LQXI_notebook.cydsn/config.h* file to implement the source voltage selection scheme.

On the CY4541 board, the supported source voltages are 5 V, 9 V, 15 V, and 20 V and a pair of VSEL GPIOs are used to select between them (separate VSEL used for each port).

For example, setting the source voltage on P1 to 15 V is done by the following macro:

```
/* Function/Macro to set P1 source voltage to 15V. */
#define APP_VBUS_SET_15V_P1                                 \
{                                                           \
    VSEL1_P1_Write(0);                                      \
    VSEL2_P1_Write(1);                                      \
}
```

The implementation of this macro can be changed to use the correct mechanism for voltage selection on the target hardware. You can implement the macros for the voltages that are supported from among 5 V, 9 V, 12 V, 13 V, 15 V, 19 V, and 20 V. The implementation for any unsupported voltage can be left as NULL.

FET Control

The provider, consumer, and VBUS discharge FET controls are implemented using the following macros:

- `APP_VBUS_SRC_FET_ON_PX` – Turn provider FET ON
- `APP_VBUS_SRC_FET_OFF_PX` – Turn provider FET OFF
- `APP_VBUS_SNK_FET_ON_PX` – Turn consumer FET ON
- `APP_VBUS_SNK_FET_OFF_PX` – Turn consumer FET OFF
- `APP_DISCHARGE_FET_ON_PX` – Turn VBus Discharge FET ON
- `APP_DISCHARGE_FET_OFF_PX` – Turn VBus Discharge FET OFF

Data Switch / MUX Control

The data switch / MUX control is implemented using the following two functions:

1. mux_ctrl_init: Initialize the MUX / Switch hardware and isolate the Type-C data pins from the USB and DisplayPort connections (ISOLATE mode).

   ```
   /* Initialize the MUX control SCB block. */
   bool mux_ctrl_init(uint8_t port);
   ```

2. mux_ctrl_set_cfg: Configure the data switch to enable the desired data path. The cfg parameter selects between ISOLATE, USB, 2-lane DisplayPort + USB, and 4-Lane DisplayPort modes. The polarity parameter specifies the Type-C connection orientation.

   ```
   /* Update the data mux settings as required. */
   bool mux_ctrl_set_cfg(uint8_t port, mux_select_t cfg, uint8_t polarity);
   ```

These functions are currently implemented using a CCG4 internal I^2C master block to communicate with two different Parade PS8740B switches on the CY4541 kit. These implementations can be changed to make use of the appropriate means for MUX control on the target hardware.

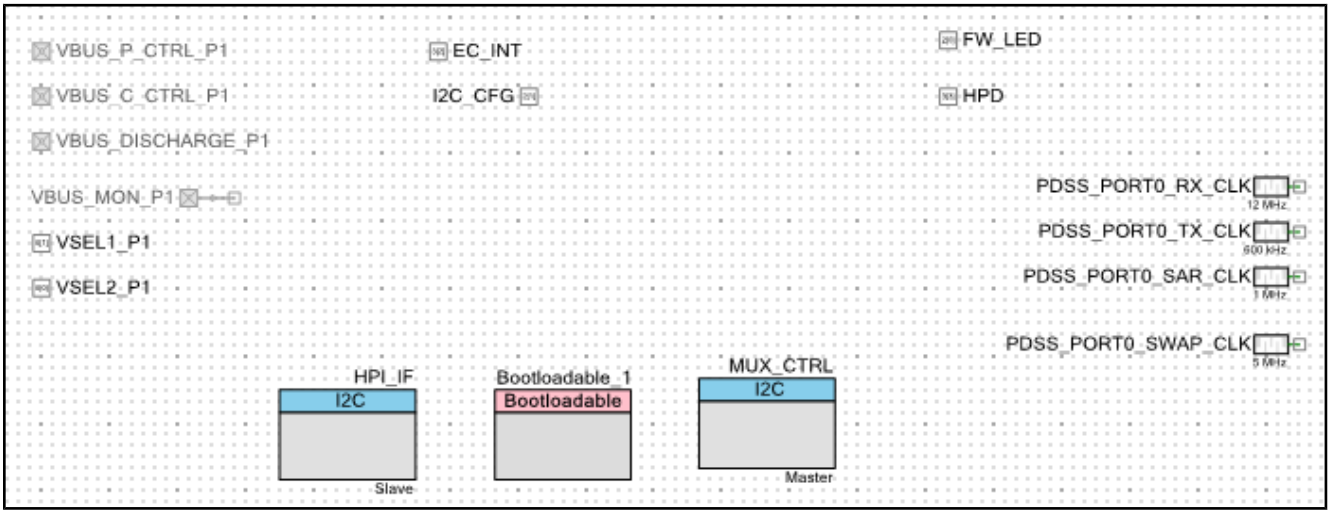
4.2.3 Updating the Default Configuration

The CCG4 notebook firmware project has an embedded default configuration in the *common\config.c* file. The contents of this file can be replaced with that of the *.c* source file generated by EZ-PD Configuration Utility. Once all of the source changes are completed, rebuild the project to generate the customized binaries.

4.3 CCG3 Notebook

4.3.1 PSoC Creator Schematic

Figure 18: PSoC Creator Schematic for CCG3 Notebook



Most aspects of the hardware design around the CCG3 device are captured in the schematics associated with the PSoC Creator firmware project.

The Creator schematic can be found in the *TopDesign.cysch* file, which is part of each Creator project. Double-click on this file to open the schematic editor window (see Figure 18Error: Reference source not found).

The schematic shows how internal resources of the CCG3 device are used in the design. This includes all of the internal clocks used by the design, the various serial interfaces and all of the GPIO pins used to communicate with external elements.

Table 9 shows the various schematic elements used in the CCG3 notebook project. The selection of some of these elements is fixed due to the capabilities of the CCG3 device and the bootloader design. The table also points out the changes allowed in the schematic design.

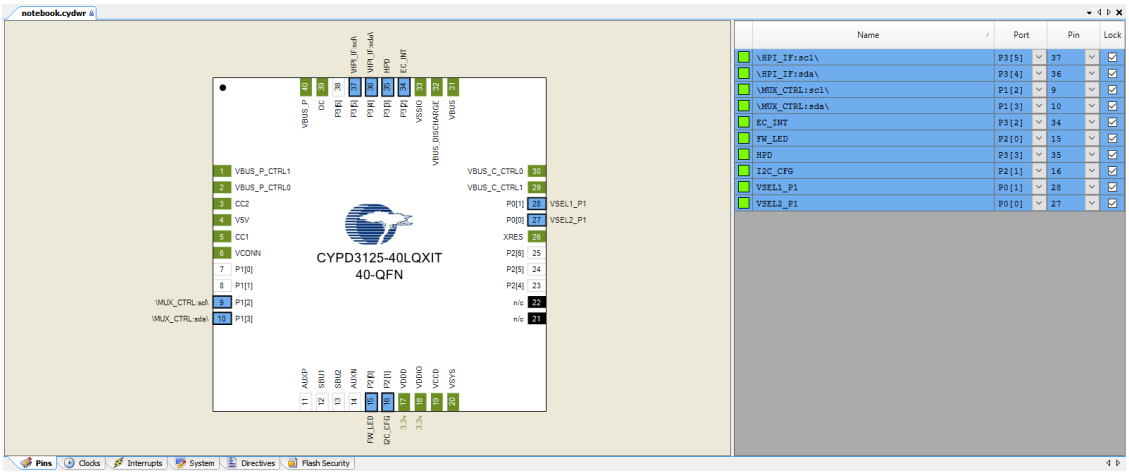
Table 9: Schematic Elements in CCG3 Notebook Design

Schematic Element	Description	Changes allowed
Bootloadable_1	This is a software block which interacts with the boot-loader on the CCG3 device.	No changes should be made to this element.
HPI_IF	This is an I^2C slave block through which the CCG3 communicates with the Embedded Controller in the Notebook design.	No changes are allowed as the HPI_IF is also used by the boot-loader which is fixed.
MUX_CTRL	This is an I^2C master block used by CCG3 to configure the Parade Type-C Interface switch on the CY4531 kit.	This block can be changed / replaced by other mechanisms (such as GPIOs) which can control the interface switch on the target design.

Schematic Element	Description	Changes allowed
PDSS_PORT0_RX_CLK	This is an internal clock that is used for the RX portion of the USB-PD block.	No changes are allowed.
PDSS_PORT0_TX_CLK	This is an internal clock that is used for the TX portion of the USB-PD block.	No changes are allowed.
PDSS_PORT0_SAR_CLK	This is an internal clock that is used for the analog portion of the USB-PD block.	No changes are allowed.
EC_INT	This is an output pin used to interrupt the Embedded Controller when there is a state change.	No changes are allowed as EC_INT is also used by boot-loader.
I2C_CFG	This is an input pin used to select the I^2C slave address used on the HPI interface.	No changes are allowed as EC_INT is also used by boot-loader.
HPD	This is the Hotplug Detect output pin from CCG3 to the DisplayPort controller on the notebook.	This pin can be removed if DisplayPort is not used. If used, the pin mapping cannot be changed.
FW_LED	This is the firmware activity LED pin.	Actual control is via the GPIO module APIs. See the APP_FW_LED_ENABLE compile-time option for more information.
VSEL1_P1 VSEL2_P1	These are output pins used to select the source voltage to be provided on the Type-C port.	These can be changed based on the voltage selection mechanism in the target hardware.

Closely associated with the Schematic is the Design Wide Resources (DWR) view, which maps each schematic element to a pin, clock, or hardware block on the CCG3 device. Open the *CYPD3125-40LQXI_notebook.cydwr* file to see the DWR settings for the project.

Figure 19: Design Wide Resource (DWR) View



As shown in Figure 19, the DWR view has several tabs, which configure aspects such as pin mapping, interrupt mapping, clock selection, flash security, and so on. It is recommended that you restrict any changes to the DWR to the pin mapping view. Do not change the clock, interrupt, system, or flash configurations. Even in the pin mapping editor, the changes should be subject to the constraints outlined in Table 9.

4.3.2 Updating Code to Match the Schematic

If you change the schematic or pin mapping, you must make corresponding changes in the firmware code that manages these schematic elements.

All of the schematic-dependent code for the notebook application is implemented in the following files:

1. *CYPD3125-40LQXI_notebook.cydsn/config.h*: This file defines macros that perform hardware-dependent actions such as selecting source voltage and turning FETs ON/OFF. These are implemented as macros because all of these actions involve simple GPIO or device register updates on the CY4531 kit. If required, the user can add a source file which implements more complex functions to perform these actions.

   Note: There is a similar config file in the noboot.cydsn project folder as well. If debugging is being used, the schematic dependent changes should be replicated there as well.

2. *common/datamux_ctrl.c*: This source file implements a pair of functions that control the Type-C interface switch on the board to select between USB and DisplayPort connections. The default implementation of these functions uses the MUX_CTRL I2C master block within CCG3.

Compile Time Options

The CCG3 Notebook port controller application supports a set of features that can be enabled/disabled using compile time options. These compile time options are set in the config.h header file that you can find under the solution folder, and are summarized inTable 10 Table 10.

Table 10: Selectable Firmware Features in CCG3 Notebook Application

Pre-processor Switch	Description	Values
VBUS_OVP_ENABLE	Enable overvoltage Protection handling on VBus. This feature can be turned off using the configuration table, even if it is enabled here.	1 for VBus OVP enable 0 for VBus OVP disable
VBUS_OVP_AUTO_CONTROL_ENABLE	Enable automatic FET control by hardware when an OVP event is detected.	1 for automatic hardware cut-off 0 for firmware cut-off
SYS_DEEPSLEEP_ENABLE	Enable flag for the low power module which keeps CCG in deep sleep mode at all possible times.	1 for low power enable 0 for low power disable
DFP_ALT_MODE_SUPP	Enable Alternate Mode handling when CCG is DFP.	1 for alternate mode enable 0 for alternate mode disable
DP_DFP_SUPP	Enable DisplayPort Alternate mode when CCG is DFP. This requires DFP_ALT_MODE_SUPP.	1 for DisplayPort enable 0 for DisplayPort disable
APP_FW_LED_ENABLE	Enable flag for firmware activity LED indication. When enabled, the user LED blinks at 1 second intervals and the user switch cannot be used. Since the LED uses the SWD_IO GPIO, it is necessary to disable it if debugging via SWD. This LED can be used for development support but is recommended to be left in the OFF state to save power in production designs.	1 for LED enable 0 for LED disable

Source Voltage Selection

Refer to the APP_VBUS_SET_XX_PX macros in the *CYPD3125-40LQXI_note*book.cydsn/config.h file to implement the source voltage selection scheme.

On the CY4531 board, the supported source voltages are 5 V, 9 V, 15 V, and 20 V and a pair of VSEL GPIOs are used to select between them (separate VSEL used for each port).

For example, setting the source voltage on P1 to 15 V is done by the following macro:

```
/* Function/Macro to set P1 source voltage to 15V. */
#define APP_VBUS_SET_15V_P1                                 \
{                                                           \
    VSEL1_P1_Write(0);                                      \
    VSEL2_P1_Write(1);                                      \

}
```

The implementation of this macro can be changed to use the correct mechanism for voltage selection on the target hardware. The user can implement the macros for the voltages that are supported from among 5 V, 9 V, 12 V, 13 V, 15 V, 19 V, and 20 V. The implementation for any unsupported voltage can be left as NULL.

FET Control

The provider, consumer, and VBUS discharge FET controls are implemented using the following macros:

- `APP_VBUS_SRC_FET_ON_PX – Turn provider FET ON`
- `APP_VBUS_SRC_FET_OFF_PX – Turn provider FET OFF`
- `APP_VBUS_SNK_FET_ON_PX – Turn consumer FET ON`
- `APP_VBUS_SNK_FET_OFF_PX – Turn consumer FET OFF`
- `APP_DISCHARGE_FET_ON_PX – Turn VBus Discharge FET ON`
- `APP_DISCHARGE_FET_OFF_PX – Turn VBus Discharge FET OFF`
- Data Switch / MUX Control

The data switch / MUX control is implemented using the following two functions:

1. mux_ctrl_init: Initialize the MUX/Switch hardware and isolate the Type-C data pins from the USB and DisplayPort connections (ISOLATE mode).

```
/* Initialize the MUX control SCB block. */
bool mux_ctrl_init(uint8_t port);
```

2. mux_ctrl_set_cfg: Configure the data switch to enable the desired data path. The cfg parameter selects between ISOLATE, USB, 2-lane DisplayPort + USB and 4-Lane DisplayPort modes. The polarity parameter specifies the Type-C connection orientation.

```
/* Update the data mux settings as required. */
bool mux_ctrl_set_cfg(uint8_t port, mux_select_t cfg, uint8_t polarity);
```

These functions are currently implemented using a CCG3 internal I^2C master block to communicate with two different Parade PS8740B switches on the CY4531 kit. These implementations can be changed to make use of the appropriate means for MUX control on the target hardware.

4.3.3 Updating the Default Configuration

The CCG3 notebook firmware project has an embedded default configuration in the common\config.c file. The contents of this file can be replaced with that of the .c source file generated by EZ-PD Configuration Utility. After all the source changes are completed, rebuild the project to generate the customized binaries.

4.4 CCG3 Type C to DP or HDMI/DVI/VGA Dongle

CCG3 TypeC to DP or HDMI/DVI/VGA Dongle is Alternate Mode Adapter which can connect a TYPE-C Display Source to DP or HDMI/DVI/VGA Display Sink. It is a bus powered device and can be powered either from VBUS or VCONN. Dongle FW uses a GPIO to determine Display side configuration i.e. DP or HDMI/DVI/VGA. Dongle uses integrated SBU-AUX switch, AUX pull- up/pull-down resistors and billboard controller. It supports FW update interface over HID interface.

4.4.1 PSoC Creator Schematic

Figure 20: PSoC Creator Schematic for CCG3 Dongle

Open TopDesign.cysch file in the PSoC creator project to access dongle's schematic. Schematic contains hardware resources used by dongle such as clocks, Hot Plug Detect IO etc.

