Cypress EZ-PD™ CCGx Host SDK User Guide

Revision 3.2.1

Doc. No. 002-24327 Rev. **

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 six 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
- EZ-PD™ CCG5: Cypress's second generation Dual-Port Type-C Port Controller
- EZ-PD™ CCG3PA: Cypress's USB Type-C Power Adapter / Power Bank Port Controller

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

1.3 CCGx Host SDK

The CCGx Host Software Development Kit (SDK) is a software solution that allows users to harness the capabilities of the Type-C controllers from Cypress to create PD Port Controller applications for notebook computers, desktops etc.

The following applications are supported by the SDK:

- CCG3 based single port notebook PD controller solution.
- CCG4 based single port notebook PD controller solution using 24-QFN part.
- CCG4 based single port notebook PD controller solution using 40-QFN part.
- CCG4 based dual port notebook PD controller solution.
- CCG5 based single port notebook PD controller solution.
- CCG5 based dual port notebook PD 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 CCG5 controllers.

The key features of CCGx notebook port controller solutions are:