Figure 21Error: Reference source not found shows the various schematic elements used in the CCG3 dongle project. The selection of some of these elements is fixed due to the capabilities of the CCG3 device and the bootloader design. The table also points out the changes allowed in the schematic design.

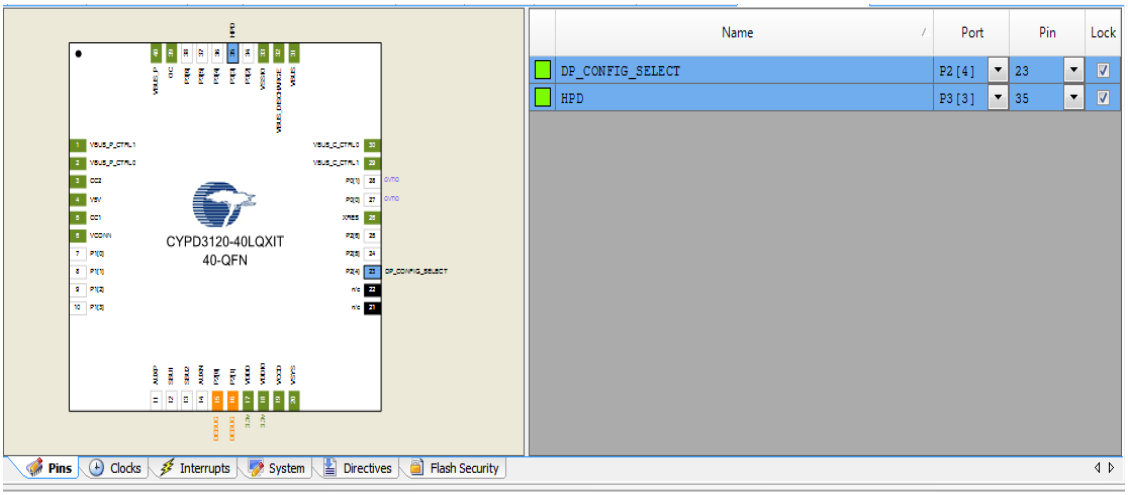
Table 11: Schematic Elements in CCG3 Dongle Design

Schematic Element	Description	Changes allowed
Bootloadable_1	This is a software block which interacts with the boot-loader on the CCG3 device.	No changes should be made to this block.
HPD	This is Hot Plug Detect IO which is used by dongle to detect HPD Plug, Unplug and IRQ events.	No changes are allowed as this GPIO is routed to internal HPD hardware block.
DP_CONFIG_SELECT	This is a GPIO which is used by dongle to choose between DP and HDMI/DVI/VGA configuration. Dongle FW samples the state of this IO in initialization sequence. If state of IO is detected as low, FW chooses HDMI/DVI/VGA configuration. Otherwise (float/high) it chooses DP configuration.	No changes are allowed as FW functionality depends on this GPIO.
PDSS_PORT0_RX_CLK	This is an internal clock that is used for the RX portion of the USB-PD block.	No changes are allowed.

Schematic Element	Description	Changes allowed
PDSS_PORT0_TX_CLK	This is an internal clock that is used for the TX portion of the USB-PD block.	No changes are allowed.
PDSS_PORT0_SAR_CLK	This is an internal clock that is used for the analog portion of the USB-PD block.	No changes are allowed.

Closely associated with the Schematic is the Design Wide Resources (DWR) view, which maps each schematic element to a pin, clock, or hardware block on the CCG3 device. Open the *CY{D3120-40LQXI_dp_dongle.cydwr* file to see the DWR settings for the project.

Figure 21: Design Wide Resource (DWR) View



As shown in Figure 21Error: Reference source not found, the DWR view has several tabs, which configure aspects such as pin mapping, interrupt mapping, clock selection, flash security, and so on. It is recommended to not change the clock, interrupt, system, or flash configurations. Even in the pin mapping editor, the changes should be subject to the constraints outlined in Error: Reference source not found.

4.4.2 Compile Time Options

The CCG3 Dongle application supports a set of features that can be enabled/disabled using compile time options. These compile time options are set in the config.h header file, and are summarized in Table 12Error: Reference source not found.

Table 12: Selectable Firmware Features in CCG3 Dongle Application

Pre-processor Switch	Description	Values
`VBUS_OVP_ENABLE`	Enable overvoltage Protection handling on VBus. This feature can be turned off using the configuration table, even if it is enabled here.	1 for VBus OVP enable 0 for VBus OVP disable
`SYS_DEEPSLEEP_ENABLE`	Enable flag for the low power module which keeps CCG in deep sleep mode at all possible times.	1 for low power enable 0 for low power disable

Pre-processor Switch	Description	Values
CCG_BB_ENABLE	Enable billboard functionality. This feature can be turned off using the configuration table, even if it is enabled here.	1 for billboard enable 0 for billboard disable
FLASHING_MODE_USB_ENABLE	Enable FW update interface over USB HID. It is recommended to not disable this option as USB is the only interface available for FW update in dongle. If CCG_BB_ENABLE is 0, this option gets automatically disabled.	1 for FW update interface enable 0 for FW update interface disable
FLASH_ENABLE_NB_MODE	Enable non-blocking flash row update feature. CCG3 supports non-blocking row update which allows other interfaces to be active while device's flash is being updated.	1 for non-blocking FW update 0 for blocking FW update
DP_GPIO_CONFIG_SELECT	Enable GPIO bases selection of Display configuration: DP or HDMI/DVI/VGA.	1 for GPIO based selection 0 for fixed configuration: DP only

4.4.3 Functional Overview

3. CCG3 Dongle implements 4 lane TYPE-C to DP or HDMI/DVI/VGA alternate mode adapter. Display configuration is selected based on GPIO 23. Dongle updates Discover Mode response based on GPIO status. In DP mode, dongle advertises DP Pin E Configuration and in HDMI/DVI/VGA mode it advertises Pin C Configuration.
4. It has integrated billboard controller. Billboard related configuration can be updated through configuration table update. Dongle advertises HID interface along with billboard interface which is used for FW update.
5. It has integrated SBU-AUX switch which is used to connect SBU-AUX lines based on CC polarity. Refer source file datamux_ctrl.c.
6. It has integrated AUX pull-up/pull-down resistors. Value of the resistors applied depends on Display configuration: DP or HDMI/DVI/VGA. Refer datamux_ctrl.c.
7. It uses HPD hardware block to receive and queue multiple HPD events i.e. Plug, Unplug and IRQ.

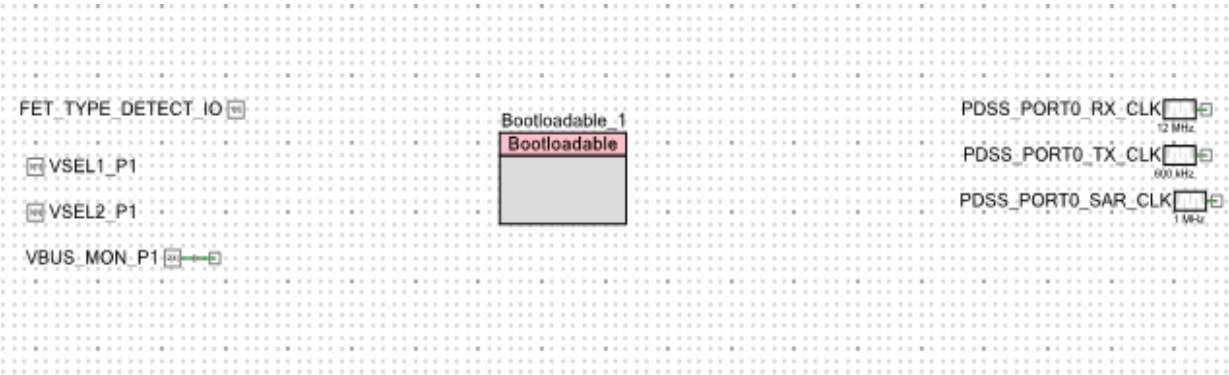
4.4.4 Updating the Default Configuration

The CCG3 dongle firmware project has an embedded default configuration in the *common\config.c* file. The contents of this file can be replaced with that of the .c source file generated by EZ-PD Configuration Utility. After all the source changes are completed, rebuild the project to generate the customized binaries.

4.5 CCG3 Power Adapter

4.5.1 PSoC Creator Schematic

Figure 22: PSoC Creator Schematic for CCG3 Power Adapter



Open TopDesign.cysch file in the PSoC creator project to access power adapter's schematic. Schematic contains hardware resources used by power adapter such as clocks, voltage selection IOs etc.

Figure 22Error: Reference source not found shows the various schematic elements used in the CCG3 power adapter project. The selection of some of these elements is fixed due to the capabilities of the CCG3 device and the bootloader design. The table also points out the changes allowed in the schematic design.

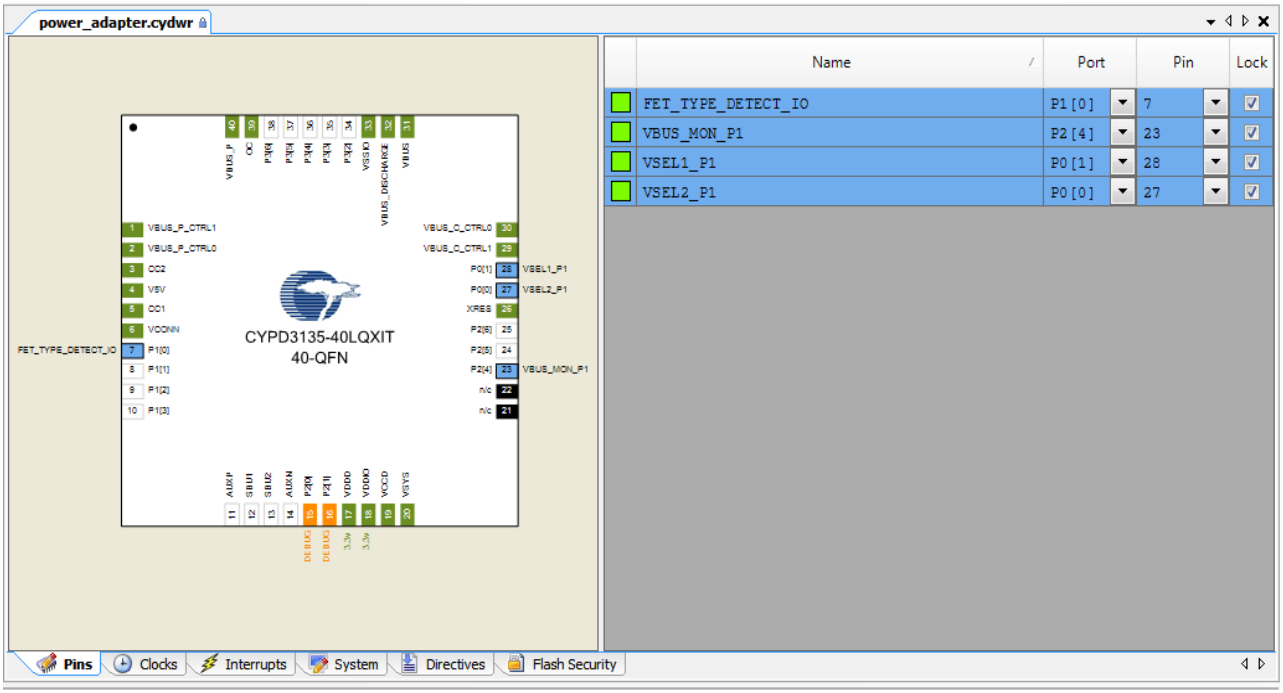
Table 13: Schematic Elements in CCG3 Power Adapter Design

Schematic Element	Description	Changes allowed
Bootloadable_1	This is a software block which interacts with the boot-loader on the CCG3 device.	No changes should be made to this element.
PDSS_PORT0_RX_CLK	This is an internal clock that is used for the RX portion of the USB-PD block.	No changes are allowed.
PDSS_PORT0_TX_CLK	This is an internal clock that is used for the TX portion of the USB-PD block.	No changes are allowed.
PDSS_PORT0_SAR_CLK	This is an internal clock that is used for the analog portion of the USB-PD block.	No changes are allowed.
VSEL1_P1 VSEL2_P1	These are output pins used to select the source voltage to be provided on the Type-C port.	These can be changed based on the voltage selection mechanism in the target hardware.
VBUS_MON_P1	Input pin used to monitor the voltage on VBus.	No changes are allowed as the connectivity to the internal comparators is fixed.

Schematic Element	Description	Changes allowed
FET_TYPE_DETECT_IO	Input pin used to detect type of voltage provider FETs.	No change is allowed as FW behavior depends on this IO. If the IO is floating, then NFETs are being used. If the IO is grounded, then PFETs are being used.

Closely associated with the Schematic is the Design Wide Resources (DWR) view, which maps each schematic element to a pin, clock, or hardware block on the CCG3 device. Open the *CYPD3135-40LQXI_power_adapter.cydwr* file to see the DWR settings for the project.

Figure 23: Design Wide Resource (DWR) View



As shown in Error: Reference source not found, the DWR view has several tabs, which configure aspects such as pin mapping, interrupt mapping, clock selection, flash security, and so on. It is recommended that you restrict any changes to the DWR to the pin mapping view. Do not change the clock, interrupt, system, or flash configurations. Even in the pin mapping editor, the changes should be subject to the constraints outlined in Error: Reference source not found.

4.5.2 Updating Code to Match the Schematic

If you change the schematic or pin mapping, you must make corresponding changes in the firmware code that manages these schematic elements.

All of the schematic-dependent code for the power adapter application is implemented in the following file:

8. *CYPD3135-40LQXI_power_adpater.cydsn/config.h*: This file defines macros that perform hardware-dependent actions such as selecting source voltage and turning FETs ON/OFF. These are implemented as macros because all of these actions involve simple GPIO or device register updates. If required, the user can add a source file which implements more complex functions to perform these actions.

   Note: There is a similar config file in the noboot.cydsn project folder as well. If debugging is being used, the schematic dependent changes should be replicated there as well.

Compile Time Options

The CCG3 power adapter port controller application supports a set of features that can be enabled/disabled using compile time options. These compile time options are set in the config.h header file that you can find under the solution folder, and are summarized in Error: Reference source not found.

Table 14: Selectable Firmware Features in CCG3 Power Adapter Application

Pre-processor Switch	Description	Values
VBUS_OVP_ENABLE	Enable overvoltage Protection handling on VBus. This feature can be turned off using the configuration table, even if it is enabled here.	1 for VBus OVP enable 0 for VBus OVP disable
VBUS_OCP_ENABLE	Enable overcurrent protection on VBus. This feature can be turned off using the configuration table, even if it is enabled here.	1 for VBus OCP enable 0 for VBus OCP disable
VBUS_OCP_MODE	This parameter selects various OCP handling modes.	0: OCP is detected by external OCP hardware. 1: Internal OCP with neither software debouce nor automatic FET control by hardware. 2: Internal OCP with automatic FET control by hardware. 3: Internal OCP with software debounce using delay in milliseconds from the configuration table.
SYS_DEEPSLEEP_ENABLE	Enable flag for the low power module which keeps CCG in deep sleep mode at all possible times.	1 for low power enable 0 for low power disable
FLASHING_MODE_PD_ENABLE	Enable FW update interface over PD. Recommendation is to keep this enabled as power adapter supports FW update interface only over PD.	1 for enabling FW update interface 0 for disabling FW update interface
FLASH_ENABLE_NB_MODE	Enable non-blocking flash row update feature. CCG3 supports non-blocking row update which allows other interfaces to be active while device's flash is being updated.	1 for non-blocking FW update 0 for blocking FW update

Source Voltage Selection

Refer to the APP_VBUS_SET_XX_PX macros in the *power_adpater*.cydsn/config.h file to implement the source voltage selection scheme.

In CCG3 power adapter FW, the supported source voltages are 5 V, 9 V, 15 V, and 20 V and a pair of VSEL GPIOs are used to select between them.

For example, setting the source voltage to 15 V is done by the following macro:

```
/* Function/Macro to set source voltage to 15V. */
#define APP_VBUS_SET_15V_P1                             \
{                                                       \
    VSEL1_P1_Write(0);                                  \
    VSEL2_P1_Write(1);                                  \

}
```

The implementation of this macro can be changed to use the correct mechanism for voltage selection on the target hardware. The user can implement the macros for the voltages that are supported from among 5 V, 9 V, 12 V, 13 V, 15 V, 19 V, and 20 V. The implementation for any unsupported voltage can be left as NULL.

FET Control

The provider and VBUS discharge FET controls are implemented using the following macros:

- `APP_VBUS_SRC_FET_ON_PX` – Turn provider FET ON
- `APP_VBUS_SRC_FET_OFF_PX` – Turn provider FET OFF
- `APP_DISCHARGE_FET_ON_PX` – Turn VBus Discharge FET ON
- `APP_DISCHARGE_FET_OFF_PX` – Turn VBus Discharge FET OFF

4.5.3 Updating the Default Configuration

The CCG3 power adapter firmware project has an embedded default configuration in the *common\config.c* file. The contents of this file can be replaced with that of the .c source file generated by EZ-PD Configuration Utility. After all the source changes are completed, rebuild the project to generate the customized binaries.

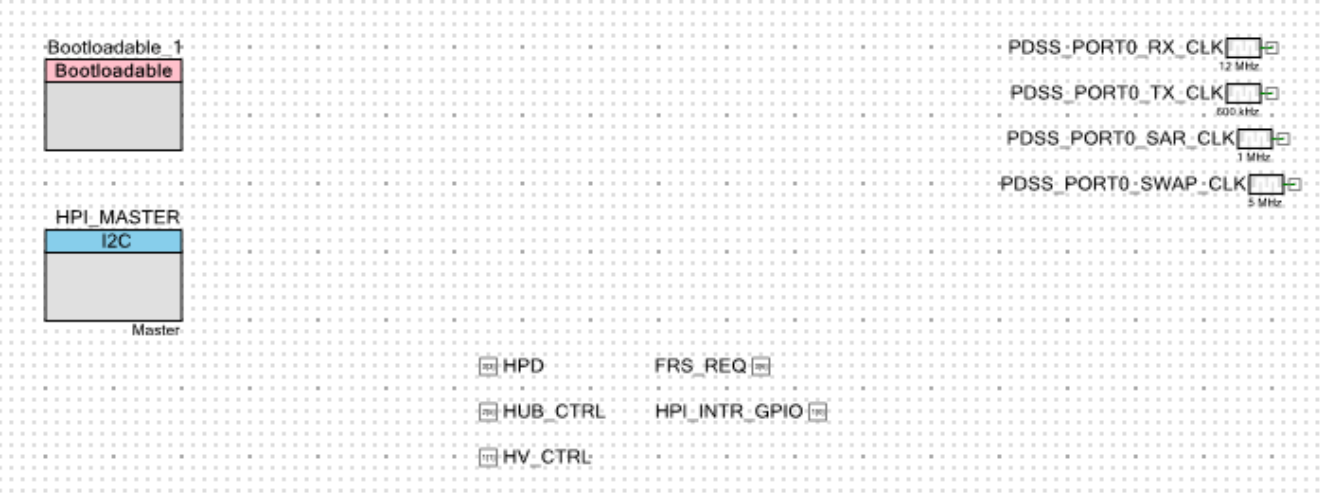
4.5.4 FW update interface

CCG3 Power Adapter supports dual FW images and non-blocking flash update feature to allow FW update without affecting normal operation of the device. PD is the FW update interface. Host (Flashing controller) needs to swap data roles with power adapter and enter CY FW update alternate mode before initiating FW update procedure. Power adapter bootloader does not support any FW update interface.

4.6 CCG3 Charge-Through Dongle

4.6.1 PSoC Creator Schematic

Figure 24: PSoC Creator Schematic for CCG3 Charge-Through Dongle



Most aspects of the hardware design around the CCG3 device are captured in the schematics associated with the PSoC Creator firmware project.

The Creator schematic can be found in the *TopDesign.cysch* file, which is part of each Creator project. Double-click on this file to open the schematic editor window (see Figure 24).

The schematic shows how internal resources of the CCG3 device are used in the design. This includes all of the internal clocks used by the design, the various serial interfaces and all of the GPIO pins used to communicate with external elements.

Table 15 shows the various schematic elements used in the CCG3 Charge-Through Dongle upstream port controller project. The selection of some of these elements is fixed due to the capabilities of the CCG3 device and the bootloader design. The table also points out the changes allowed in the schematic design.

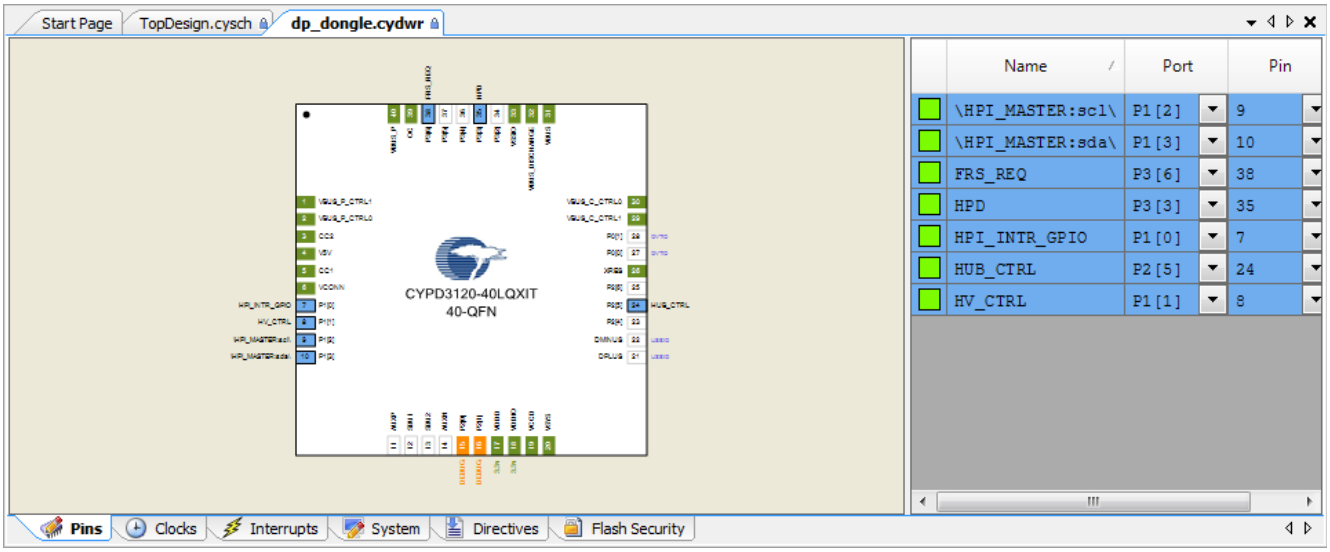
Table 15: Schematic Elements in CCG3 Charge-Through Dongle Design

Schematic Element	Description	Changes allowed
Bootloadable_1	This is a software block which interacts with the boot-loader on the CCG3 device.	No changes should be made to this element.
HPI_MASTER	This is an I^2C master block through which the CCG3 communicates with the PD port controller for the downstream port.	The pins used can be changed to any other I2C (SCB) blocks on the CCG3 device.
PDSS_PORT0_RX_CLK	This is an internal clock that is used for the RX portion of the USB-PD block.	No changes are allowed.
PDSS_PORT0_TX_CLK	This is an internal clock that is used for the TX portion of the USB-PD block.	No changes are allowed.
PDSS_PORT0_SAR_CLK	This is an internal clock that is used for the analog portion of the USB-PD block.	No changes are allowed.
PDSS_PORT0_SWAP_CLK	This is an internal clock that is used for the fast role swap logic in the USB-PD block.	No changes are allowed.
HPI_INTR_GPIO	This is an input pin used by the downstream port controller to notify CCG3 about downstream port state changes.	Can be moved to other free GPIOs on the CCG3 device.
FRS_REQ	This is an input pin which is used to trigger a PD 3.0 fast-role swap from the CCG3 device.	No changes are allowed as the FRS trigger pin on CCG3 is fixed.
HPD	This is the Hotplug Detect input pin from the DisplayPort driver to the CCG3 device.	This pin can be removed if DisplayPort is not used. If used, the pin mapping cannot be changed.
HUB_CTRL	This is an output pin used to enable the USB 3.0 hub in the dongle design.	Should not be changed because this pin is used by the fixed function boot-loader as well.

Schematic Element	Description	Changes allowed
HV_CTRL	This is an output pin used to enable the pass-through connection of the downstream port power supply to the upstream port.	Can be moved to other free GPIOs on the CCG3 device.

Closely associated with the Schematic is the Design Wide Resources (DWR) view, which maps each schematic element to a pin, clock, or hardware block on the CCG3 device. Open the *CYPD3123-40LQXI_ctd_us.cydwr* file to see the DWR settings for the project.

Figure 25: Design Wide Resource (DWR) View



As shown in Figure 25, the DWR view has several tabs, which configure aspects such as pin mapping, interrupt mapping, clock selection, flash security, and so on. It is recommended that you restrict any changes to the DWR to the pin mapping view. Do not change the clock, interrupt, system, or flash configurations. Even in the pin mapping editor, the changes should be subject to the constraints outlined in Table 15.

4.6.2 Updating Code to Match the Schematic

If you change the schematic or pin mapping, you must make corresponding changes in the firmware code that manages these schematic elements.

All of the schematic-dependent code for the charge-through dongle application is implemented in the following files:

1. *common/ctd_us_solution.c*: This file makes use of the HV_CTRL GPIO to control the power circuits as required. These operations can be changed as required based on the hardware design.
2. *common/datamux_ctrl.c*: This source file implements a function that controls the internal SBU MUX on the CCG3 device and the external HUB enable GPIO based on the active PD alternate modes.

Compile Time Options

The CCG3 Charge-Through Dongle upstream port controller application supports a set of features that can be enabled/disabled using compile time options. These compile time options are set in the config.h header file that you can find under the solution folder, and are summarized in Table 16.

Table 16: Selectable Firmware Features in CCG3 Charge-Through Dongle Application

Pre-processor Switch	Description	Values
VBUS_OVP_ENABLE	Enable overvoltage Protection handling on VBus. This feature can be turned off using the configuration table, even if it is enabled here.	1 for VBus OVP enable 0 for VBus OVP disable
VBUS_OVP_AUTO_CONTROL_ENABLE	Enable automatic FET control by hardware when an OVP event is detected.	1 for automatic hardware cut-off 0 for firmware cut-off
SYS_DEEPSLEEP_ENABLE	Enable flag for the low power module which keeps CCG in deep sleep mode at all possible times.	1 for low power enable 0 for low power disable
UFP_ALT_MODE_SUPP	Enable Alternate Mode handling when CCG is UFP.	1 for alternate mode enable 0 for alternate mode disable
DP_UFP_SUPP	Enable DisplayPort Alternate mode when CCG is UFP. This requires UFP_ALT_MODE_SUPP.	1 for DisplayPort enable 0 for DisplayPort disable
CCG_BB_ENABLE	Enable the internal Billboard USB device operation.	1 for Billboard enable 0 for Billboard disable

4.6.3 Updating the Default Configuration

The CCG3 charge through dongle firmware project has an embedded default configuration in the *common\config.c* file. The contents of this file can be replaced with that of the .c source file generated by EZ-PD Configuration Utility. After all the source changes are completed, rebuild the project to generate the customized binaries.

4.7 USB-PD Specification Revisions

The example applications provided for CCG3 and CCG4 devices support USB-PD specification revision 3.0 by default. Since PD 3.0 support requires significant code addition, this leaves little room for the addition of customer specific code in these applications.

It is possible to gain more space in the CCG device flash by restricting the applications to USB-PD Revision 2.0 support. The following steps are required to switch applications between PD 3.0 and PD 2.0 support:

1. The PD stack parameters configuration file (stack_params.h) has a few pre-processor definitions that enable PD 3.0 support in the application. The definitions CCG_PD_REV3_ENABLE, CCG_FRS_RX_ENABLE and CCG_FRS_TX_ENABLE should be set to 0 to disable PD 3.0 support.
2. Two versions of the PD stack libraries are provided: libccgx_pd.a and libccgx_pd3.a. In the linker settings sections of the build settings of the project, switch between ccgx_pd and ccgx_pd3 to switch between PD 2.0 and PD 3.0 support.
3. The configuration table contents for the application should be changed based on the specification version to be supported. The SRC_PDO, SNK_PDO and DISCOVER_ID response parameters in the configuration table have fields that are defined only for PD 3.0. These values should be adjusted as required when switching between PD revisions.

Error: Reference source not found Design

Error: Reference source not foundError: Reference source not foundError: Reference source not foundpplication

Functional Overview

Error: Reference source not found Design

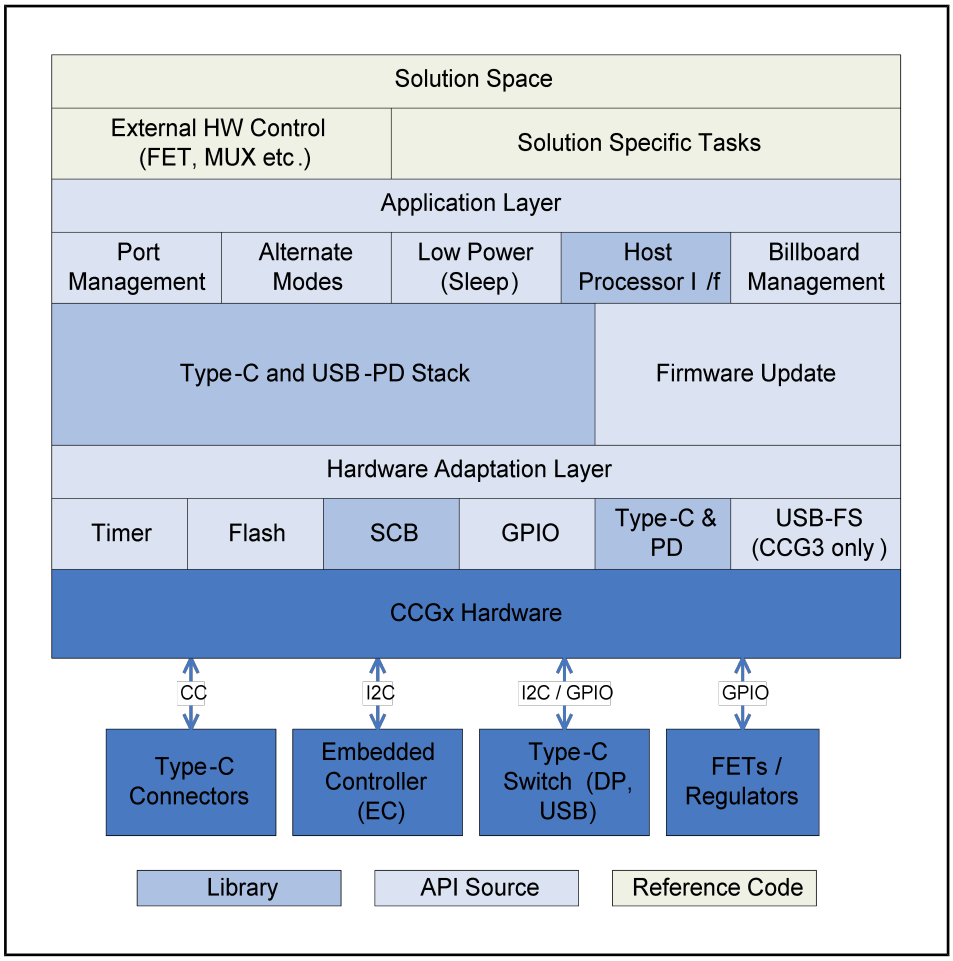
Error: Reference source not foundError: Reference source not foundError: Reference source not found

5. Firmware Architecture

5.1 Firmware Blocks

The CCGx firmware architecture allows users to implement a variety of USB-PD applications using the CCG3 and CCG4 devices and a fully tested firmware stack. A block diagram of the CCGx firmware architecture is shown in The CCGx firmware architecture allows users to implement a variety of USB-PD applications using the CCG3 and CCG4 devices and a fully tested firmware stack. A block diagram of the CCGx firmware architecture is shown inThe CCGx firmware architecture allows users to implement a variety of USB-PD applications using the CCG3 and CCG4 devices and a fully tested firmware stack. A block diagram of the CCGx firmware architecture is shown inThe CCGx firmware architecture allows users to implement a variety of USB-PD applications using the CCG3 and CCG4 devices and a fully tested firmware stack. A block diagram of the CCGx firmware architecture is shown inThe CCGx firmware architecture allows users to implement a variety of USB-PD applications using the CCG3 and CCG4 devices and a fully tested firmware stack. A block diagram of the CCGx firmware architecture is shown in . Figure 26. Figure 26. Figure 26. Figure 26.

Figure 26: CCGx Firmware Block Diagram



The CCGx firmware architecture contains the following components:

- Hardware Adaptation Layer (HAL): This includes the low-level drivers for the various hardware blocks on the CCG device. This includes drivers for the Type-C and USB-PD block, Serial Communication Blocks (SCBs), GPIOs, flash module, timer module and USB full speed device module (only for CCG3 Dongle).
- USB Type-C and USB-PD Protocol Stack: This is the complete USB-PD protocol stack that includes the Type-C and USB-PD port managers, USB-PD protocol layer, the USB-PD policy engine, and the device policy manager. The device policy manager is designed to allow all policy decisions to be made at the application level, either on an external Embedded Controller (EC) or in the CCG firmware itself.
- Firmware update module: This is a firmware module that allows the device firmware maintained in internal flash to be updated. In Notebook PD port controller applications, the firmware update will be done from the EC side through an I2C interface.
- Billboard Management: This module handles all the billboard enumeration sequences and is applicable only for CCG3 Dongle implementations. The billboard module is used internally by the Alternate Modes modules and is not expected to be invoked explicitly.
- Host Processor Interface (HPI): The Host Processor Interface (HPI) is an I2C-based control interface that allows an Embedded Controller (EC) to monitor and control the USB-PD port on the CCG device. The HPI is the means to allow the PC platform to control the PD policy management.
- Port Management: This module handles all of the PD port management functions including the algorithm for optimal contract negotiations, source and sink power control, source voltage selection, port role assignment, and swap request handling.

- Alternate Modes: This module implements the alternate mode handling for CCG as a DFP and UFP. A fully tested implementation of DisplayPort alternate mode with CCG as DFP is provided. The module also allows users to implement their own alternate mode support in both DFP and UFP modes.
- Low Power: This module attempts to keep the CCG device in the low-power standby mode as often as possible to minimize power consumption.
- External Hardware Control: This is a hardware design-dependent module, which controls the external hardware blocks such as FETs, regulators, and Type-C switches.
- Solution specific tasks: This is an application layer module where any custom tasks required by the user solution can be implemented.

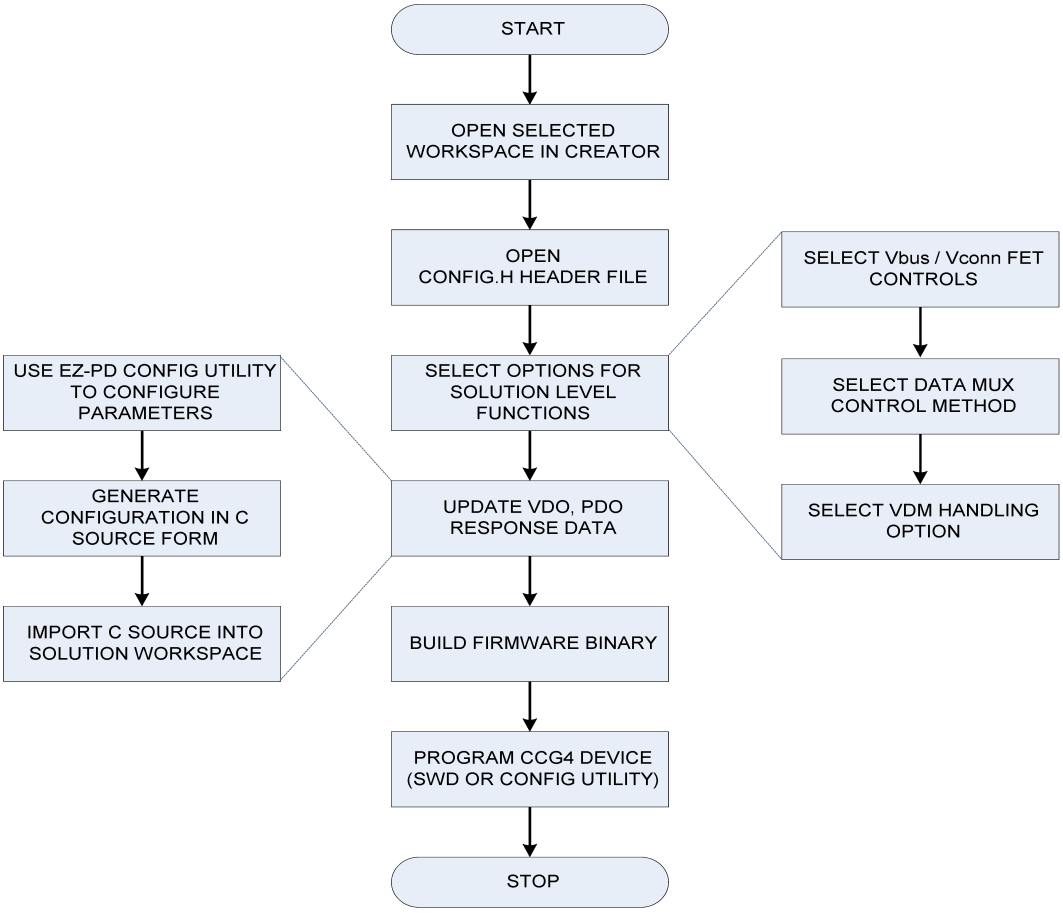
5.2 SDK Usage Model

Users of the CCG notebook solution must follow these steps to use the SDK components:

1. Load the solution workspace using PSoC Creator.
2. Edit the project schematics and solution configuration header file if needed.
3. Use the EZ-PD Configuration Utility to build the configuration table, and copy the generated C source file into the Creator project if necessary. The configuration table can also be updated by editing the *config.c* file in PSoC Creator Source Editor.
4. Build the application projects using the PSoC Creator. The firmware binaries will be generated in ELF, HEX, and CYACD formats suitable for SWD programming, Miniprog, and the EZ-PD configuration utility.
5. Load the firmware binary onto the target hardware for evaluation and testing.

This usage flow is illustrated in Figure 27. Many of these steps, such as changing the compile time configurations and using the EZ-PD Configuration Utility to change the configuration table, are only required if the customer wants to change the way the application works.

Figure 27. SDK Usage Flow



5.3 Firmware Versioning

Each project has a firmware version (base version) and an application version number.

The base firmware version number shall consist of major number, minor number, and patch number in addition to an automatically updated build number.

The base firmware version applies to the whole stack and is common for all applications and projects using the stack. The version information can be found in the ***src/system/ccgx_version.h*** header file.

The application version shall be modified for individual customers based on requirements. This shall have a major version, minor version, external circuit specification, and application name. This version information can be updated by users as required, and is located in the ***Firmware/projects/<project_name>/common/app_version.h*** header file.

Note: Ensure that you do not change the application name from the value defined for the CCG application type. The application type information is used by the EZ-PD configuration utility to interpret the configuration table content.

The version number information for each firmware shall be stored in an eight-byte data field and shall be retrieved over the HPI interface. The following table denotes the version structure and format.

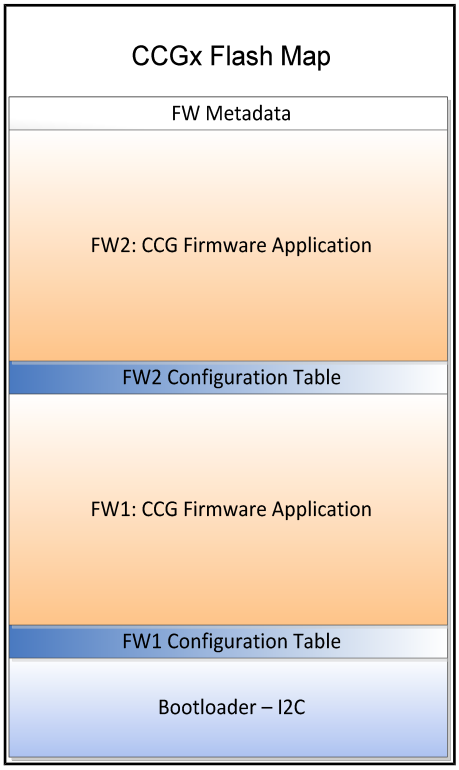
Table 17: CCGx Firmware Version Structure

Bit Field	Name	Description
[15:0]	Base FW Build number	This field corresponds to base firmware version and shall be automatically incremented during nightly build. This field should not be manually edited. This field is expected to be reset on every SNPP release cycle and not modified throughout the release.
[23:16]	Base FW Patch version number	This field corresponds to base firmware patch version number. This field shall be updated manually by the core PD team for base firmware releases. This field shall be incremented for every intermediate release done to customer or an actual patch release performed for a previous full release.
[27:24]	Base FW Minor version number	This field corresponds to base firmware minor version number. This field shall be updated manually by the core PD team for base firmware releases. This field is generally updated once for every SNPP release cycle at ES100 RC build. The exception is when an intermediate customer release which breaks compatibility.
[31:28]	Base FW Major version number	This field corresponds to base firmware major version number. This field shall be updated manually by the core PD team for base firmware releases. The major number is generally updated on a major project level change or when we have cycled through all minor numbers. The number shall be determined at the beginning of every SNPP release cycle.
[47:32]	Application Name / number	This field is left for any application / customer-specific changes to be done by applications team. By default, this field shall be released by the base firmware version team will have the following values: Notebook: "nb" Power Adapter: "pa" Alternate Mode Adapter (AMA): "aa" **NOTE**: This information is used by Ez-PD Configuration Utility to determine the application type and should not be modified for standard applications.
[55:48]	External circuit number	This field is left for any application / customer-specific changes to be done by applications team. By default, this field shall be released by the base firmware team as 0. The circuit number values from 0x00 to 0x1F are reserved for base firmware team. This is because base firmware team may have to support same application on multiple platforms in the future.
[59:56]	Application minor version number	This field is left for any application / customer-specific changes to be done by applications team. By default, this field shall be released by the base firmware team as 0
[63:60]	Application major version number	This field is left for any application / customer-specific changes to be done by applications team. By default, this field shall be released by the base firmware team as 0

5.4 Flash Memory Map

CCGx has a 128-KB flash memory that is designated to store a bootloader, along with two copies of the firmware binary along with the corresponding configuration table. The flash memory map for the device is shown in Figure 28.

Figure 28: CCGx Flash Memory Map



The bootloader in notebook and dongle applications is used to upgrade the CCGx application firmware. Bootloader in power adapter application does not support firmware update interface. It is allocated a fixed area. This memory area can only be written to from the SWD interface. CCG4 Notebook only supports I^2C bootloader and uses 5 KB of memory. CCG3 Notebook also supports I^2C bootloader and uses 6KB of memory. CCG3 Dongle supports only USB-HID bootloader and uses 6KB of memory.

The configuration table holds the default PD configuration for the CCG application and is located at the beginning of each firmware binary. The size of each configuration table is 1 KB for notebook and power adapter applications. The size of each configuration table is 2KB for dongle application to provide extra space for billboard related parameters.

The CCG firmware area is used for the main CCG firmware application. In all applications, two copies of firmware (called FW1 and FW2) are used.

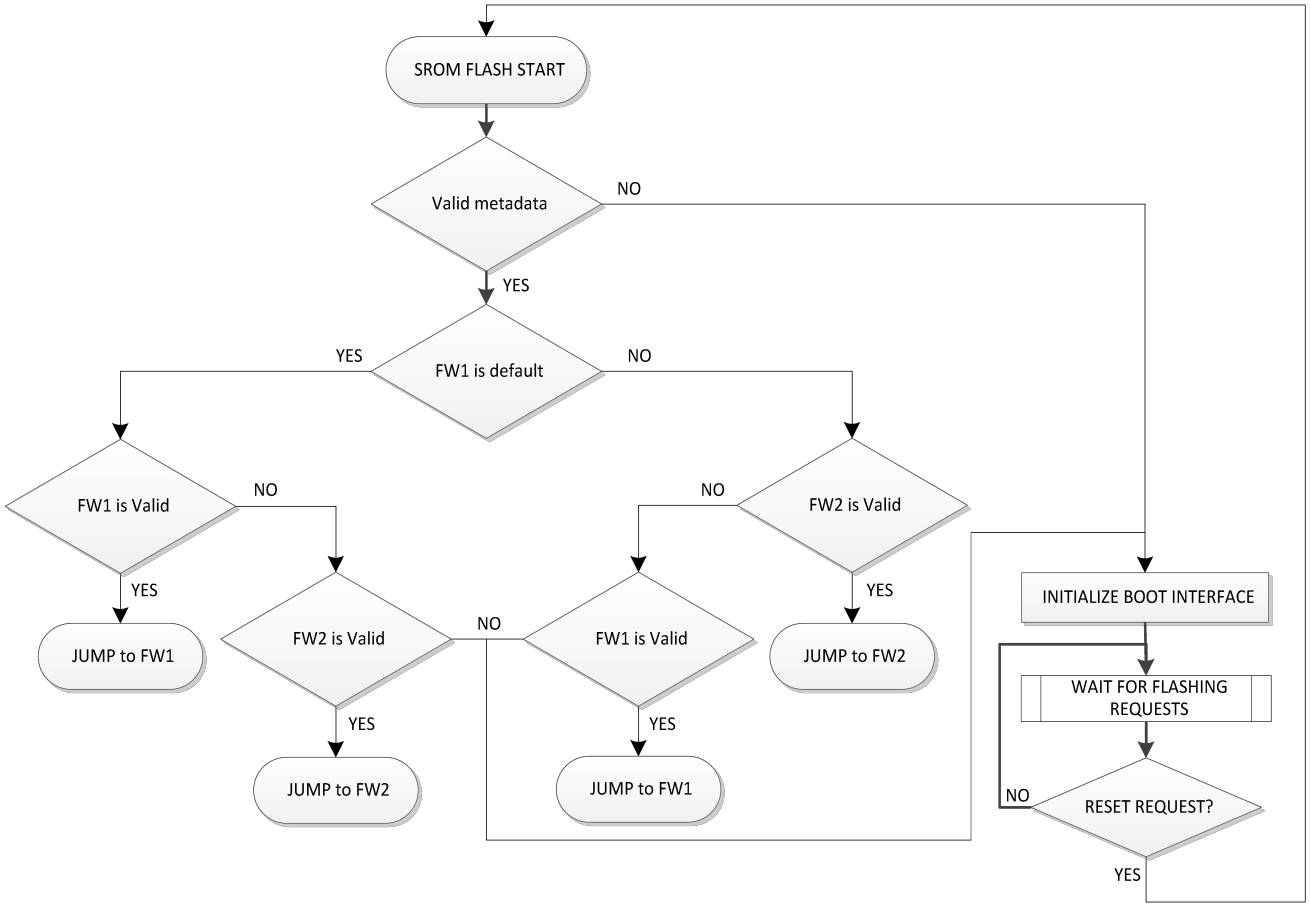
In case of CCG4, FW1 uses the space from 5 KB to 64 KB, and FW2 uses the remaining space. CCG3 FW1 uses the space from 6 KB to 64 KB and FW2 uses the remaining space.

The metadata area holds metadata about the firmware binaries. The firmware metadata follows the definition provided by the PSoC Creator bootloader component; and includes firmware checksum, size, and start address.

5.5 Bootloader

The Flash-based bootloader mainly functions as a boot-strap and is the starting point for firmware execution. It validates the firmware based on checksum stored in Flash. The boot-strap also includes the flashing module in notebook and dongle applications. The bootloader flow diagram follows.

Figure 29: Bootloader Flow Diagram



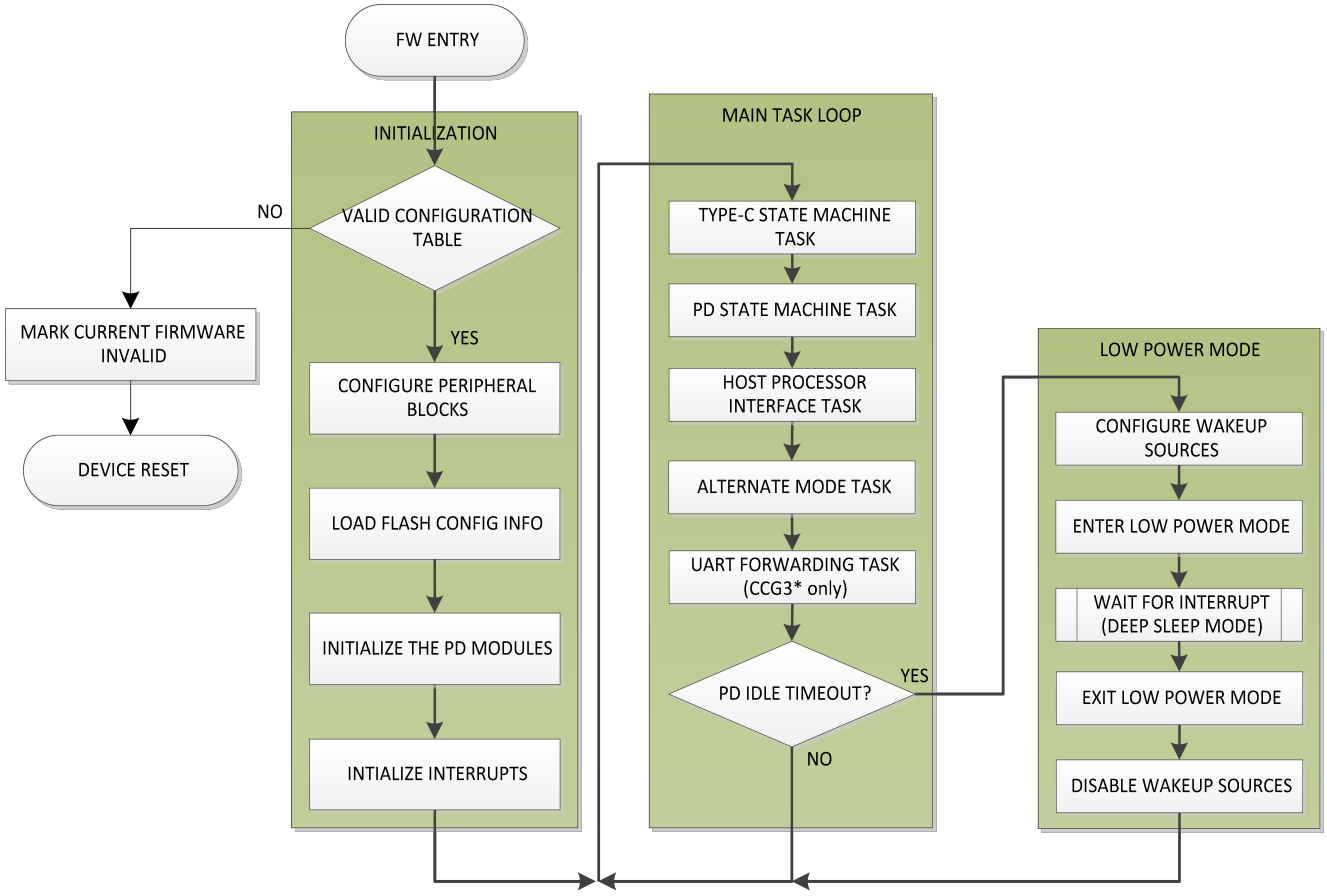
Since CCGx uses redundant firmware images that can update each other, it is expected that the device always has at least one functional image that can be booted by the bootloader. The programming through bootloader capability in notebook and dongle applications will only be used when using a CCGx part programmed with only the bootloader at factory.

As described in Chapter 3, the bootloader keeps track of the last updated firmware image through the metadata; and loads it during start-up.

5.6 Firmware Operation

Figure 30 shows the firmware initialization and operation sequence. The notebook firmware is implemented in the form of a set of state machines and tasks that need to be performed periodically.

Figure 30: Notebook Firmware Flow Diagram



The code flow for the application is implemented in the ***common\main.c*** file. As can be seen from the main () function, the implementation is a simple round-robin loop, which services each of the tasks that the application has to perform.

All of the PD management, HPI command handling, and VDM handling is encapsulated in the task handlers in the CCGx Firmware Stack. Refer to the CCGx FW API Guide document for more details of these functions and handlers.

6. Firmware APIs

This section provides a summary of the APIs provided by the PD stack and other layers in the CCGx firmware solution. Only the APIs that are expected to be used directly from user code are documented here. Refer to the API Reference Guide for more details on the data structures used and all of the APIs.

6.1 API Summary

6.1.1 Device Policy Manager (DPM) API

These functions are declared in the ***src/pd_common/dpm.h*** header file.

Table 18: List of Device Policy Manager API

Function	Description	Parameters	Return
`dpm_init`	Initialize the Device Policy Manager interface for a given USB-PD port. For a dual-port part, the dpm_init needs to be done separately for each port.	port: Port to be initialized app_cbk: Structure with function pointers that the PD stack can call to handle various events.	Call status.
`dpm_start`	Start the PD state machine on the given USB-PD port.	port: Port to be enabled.	Call status.
`dpm_stop`	Stop the PD state machine on the given USB-PD port.	port: Port to be disabled	Call status.
`dpm_deepsleep`	Check for PD state machine idle state and prepare for deep sleep.	port: Port to be checked.	1 if deepsleep is possible. 0 if PD state machine is busy.
`dpm_sleep`	Check for PD state machine idle state and prepare for sleep.	port: Port to be checked.	1 if sleep is possible. 0 if PD state machine is busy.
`dpm_wakeup`	Update the PD block after device resumes from deep sleep.	port: Port to be re-enabled.	Always returns 1.
`dpm_task`	PD state machine task. This should be called periodically from the main application.	port: Port to be serviced.	Call status.

Function	Description	Parameters	Return
dpm_pd_command	Initiate a PD command such as VDM, DR_SWAP etc.	port: Port on which command is to be initiated. cmd: Command to be initiated. buf_ptr: Command parameters. cmd_cbk: Callback to be called at the end of command.	Call status.
dpm_typec_command	Initiate a Type-C command such as Rp change.	port: Port on which command is to be initiated. cmd: Command to be initiated. cmd_cbk: Callback to be called at the end of command.	Call status.
dpm_get_info	Get the DPM status for the device. This is mainly intended for use within other layers in the Cypress provided firmware modules.	port: Port whose status is to be queried.	Pointer to the DPM status structure.
dpm_update_swap_response	Update the response that CCG will send for various swap commands.	port: Port whose swap handler is to be changed. value: Bitmap in the following format. Bits 1:0 => DR_SWAP response Bits 3:2 => PR_SWAP response Bits 5:4 => VCONN_SWAP response 0 => ACCEPT 1 => REJECT 2 => WAIT 3 => NOT_SUPPORTED	Call status.
dpm_update_src_cap	Update the source capabilities PDO list. The provided values will replace the settings from the configuration table.	port: Port to be updated. count: Number of PDOs in list. Maximum allowed value is 7 pdo: Pointer to array containing PDOs.	Call status.
dpm_update_src_cap_mask	Change the mask that enables specific source PDOs from the list.	port: Port to be updated. mask: New PDO enable mask.	Call status.

Function	Description	Parameters	Return
dpm_update_snk_cap	Update the sink capabilities PDO list. The provided values will replace the settings from the configuration table.	port: Port to be updated. count: Number of PDOs in list. Maximum allowed value is 7 pdo: Pointer to array containing PDOs.	Call status.
dpm_update_snk_cap_mask	Change the mask that enables specific sink PDOs from the list.	port: Port to be updated. mask: New PDO enable mask.	Call status.
dpm_update_snk_max_min	Change the min/max current fields associated with each Sink PDO.	port: Port to be updated. count: Number of PDOs in list. Maximum allowed value is 7 max_min: Pointer to array containing new Min/Max operating current values.	Call status.
dpm_update_port_config	Change the USB-PD configuration: port role, default role etc. The port should have been disabled using dpm_typec_command (DPM_CMD_PORT_DISABLE) before the change is attempted.	port: Port to be updated. role: New port role setting. dflt_role: New default port role for DRP. toggle_en: DRP toggle enable flag try_src_en: Try.SRC enable flag	Call status.
dpm_get_polarity	Get the current polarity of the Type-C connection	port: Port to be queried.	0 if CC1 is connected 1 if CC2 is connected
dpm_typec_deassert_rp_rd	De-assert both Rp and Rd on the specified PD port.	port: Port to be updated. channel: Channel on which terminations are to be disabled.	Call status.
dpm_update_port_status	Update the USB-PD port status that will be returned by CCG as part of a Get_Status response.	port: Port to be updated. input: Present power input status battery: Present battery status.	None
dpm_update_ext_src_cap	Updated the Extended Source Capabilities returned by the CCG device.	port: Port to be updated. buf_p: Pointer to buffer containing the extended source capabilities.	None
dpm_update_frs_enable	Update the Fast-Role Swap feature support in the CCG PD state machines. The change will only take effect after a fresh contract negotiation.	port: Port to be updated. frs_rx_en: Whether FRS receive is to be enabled. frs_tx_en: Whether FRS transmit is to be enabled.	None

6.1.2 Host Processor Interface (HPI) API

This section documents the APIs provided by the HPI firmware module. Refer to the ***src/hpi/hpi.h*** file for details. Contact Cypress (ccg_sw@cypress.com) for access to detailed Host Processor Interface (HPI) documentation.

Table 19: List of Host Processor Interface API

Function	Description	Parameters	Return
`hpi_init`	Initialize the HPI protocol module. The serial block index to be used is specified as parameter. However, arbitrary change to the SCB block will not work as the SCB used in the bootloader binary is fixed.	scb_idx: SCB index to be used.	None
`hpi_send_fw_ready_event`	Send a firmware ready (device out of reset) notification to the EC through the HPI interface.	None	None
`hpi_set_fixed_slave_address`	Configure the HPI slave address to a fixed value. If this API is not called, the HPI slave address is determined based on the state of the I2C_CFG pin. This should be called before hpi_init().	slave_addr: Desired slave address minus the read/write bit.	None
`hpi_set_no_boot_mode`	Force the HPI interface to run in no-boot mode. The device will not support any flash update commands in this case. This should be called before hpi_init().	Enable: Whether to enable no-boot mode.	None
`hpi_task`	HPI task handler. HPI commands are queued in ISR and handled here. This should be called periodically from the main application.	None	None
`hpi_reg_enqueue_event`	Send an event notification to the EC.	section: Specifies the PD port corresponding to the event. status: Type of event to send. length: Length of data associated with the event. data: Buffer containing event data.	true if the event was queued. false if event was dropped due to queue overflow.
`hpi_pd_event_handler`	HPI handler for stack event notifications. The solution layer can call this function on getting event notifications from the PD stack.	port: Port on which event occurred. evt: Type of event. data: Event data provided by stack.	None

Function	Description	Parameters	Return
hpi_update_versions	Update the HPI version registers. See section 5.3	bl_version: Boot-loader version. fw1_version: FW1 version. fw2_version: FW2 version.	None
hpi_set_mode_regs	Update the HPI registers that report device mode.	dev_mode: Device mode register value. mode_reason: Boot mode reason register value	None
hpi_update_fw_locations	Update the firmware locations in HPI registers.	fw1_location: Location of FW1 image. fw2_location: Location of FW2 image.	None
hpi_sleep_allowed	Check if the HPI interface is idle, so that device can go to sleep.	None	true if sleep is allowed. false if HPI is busy.
hpi_sleep	Prepare the HPI interface for device deep sleep.	None	true if sleep preparation is completed. false if HPI is busy.
hpi_get_port_enable	Check whether PD ports are enabled.	None	Port enable bit map.

6.1.3 Application Layer API

Table 20 lists the application layer APIs provided by the CCGx SDK. These function declarations and definitions can be found under the ***src/app*** folder.

Table 20: List of Application Layer APIs

Function	Description	Parameters	Return
app_init	Initializes the application layer for operation.	None	None
app_task	Perform application layer tasks. This includes VDM handling and alternate mode implementation.	port: Port on which the application task is to be performed.	1 if successful. 0 if failure
app_event_handler	Application level handler for PD stack events. This updates the internal application state, and then calls the solution level event handler.	port: Port on which event occurred. evt: Type of event. dat: Event data provided by stack.	None
app_get_status	Get the current application status information.	port: Port to be queried.	Pointer to application status structure.

Function	Description	Parameters	Return
app_sleep	Check whether the application layer is ready for device low power mode.	None	true if application layer is idle. false if application layer is busy.
app_wakeup	This is called after device wakes up from sleep, and can be used to restore any state that was saved as part of sleep entry.	None	None
system_sleep	Top level sleep mode entry function. This should be called from the main loop to allow CCG device to consume minimal power.	None	None
eval_src_cap	Evaluates the source capabilities advertised by the port partner and identifies the optimal power contract setting.	port: PD port on which SRC. CAP has been received. src_cap: Source capabilities that were received. app_resp_handler: Callback function to be called to report decision.	None
eval_rdo	Evaluate a PD request (RDO) received and decide whether to accept/reject.	port: PD port on which request has been received. rdo: Received RDO value. app_resp_handler: Callback function to be called to report decision.	None
psnk_set_voltage	Power sink (consumer) handler for voltage change. Default implementation sets up the OVP voltage level based on the provided voltage.	port: Port to be updated. volt_50mV: Expected VBus voltage in 50 mV units.	None
psnk_set_current	Power sink (consumer) handler for operating current change. The default implementation does not do anything.	port: Port to be updated. cur_10 mA: Expected operating current in 10 mA units.	None
psnk_enable	Enable the power sink path.	port: Port to be updated.	None
psnk_disable	Disable the power sink path.	port: Port to be updated.	None
psrc_set_voltage	Set the desired voltage for the power source (provider) output. This function is expected to make the regulator updates as well as set the OVP thresholds based on the voltage. This can be updated if the voltage selection mechanism should be changed.	port: Port to be updated. volt_50mV: Expected VBus voltage in 50 mV units.	None

Function	Description	Parameters	Return
psrc_set_current	Set the current level for the power source output. The default implementation does not do anything.	port: Port to be updated. cur_10mA: Expected operating current in 10 mA units.	None
psrc_enable	Enable the power source output.	port: Port to be updated.	None
psrc_disable	Disable the power source output.	port: Port to be updated.	None
vconn_enable	Enable the VConn supply. The VConn FET is internal to CCG4 device.	port: Port to be updated.	Call status.
vconn_disable	Disable the VConn supply.	port: Port to be updated.	Call status.
vconn_is_present	Check whether VConn supply is present.	port: Port to be queried.	true if VConn is present false if VConn is absent.
vbus_is_present	Check whether VBus is present within a specific range.	port: Port to be queried. volt: Expected VBus voltage. per: Allowed variance in voltage as percentage of expected voltage.	true if VConn is present false if VConn is absent.
vbus_discharge_on	Enable the VBus discharge path.	port: Port to be updated.	None
vbus_discharge_off	Disable the VBus discharge path.	port: Port to be updated.	None
eval_dr_swap	Evaluate a DR_SWAP request from the port partner.	port: Port to be updated. app_resp_handler: Callback function to be called to report decision.	None
eval_pr_swap	Evaluate a PR_SWAP request from the port partner.	port: Port to be updated. app_resp_handler: Callback function to be called to report decision.	None
eval_vconn_swap	Evaluate a VCONN_SWAP request from the port partner.	port: Port to be updated. app_resp_handler: Callback function to be called to report decision.	None
vdm_data_init	Initialize the VDM handler data structure with data from configuration table.	port: Port to be updated.	None

Function	Description	Parameters	Return
`vdm_update_data`	Update the VDM handler data structure with custom data.	port: Port to be updated. id_vdo_cnt: Number of DOs in Discover ID response. id_vdo_p: Array containing the Discover ID response. svid_vdo_cnt: Number of DOs in Discover SVID response. svid_vdo_p: Array containing the Discover SVID response. mode_resp_len: Total length of all Discover Mode responses. mode_resp_p: Array containing actual Discover Mode responses.	None
`eval_vdm`	Evaluate a received PD VDM and respond to it.	port: Port on which VDM is received. vdm: Received VDM pointer. vdm_resp_handler: Callback to be notified about the VDM response.	None
`app_ovp_enable`	Enable the Over-Voltage Protection function.	Port: Port on which OVP is to be enabled. volt_50mV: Allowed maximum voltage in 50 mV units. pfet: Whether the CCG device is power source. ovp_cb: Callback function to be called when OVP is detected.	None
`app_ovp_disable`	Disable the OVP function on a PD port.	port: Port on which OVP is to be disabled. pfet: Whether CCG is a power source at this time.	None

lists the functions that the PD stack and application layer expect to be implemented at the solution level. These functions must be implemented in the source files within the PSoC Creator project workspace. If the target application does not require one or more of these functions; a stub implementation that does nothing should still be provided.

Table 21: Solution-level Functions

Function	Description	Parameters	Return
`mux_ctrl_init`	Initialize the Type-C switch and corresponding control interface. The Type-C data pins should be isolated from the USB and DisplayPort controller pins at this stage.	port: Port to be updated.	true if successful. false if failure
`mux_ctrl_set_cfg`	Update the Type-C switch to enable/disable the desired USB and DisplayPort connections.	port: Port to be updated. cfg: Desired data connection mode. polarity: Polarity of current Type-C connection. 0 for CC1 and 1 for CC2.	true if successful. false if failure
`sln_pd_event_handler`	This is top level handler for system event notifications provided by the PD stack. The default implementation of this function calls the HPI event handler so that the EC can be notified about these events.	port: Port on which event occurred. evt: Type of event. data: Event data provided by stack.	None
`app_get_callback_ptr`	Function that returns a structure filled with callback function pointers for various system events. Default implementations for all of these functions are provided under the app folder, and the structure can be initialized with the corresponding pointers.	port: Port to be queried.	Pointer to structure containing callback function pointers. This structure should remain valid throughout the device operation.

6.1.4 Alternate Mode API

This section documents the alternate mode related API provided in the CCGx SDK. These APIs are defined in the sources under ***src/app/alt_mode*** and are summarized in Table 22.

Table 22: List of Alternate Mode APIs

Function	Description	Parameters	Return
`enable_vdm_task_mngr`	Enabled the alternate mode manager.	port: Port to be updated.	None
`vdm_task_mngr_deinit`	De-initialize the alternate mode manager.	port: Port to be updated.	None
`is_vdm_task_idle`	Check if the alternate mode manager is idle, so that device can enter low power mode. Each port has to be queried separately.	port: Port to be queried.	true if the manager is idle. false if the manager is busy.
`vdm_task_mngr`	Alt. mode manager state machine task. This is called from app_task.	port: Port to be serviced.	None

Function	Description	Parameters	Return
eval_rec_vdm	Evaluate VDM message received.	port: Port to be updated. vdm_rcv: Pointer to received attention VDM.	true if VDM is to be ACKed. false if VDM is to be NACKed.

6.1.5 Hardware Adaptation Layer (HAL) API

This section documents the API provided as part of the Hardware Adaptation Layer (HAL), which provides drivers for various hardware blocks on the CCG device.

GPIO API

The PSoC Creator GPIO component and associated APIs can be used in all CCG projects. However, the SDK also provides a set of special API for reduced memory footprint. These APIs are defined in the ***src/system/gpio.c*** file and are summarized in Table 23.

Table 23: List of GPIO APIs

Function	Description	Parameters	Return
hsiom_set_config	Update the IO matrix configuration for a given pin.	port_pin: CCG pin identifier. hsiom_mode: Desired IO configuration	None
gpio_set_drv_mode	Select GPIO drive mode for a given pin.	port_pin: CCG pin identifier. drv_mode: Desired drive mode	None
gpio_hsiom_set_config	Update IO matrix and drive mode for a given pin.	port_pin: CCG pin identifier. hsiom_mode: Desired IO configuration drv_mode: Desired drive mode value: Desired output state	None
gpio_int_set_config	Configure interrupt associated with a given pin.	port_pin: CCG pin identifier. int_mode: Desired interrupt configuration.	None
gpio_set_value	Update the output value of a given pin. The IO configuration and drive mode for the pin should have been set separately.	port_pin: CCG pin identifier. value: Desired output state	None
gpio_read_value	Get the current state of a given pin.	port_pin: CCG pin identifier.	true if the pin is high. false if the pin is low.
gpio_get_intr	Check if there is an active interrupt associated with the given pin.	port_pin: CCG pin identifier.	true if interrupt is active. false if interrupt is not active.

Function	Description	Parameters	Return
gpio_clear_intr	Clear any interrupts associated with the given pin.	port_pin: CCG pin identifier.	None

I2C API

The Serial Communication Block component in PSoC Creator can be used with the CCG device. However, the SDK provides a dedicated I²C slave mode driver, which is optimized for HPI implementation. These API definitions are provided in ***src/scb/i2c.c*** and are summarized in Table 24.

Table 24: List of I2C driver APIs

Function	Description	Parameters	Return
i2c_scb_init	Initialize the I2C slave block and set driver parameters.	scb_index: SCB index to be used. mode: Mode of I2C block operation. clock_freq: Expected bit rate on the interface. slave_addr: Slave address to be used. slave_mask: Mask to be applied on the slave address to detect I2C addressing. cb_fun_ptr: Callback function to be called for read/write/error notifications. scratch_buffer: Pointer to scratch buffer to be used to received incoming data. scratch_buffer_size: Size of scratch buffer.	None
i2c_scb_write	Write data into the I2C block transmit FIFO.	scb_index: SCB index to be used. source_ptr: Location of data to be written. size: Size of data to be written. Should be 8 bytes or lesser. count: Return parameter indicating actual size of data written.	None
i2c_reset	Reset the I2C block.	scb_index: SCB index to be used.	None
i2c_slave_ack_ctrl	Used to enable/disable clock stretching in the device address stage.	scb_index: SCB index to be used. enable: Enable device address acknowledgement.	None
i2c_scb_is_idle	Check whether the I2C module is idle.	scb_index: SCB index to be used.	true if the block is idle. false if the block is busy.
i2c_scb_enable_wakeup	Enable I2C device addressing as a wakeup source from low power mode.	scb_index: SCB index to be used.	None

Flash API

The flash API provides the core functionality used for CCG configuration and firmware updates. These are wrappers over the PSoC Creator provided flash APIs, and implement checks to ensure that a firmware binary is not corrupted by writing while it is being accessed. The flash related API are defined in ***src/system/flash.c*** and are summarized in Table 25.

Table 25: List of flash API

Function	Description	Parameters	Return
`flash_enter_mode`	Enable flash updates through the specified interface. Flash updates are only allowed through one interface (I2C, CC etc.) at a time.	is_enable: Whether to enable or disable flash access. mode: Flash access interface	None
`flash_access_enabled`	Check whether flash access through any of the specified interfaces is enabled.	modes: Bitmap representing the flash interfaces to be checked.	true if flash access is enabled. false otherwise.
`flash_set_access_limits`	Set limits regarding the flash rows that can be accessed. The firmware application should identify the memory range that it is using, and use this API to protect it from updates.	start_row: Lowest flash row that can be accessed. last_row: Highest flash row that can be accessed. md_row: Metadata row that can be accessed. bl_last_row: Last row used by boot-loader. Any flash row above this value can be read.	None
`flash_row_clear`	Clear the contents of the specified flash row.	row_num: Row to be cleared.	Flash erase status
`flash_row_write`	Write the desired content into a flash row. This is a blocking operation.	row_num: Flash row to be updated. data: Buffer containing flash data. cbk: Must be zero as non-blocking writes are not supported by the device.	Flash write status
`flash_row_read`	Read the content of a flash row into the provided buffer.	row_num: Flash row to be read. data: Buffer to read the data into.	Flash read status.

Timer API

The CCG firmware stack uses a soft timer implementation for various timing measurements. The soft timer granularity is 1 ms, and it uses a single hardware timer. If the timer block used is WDT (WatchDog Timer), the timers can be used across device sleep modes; and it is possible to use a tickless implementation which reduces interrupt frequency. The soft timer related API are defined in ***src/system/timer.c*** and are summarized in Table 26.

Table 26: List of timer API

Function	Description	Parameters	Return
`timer_init`	Initialize the timer module. This enables the hardware timer and interrupt as well.	None	None

Function	Description	Parameters	Return
timer_start	Start one soft timer (one shot).	instance: Soft timer group or instance. Separate timer groups are maintained for each PD port. id: ID of timer to be started. Timer IDs are assigned statically. period: Timer period in milliseconds. cb: Timer expiry callback.	true if successful. false if failure.
timer_stop	Stop a running soft timer.	instance: Soft timer group or instance. id: ID of timer to be stopped.	None
timer_is_running	Check whether a soft timer is running.	instance: Soft timer group or instance. id: ID of timer to be queried.	true if running. false if not running.
timer_stop_all	Stop all soft timers in a group.	instance: Timer group to be stopped.	None
timer_stop_range	Stop all soft timers whose IDs fall in a range.	instance: Timer group to be stopped. start: Lowest timer ID to be stopped. stop: Highest timer ID to be stopped.	None
timer_num_active	Get number of active timers in a group.	instance: Soft timer group or instance.	Count of active timers.
timer_enter_sleep	Prepare the timer module for sleep entry.	None	None

6.1.6 Firmware Update API

CCG application support firmware updates through interfaces like HPI (I2C) and CC (Unstructured VDMs). The firmware update APIs are common functions that are used by each of these protocol modules to implement the firmware update functionality.

The firmware update related API are defined in ***src/system/boot.c*** and are summarized in Table 27.

Table 27: Firmware Update API

Function	Description	Parameters	Return
boot_validate_fw	Validate the firmware image in device flash.	fw_metadata: Pointer to metadata regarding the firmware image.	Valid/invalid status.
boot_validate_configtable	Validate the configuration table in device flash.	table_p: Pointer to the configuration table.	Valid/invalid status.
get_boot_mode_reason	Compute the boot mode reason register value by validating firmware and configuration tables.	None	Boot mode reason bitmap value.
boot_get_boot_seq	Get the flash sequence number associated with a given firmware binary.	fwid: ID of firmware binary to be queried.	Flash sequence number value.
sys_set_device_mode	Set the current firmware mode for the CCG device.	fw_mode: Firmware mode to be set.	None
sys_get_device_mode	Get the current firmware mode for the CCG device.	None	Current firmware mode.

6.2 API Usage Examples

This section provides a few examples for the usage of the APIs documented under Section 6.1Refer to the API reference guide for more details.

Most of the PD operations are initiated using the `dpm_pd_command()` and `dpm_typec_command()` APIs. These APIs are non-blocking, and only initiate the operation. A callback function can be passed to the API; and it will be called on completion of the operation. Completion of these operations will require the tasks in the main loop to be executed, and therefore, the caller cannot block waiting for the callback to arrive.

If there is a need to wait for the operation to complete and then initiate other operations, this can be done in two ways:

6. Initiate the follow-on operations from the callback function itself.
7. Modify the main loop to detect the callback arrival, and then initiate the next operation after this.

6.2.1 Boot API Usage

The bootloader and firmware application communication in the CCGx SDK is built using the PSoC Creator Bootloader and Bootloadable components. This section shows how the PSoC Creator bootloader and bootloadable components along with the wrapper APIs in the SDK to transfer control from the application firmware to the bootloader or to the application in the alternate memory bank.

Perform Device Reset

Since the order in which the bootloader prioritizes firmware images is fixed, resetting the device causes the device to boot back into the same mode that it previously was in. The `CySoftwareReset()` API can be used to initiate a CCG device reset.

```
/* Include relevant header files. */
#include <project.h>

void reset_ccgx_device (void)
{
       /* Initiate device reset. */
       CySoftwareReset ();
}
```

Jump to Bootloader

This operation is not required because firmware update and flash read functionality is provided by the application firmware itself. Also, the bootloader is a fixed binary application which cannot be updated to include additional functionality.

However, you can use the `Bootloadable` component API to transfer control to the bootloader from the application firmware. You can do this by specifying the boot type for the next run using the `Bootloadable_SET_RUN_TYPE()` macro and then initiating a reset using `CySoftwareReset()`.

```
/* Include relevant header files. */
#include <project.h>
#include <boot.h>

void jump_to_bootloader(void)
{
       /* Select the boot mode for the next run. */
       Bootloadable_SET_RUN_TYPE(CCG_BOOT_MODE_RQT_SIG);
       /* Initiate device reset. */
       CySoftwareReset();
}
```

Jump to Alternate Firmware

As described in Chapter 5, the CCGx firmware projects are set up such that they generate two copies of the application firmware. While both of these copies are expected to be equivalent, there may be cases where you need to use a fixed backup firmware along with a dynamically updated primary firmware. In such cases, it would be desirable to transfer control to the backup firmware in order to update the primary firmware.

The APIs shown in Section can also be used to transfer control to the alternate firmware binary. You will need to determine the identity of the current firmware binary (FW1 or FW2) and then initiate the switch accordingly.

```
/* Include relevant header files. */
#include <project.h>
#include <boot.h>

void jump_to_alternate_fw(void)
{
	/* Set the next boot mode based on the current FW ID. */
	if (sys_get_device_mode() ==  SYS_FW_MODE_FWIMAGE_1)
	{
		Bootloadable_SET_RUN_TYPE(CCG_FW2_BOOT_RQT_SIG);
	}
	else
	{
		Bootloadable_SET_RUN_TYPE(CCG_FW1_BOOT_RQT_SIG);
	}

	/* Initiate device reset. */
	CySoftwareReset();
}
```

GPIO API Usage

All of the APIs provided by the PSoC Creator Pins component can be used in CCGx firmware solutions. In addition to these, specific APIs to perform common GPIO functions are provided in the CCGx SDK. The list of GPIO APIs is provided in Section .

Configuring a CCGx Pin as an Edge Triggered Interrupt Input

The `gpio_hsiom_set_config()` API is used to set the I/O mapping and drive mode settings for a given CCGx pin. The `gpio_int_set_config()` API is used to enable interrupt functionality on a CCGx pin. The following code snippet shows how the pin P3[1] on CCGx can be configured as an input signal triggering interrupts on a falling edge.

```
/* We are using P3.1 as the interrupt pin. */
#define INTR_GPIO_PORT_PIN       (GPIO_PORT_3_PIN_1)

/* The ISR vector number corresponds to PORT3. */
#define CCGX_PORT3_INTR_NO       (3u)

/* ISR for the GPIO interrupt. */
CY_ISR(gpio_isr)
{
	/* Clear the interrupt. */
	gpio_clear_intr(INTR_GPIO_PORT_PIN);

	/* Custom interrupt handling actions here. */
	...;
```

```
}

/* Function to configure and enable the interrupt. */
void configure_intr_input(void)
{
	/* Configure the IO modes for the pin. */
	gpio_hsiom_set_config(INTR_GPIO_PORT_PIN,
		HSIOM_MODE_GPIO, GPIO_DM_HIZ_DIGITAL, false);

	/* Configure the interrupt mode for the pin. */
	gpio_int_set_config(INTR_GPIO_PORT_PIN,
		GPIO_INTR_FALLING);

	/* Set the ISR routine and enable the interrupt. */
	CyIntSetVector(CCGX_PORT3_INTR_NO, gpio_isr);
	CyIntEnable(CCGX_PORT3_INTR_NO);
}
```

Connecting a pin to the internal ADC

Refer to the CCGx device datasheet to identify pins that can be connected to the internal ADC blocks through the Analog MUX configuration. The `hsiom_set_config()` API can be used to connect a specific pin to the ADC.

```
#define VBUS_MON_PORT_PIN  (GPIO_PORT_3_PIN_1)

void connect_vbus_mon_to_adc (void)
{
	/* Connect the pin to AMUXB. */
	hsiom_set_config (VBUS_MON_PORT_PIN, HSIOM_MODE_AMUXB);
}
```

6.2.2 Timer API Usage

The CCGx SDK provides a soft timer module, which can be used for task scheduling. The timer APIs allow users to create one-shot timer objects with callback notification on timer expiry.

Soft timers are identified using a single byte timer ID, and the caller should ensure that the timer ID used does not collide with timers used elsewhere. This is facilitated by reserving the timer ID range from 0xE0 to 0xFF for use by user application code. These timer IDs are not used internally within the CCGx firmware stack and are safe for use.

A soft timer is started using the `timer_start()` API and can be aborted using the `timer_stop()` API.

```
#define APP_TIMER_ID              (0xF0)

static void timer_expiry_callback(uint8_t instance, timer_id_t id)
{
	/* Start the desired task here. */
	...;
}

/* Use a timer to schedule task to be run delay_ms milliseconds later. */
void schedule_task(uint16_t delay_ms)
{
	/* Start an application timer to wait for delay_ms. */
	/* Devices with two USB-PD ports support two sets of timers, and
	   the set to be used is selected using the first parameter. */
	timer_start(0, APP_TIMER_ID, delay_ms, timer_expiry_callback);
}
```

6.2.3 HPD API Usage

The ***HotPlugDetect*** output pin from CCGx is used in Notebook implementations to signal interrupts from the far-end DisplayPort (DP) peripheral to the DP controller in the Notebook system. The DP peripheral will signal HPD events to the CCGx Notebook controller through PD messages, and CCGx will relay these HPD events to the DP controller through the GPIO output.

The `hpd_transmit_init()` and `hpd_transmit_sendevt()` APIs are used to initialize the HPD transmit logic and to send event notifications to the DP controller respectively.

```
/* Callback that notifies user of completion of HPD signaling. */
static void hpd_callback(uint8_t port, hpd_event_type_t event)
{
	if (event == HPD_COMMAND_DONE)
	{
		/* Requested HPD command is complete. */
	}
}

/* Initialize the HPD transmit logic for USB-PD port 0, and register
   the command completion callback. */
void initialize_hpd_logic(void)
{
	hpd_transmit_init(0, hpd_callback);
}

/* Send HPD_IRQ to the DP controller. */
void send_hpd_irq(void)
{
	/* Asynchronous mode: Do not wait for completion. */
	hpd_transmit_sendevt(0, HPD_EVENT_IRQ, false);
}
```

6.2.4 Sleep Mode Control

The decision to enter device deep sleep mode to save power is made at the application level. The `system_sleep()` function call in the main loop can be disabled if deep sleep mode entry is to be disabled.

6.2.5 DPM API Usage

Enabling a PD port

The `dpm_start()` API can be used to enable a PD port for operation. The `dpm_init()` API should have been called prior to doing this.

```
bool enable_pd_port(uint8_t port)
{
	if (dpm_start(port) == 0)
	{
		/* DPM start failed. Handle errors. */
		return (false);
	}
	return (true);
}
```

Disabling a PD port

The `dpm_stop()` API should not be used to directly disable a PD port, as the port might already be in contract. The `dpm_typec_command()` API should be used initiate the `DPM_CMD_PORT_DISABLE` command. This will ensure that the port is disabled safely and the VBus voltage is discharged to a safe level, before the completion callback is issued.

```
static volatile bool pd_disable_completed = true;
static volatile bool pd_disable_issued    = false;

/* Callback for the PD disable command. */
static void pd_port_disable_cb(uint8_t port, dpm_typec_cmd_resp_t resp)
{
       pd_disable_completed = true;
       /* Other APIs can be started here, if required. */
}

bool disable_pd_port(uint8_t port)
{
       /* Store state of operation. */
       pd_disable_issued    = true;
       pd_disable_completed = false;

       /* Initiate port disable. */
       if (dpm_typec_command(port, DPM_CMD_PORT_DISABLE,
                     pd_port_disable_cb) != CCG_STAT_SUCCESS)
       {
              /* Handle error here. */
              pd_disable_issued = false;
              return false;
       }

       /* Port disable has been queued. We cannot block for callback.
          Wait for callback in the main loop.
        */
       return true;
}

int main ()
{
       /* Init tasks here. */
       ...;

       while (1)
       {
              /* Call regular task handlers (DPM, APP, HPI) here. */
              ...;

              if ((pd_disable_issued) && (pd_disable_completed))
              {
                     /* Port is now disabled. */
                     ...;

                     pd_disable_issued = false;
              }
       }
}
```

Sending a DISCOVER_ID VDM

The `dpm_pd_command()` API should be used to send VDMs and other PD commands to the port partner. The send operation is non-blocking and the completion callback will notify that the operation is complete. Note that the main loop should continue to run for proper completion of the VDM operation.

```
static volatile bool    abort_cmd = false;
static dpm_pd_cmd_buf_t cmd_buf;

static void pd_command_cb(uint8_t port, resp_status_t resp,
        const pd_packet_t *vdm_ptr)
{
       uint32_t response;

       if (status == RES_RCVD)
       {
              /* Response received. Check handshake. */
              response = vdm_ptr->dat[0].std_vdm_hdr.cmd_type;
              switch (response)
              {
                     case CMD_TYPE_RESP_ACK:
                            /* ACK received. */
                            ...;
                            break;
                     case CMD_TYPE_RESP_BUSY:
                            /* BUSY received. */
                            ...;
                            break;
                     case CMD_TYPE_RESP_NAK:
                            /* NACK received. */
                            ...;
                            break;
              }
              /* Next operation can be started from here. */
       }
}

bool send_discover_id(uint8_t port)
{
       /* Store state of operation. */
       pd_command_issued    = true;
       pd_command_completed = false;

       /* Format the command parameters.
          Single DO with standard Discover_ID command to SOP controller.
          Timeout is set to 100 ms.
        */
       cmd_buf.cmd_sop      = SOP;
       cmd_buf.cmd_do[0]    = 0xFF008001;
       cmd_buf.no_of_cmd_do = 1;
       cmd_buf.timeout      = 100;

       /* Initiate the command. Keep trying until accepted. */
       while (dpm_pd_command(port, DPM_CMD_SEND_VDM,
```

```
        &cmd_buf, pd_command_cb) != CCG_STAT_SUCCESS)
    {
        /* Can implement a timeout/abort here. */
        if (abort_cmd)
            return false;
    }
    /* Command has been queued. We cannot block for callback here. */
    return true;
}
```

Getting Current PD Port Status

The Device Policy Manager interface layer in the CCGx PD stack maintains a status data structure that provides complete status information about the USB-PD port.

This structure can be retrieved using the `dpm_get_info()` API. The API returns a `const` pointer to a `dpm_status_t` structure which includes the following status fields:

1. `attach`: Specifies whether the port is currently attached.
2. `cur_port_role`: Specifies whether the port is currently a Source or a Sink
3. `cur_port_type`: Specifies whether the port is currently a DFP or an UFP
4. `polarity`: Specifies the Type-C connection polarity (CC1 or CC2 being used)
5. `contract_exist`: Specifies whether a PD contract exists
6. `contract`: Specifies the current PD contract (voltage and current) information.
7. `emca_present`: Specifies whether CCGx as DFP has detected a cable marker
8. `src_sel_pdo`: Specifies the PDO that CCGx as source used to establish contract
9. `snk_sel_pdo`: Specifies the Source Cap that CCGx as sink accepted to establish contract
10. `src_rdo`: Specifies the RDO that CCGx received for PD contract
11. `snk_rdo`: Specifies the RDO that CCGx as Sink sent for PD contract.

Issue a DR_SWAP where required

CCGx in a Notebook Port controller application is expected to function as a DFP. You can check the current port type of the CCGx device and initiate a DR_SWAP if CCGx is a UFP, so that we can ensure that the supported alternate modes can be enabled.

The current port type is checked as described in Section , and the dpm_pd_command() API can be used to initiate a DR_SWAP.

```
/* Function to initiate a DR_SWAP if CCGx is UFP. */
void dr_swap_if_required()
{
    const dpm_status_t *dpm_stat = dpm_get_info (0);
    dpm_pd_cmd_buf_t param;
    ccg_status_t status;

    if (dpm_stat->cur_port_type == PRT_TYPE_UFP)
    {
        /* CCGx is UFP. Initiate DR_SWAP.
           We keep retrying while the PD port is in a busy state. */
        param.cmd_sop = SOP;
        do {
            status = dpm_pd_command (0, DPM_CMD_SEND_DR_SWAP,
                &param, cmd_callback);
```

```
        } while (status == CCG_STAT_BUSY);

    }
}
```

Change the Source Capabilities

The `dpm_update_src_cap()` and `dpm_update_src_cap_mask()` APIs can be used to update the source capabilities supported by CCGx.

At any time, CCGx can support a set of maximum seven source capabilities. These seven capabilities are maintained in the form of a the `cur_src_pdo` array in the `dpm_status_t` structure. A subset of these PDOs can be enabled at runtime using a PDO enable bit mask setting. The current PDO enable mask value can be read from the `src_pdo_mask` field of the `dpm_status_t` structure.

The PDO enable mask can be changed using the `dpm_update_src_cap_mask()` API.

The set of PDOs can be changed using the `dpm_update_src_cap()` API. The PDO enable mask will also need to be updated after updating the set of PDOs.

```
/* Function to configure and enable a desired source PDO. */
void select_source_pdo(pd_do_t new_pdo)
{
    const dpm_status_t *dpm_stat = dpm_get_info (0);
    uint8_t index;
    bool pdo_found = false;

    /* See if the new_pdo is already part of the list. */
    for (index = 0; index < dpm_stat->src_pdo_count; index++)
    {
        if (dpm_stat->src_pdo[index].val == new_pdo.val)
        {
            pdo_found = true;
            break;
        }
    }

    if (pdo_found)
    {
        /* PDO found. Just enable it. */
        dpm_update_src_cap_mask(0,
            (dpm_stat->src_pdo_mask | (1 << index)));
    }
    else
    {
        /* PDO not found, update the PDO list and enable it.
           Note: For this example, we are replacing the complete list
           with a single PDO. This needs be updated to retain the
           other required PDOs.
        */
        dpm_update_src_cap(0, 1, &new_pdo);
        dpm_update_src_cap_mask(0, 1);
    }
}
```

Refer to the Alternate Mode module source for more examples of using the DPM APIs.

6.2.6 Solution Level Examples

PD Event Handling

The PD events raised by the stack are handled at the solution level in the `sln_pd_event_handler()` function. In the normal case where policy decisions are handled through the EC, it is sufficient to pass the events onto the EC through the HPI interface. See below for a sample implementation of the event handler.

```
/* Solution PD event handler */
void sln_pd_event_handler(uint8_t port, app_evt_t evt, const void *data)
{
    /* Pass the event onto the EC through HPI. */
    hpi_pd_event_handler(port, evt, data);
}
```

Application Callback Registration

The application callbacks that handle various operations requested by the PD stack are registered through a structure that contains pointers to all the functions. The callbacks are registered using the `app_get_callback_ptr()` function. A sample implementation of this function is shown below.

```
/*
 * Application callback functions for the DPM. Since this application
 * uses the functions provided by the stack, loading with the stack defaults.
 */
const app_cbk_t app_callback =
{
    app_event_handler,     /* Event handler. */
    psrc_set_voltage,      /* Source voltage update function. */
    psrc_set_current,      /* Source current update function. */
    psrc_enable,           /* Enable source FET. */
    psrc_disable,          /* Disable source FET. */
    vconn_enable,          /* Enable VConn supply. */
    vconn_disable,         /* Disable VConn supply. */
    vconn_is_present,      /* Check if VConn is present. */
    vbus_is_present,       /* Check if VBus is in the expected range. */
    vbus_discharge_on,     /* Enable VBus discharge path. */
    vbus_discharge_off,    /* Disable VBus discharge path. */
    psnk_set_voltage,      /* Set sink voltage. */
    psnk_set_current,      /* Set sink current. */
    psnk_enable,           /* Enable sink FET. */
    psnk_disable,          /* Disable sink FET. */
    eval_src_cap,          /* Evaluate source power capabilities. */
    eval_rdo,              /* Evaluate partner power request. */
    eval_dr_swap,          /* Evaluate DR_SWAP command. */
    eval_pr_swap,          /* Evaluate PR_SWAP command. */
    eval_vconn_swap,       /* Evaluate VCONN_SWAP command. */
    eval_vdm               /* Evaluate received VDM. */
};
app_cbk_t* app_get_callback_ptr(uint8_t port)
{
    /* Solution callback pointer is same for all ports */
    (void)port;
    return ((app_cbk_t *)(&app_callback));
}
```

Change the Source PDO selection logic

The `eval_src_cap()` callback function is invoked by the PD stack on receiving source capabilities message from the Source. The default implementation of the same is available in ***src/app/pdo.c***. This function can be overridden during application callback registration (Section).

The `eval_src_cap() and is_src_acceptable_snk()` functions in ***src/app/pdo.c*** can be used as template and a custom function can be implemented in the solution.

For example, If an additional check needs to be done for maximum current support in the source PDO, then this can be done by changing the `eval_src_cap()` callback function to `my_eval_src_cap()`.

```
/* Custom function to check if the source PDO is acceptable or not. */
void my_is_src_acceptable_snk(uint8_t port, pd_do_t* pdo_src, uint8_t snk_pdo_idx)
{
    ...
    case PDO_FIXED_SUPPLY:
        if(fix_volt == pdo_snk->fixed_snk.voltage)
        {
            compare_temp = GET_MAX (max_min_temp, pdo_snk->fixed_snk.op_current);
            if (pdo_src->fixed_src.max_current >= compare_temp)
            {
                /* Added new check for absolute maximum current. */
                if (pdo_src->fixed_src.max_current <= MY_MAX_SNK_CURRENT)
                {
                    op_cur_power[port] = pdo_snk->fixed_snk.op_current;
                    out = true;
                }
            }
        }
        break;
    ...
}


/* Function to evaluate source PDO message. */
void my_eval_src_cap(uint8_t port, const pd_packet_t* src_cap, app_resp_cbk_t
      app_resp_handler)
{
    ...
    for(snk_pdo_index = 0u; snk_pdo_index < dpm->cur_snk_pdo_count;
             snk_pdo_index++)
   {
      for(src_pdo_index = 0u; src_pdo_index < num_src_pdo; src_pdo_index++)
      {
          if(my_is_src_acceptable_snk(port, (pd_do_t*)(&src_cap-
      >dat[src_pdo_index]),
                    snk_pdo_index))
          {
              ...
          }
          ...
      }
   }
}
```

Refer to the notebook project source files (main.c) for more examples of the solution level code.

6.3 Alternate Mode Handling

Support for USB-PD alternate modes is a critical part of the CCG firmware functionality. Support for the DisplayPort alternate mode is pre-built into the Notebook and dongle applications. Users can add additional alternate mode support to the firmware. The procedure to add additional alternate mode handler to the firmware includes two steps:

1. Implementing the handlers for the alternate modes. This includes code that will discover UFP capabilities and handle attention messages in a DFP role, and/or code that will receive and handle alternate mode requests as an UFP.
2. Registering the alternate mode handlers with the manager.

6.3.1 Implementing the Alternate Mode Handlers

The CCG firmware stack provides a generic alternate mode manager which holds information about the supported alternate modes. This manager will invoke the handler functions specific to the alternate modes registered in the firmware application.

When CCG is a DFP, the alternate mode manager will discover whether the connected UFP supports any alternate modes for which handlers have been registered; and then call the associated handler functions.

When CCG is a UFP, the alternate mode manager will check whether incoming VDMs correspond to any registered alternate modes; and then call the associated handler functions.

Alternate Mode Data Structure

The alt_mode_info_t structure serves as the interface between the alternate mode manager and the handlers for each alternate mode. An array of such structures is maintained by the alternate mode manager. This structure incorporates the following fields:

Table 28: List of alternate mode information structure members

Field	Type	Description
mode_state	enum alt_mode_state_t	State of the alternate mode. Can be one of DISABLED, IDLE, INIT, SEND_CMD, WAIT_FOR_RESP, FAIL or EXIT.
sop_state[]	Array of enum alt_mode_state_t	Alternate mode state corresponding to each PD packet type (SOP, SOP' and SOP''). This is useful in cases where the alternate mode requires any EMCA cables to support the mode as well.
vdo_max_numb	Unsigned char	Maximum number of VDOs that the alternate mode handler can handler.
`alt_mode_id`	Unsigned char	The alternate mode id for this mode.
`vdm_header`	pd_do_t	Holds the VDM header for the received messages.
`vdo`	Array of pd_do_t pointers	Pointers to buffers where alt. mode VDOs with various packet types should be stored. The storage is provided by the alternate mode handler and used by the manager.
`cbk`	Function pointer	Callback used by alternate mode manager to invoke the specific alternate mode handler.

Alternate Mode Handling Flow

The alternate mode handler uses the mode state to communicate (receive/send VDM) with alternate mode manager.

- IDLE state is responsible for received VDM processing;
- WAIT_FOR_RESP state is responsible for received VDM response processing;
- FAIL state is responsible for the failed VDM processing;
- INIT state is responsible for initialization of the alt mode (DFP only);
- EXIT state is responsible for de-initialization/exit of the alt mode (DFP only);
- SEND state should be set by alt mode to inform alt modes manager that a VDM packet is ready and should be sent to the port partner.

The alternate mode handler can use internal states to process the received VDM or VDM responses. These states are not used by the alternate mode manager.

6.3.1.1 DFP Handling

When alternate mode is DFP when main DFP state machine operates with five states:

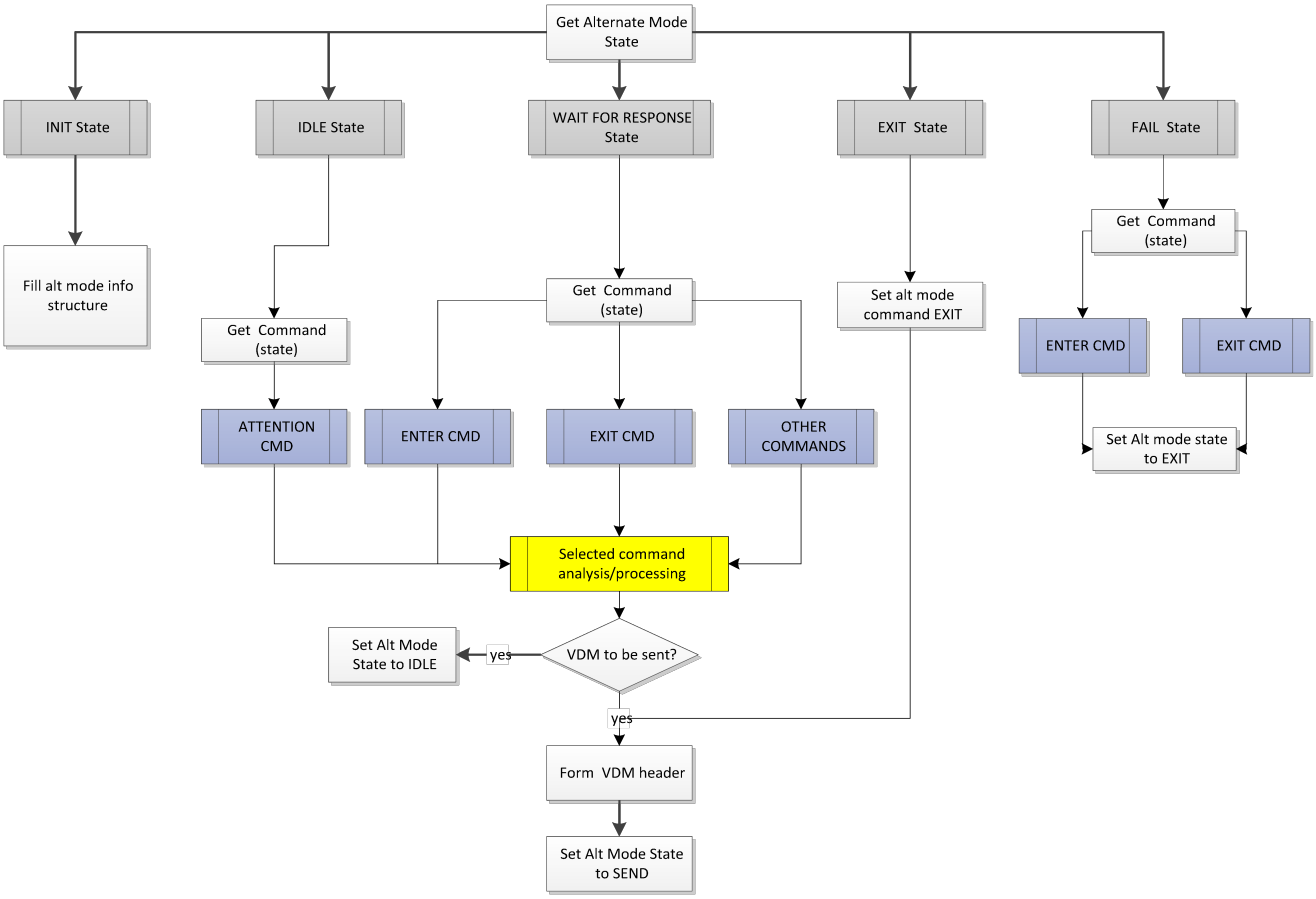
1. The INIT state is used to initiate alternate mode handling. This state uses in two cases:
    a. When DFP alternate mode registrations is successful and we need to initiate alternate mode handling.
    b. When Alt modes manager initiates asynchronous entry of the alternate mode.
    c. During initialization next steps should be done:
        i. Set sop_state array variables as ALT_MODE_STATE_SEND in the dependence if SOP/SOP'/SOP" VDMs should be send while entering the mode
        ii. Save pointers to the VDO buffer in the info structure
        iii. Assign enter mode VDOs if needed
        iv. Save alt mode DFP state machine function in the info structure
        v. Save App command handler function in the info structure
        vi. Set alternate mode state as enter
        vii. Additional initialization code as required by the alternate mode.
2. IDLE state is used to analyze received VDM related to the alt mode. When a VDM is received, the following steps should be completed:
    a. Get the received command from VDM header
    b. Run custom analysis of the received VDM
    c. If a response VDM should be sent after analysis, then change the internal alternate mode state in compliance with the specific command number, fill the VDO buffer and set alternate mode state to the SEND state.
3. WAIT_FOR_RESP state is used to process/analyze received VDM response. When VDM response is received, next steps should be done:
    a. Get internal alternate mode depending on command from VDM header
    b. Set alternate mode state to IDLE
    c. Analyze the received VDM response

d. If another VDM should be sent based on the received response, then change the internal alternate mode state in compliance with the specific command number, fill the VDO buffer and set alternate mode state to the SEND state.

4. FAIL state is used to make decision when sent VDM was failed (e.g. NAKed response, Good CRC was not received etc.)

5. EXIT state is used when the alternate mode manager initiates asynchronous exit of the alternate mode. In this case, alternate mode should send exit mode command to the port partner.

Figure 31: Alternate mode handler state machine for DFP



6.3.1.2 UFP Handling

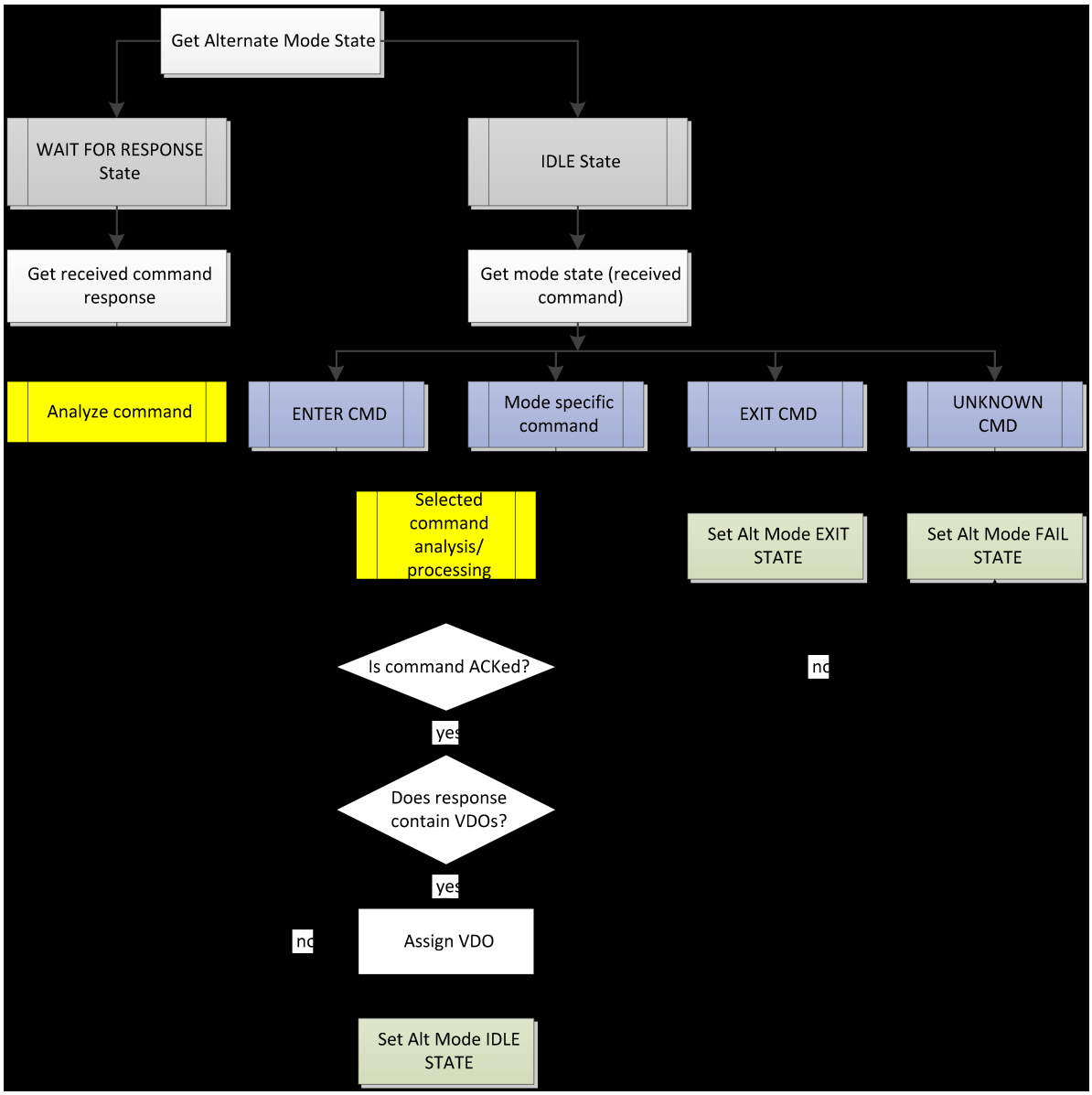
Alternate mode handling when CCG is UFP operates in one of two states:

1. IDLE state is used to analyze received VDMs related to the alternate mode. When a VDM is received, the following should be done:
   a. Analyze the VDM based on the alternate mode rules.
   b. If the VDM received is valid and an ACK response is to be sent, then set alternate mode state to IDLE.
   c. If an alternate mode specific response is to be returned, fill the vdo array with data to be sent and set the alternate mode state to SEND.

2. WAIT_FOR_RESP state is used to process/analyze received VDM responses. When a VDM response is received, the following steps should be performed:
   a. Get internal alternate mode depending on command from VDM header

b. Set alternate mode state to IDLE
c. Analyze the response received
d. If another VDM should be sent after analysis (e.g. attention), then change the internal alternate mode state in compliance with the specific command number, fill the VDO buffer and alternate mode state to the SEND state.

Figure 32: Alternate mode handler state machine for UFP



6.3.2 Registering the Alternate Mode Handlers

1. Update the alt_modes_config.h header file in the project to include information about the alternate modes to be supported. This file contains the following definitions which need to be updated.

    a. DFP_MAX_SVID_SUPP: This constant determines the number of distinct alternate modes supported by CCGx in a DFP role. The number should be less than or equal to 4. If this is non-zero, please ensure that the DFP_ALT_MODE_SUPP setting in config.h is set to 1.

b. UFP_MAX_SVID_SUPP: This constant determines the number of distinct alternate modes supported by CCGx in an UFP role. The number should be less than or equal to 4. If this is non-zero, please ensure that the UFP_ALT_MODE_SUPP setting in config.h is set to 1.

c. supp_svid_tbl: This array holds the list of SVIDs for the alternate modes supported by the CCG device in either DFP or UFP roles.

   e.g: If the design supports two alternate modes with SVIDs SVID1 and SVID2, the table should be initialized as follows:

```
const uint16_t supp_svid_tbl[2] =
{
    SVID1,
    SVID2
};
```

d. is_alt_mode_allowed: This array holds a list of pointers to functions which initialize the corresponding alternate mode operation. The entries in this array should correspond to that in the supp_svid_tbl array, and should provide pointers to functions which will check whether the alternate mode operation is allowed, and then perform the appropriate initialization.

   e.g.: To register handler functions for two alternate modes, the table should be configured as below:

```
alt_mode_info_t*
(*const is_alt_mode_allowed [2]) (uint8_t, alt_mode_reg_info_t*) =
{
    register_svid1,
    register_svid2
};
```

e. dfp_compatibility_mode_table: This array serves as a registry of alternate modes supported by the CCG firmware in a DFP role. This will be a 2-dimensional array which maps the alternate mode SVID value to a unique alternate mode ID value which will be used by the alternate mode manager. The array is arranged in priority order, with the first mentioned SVID being prioritized over the later ones.

   e.g.: If the design supports two alternate modes with SVIDs SVID1 and SVID2, the table can be setup as below:

```
const comp_tbl_t dfp_compatibility_mode_table[2][2] = {
    {
        {SVID1, SVID1_ID},
        {0,     0}
    },
    {
        {0,     0},
        {SVID2, SVID2_ID}
    }
};
```

f. ufp_compatibility_mode_table: This array is similar to the dfp_compatibility_mode_table, and applies when CCG is in the UFP role.

Revision History

Document Revision History

Document Title: **Cypress EZ-PD™ CCGx SDK User Guide** Document Number: 002-12541			
Revision	**Issue Date**	**Origin of Change**	**Description of Change**
**		KYS	Initial release
*A		KNI	Updates for CCGx SDK 3.0.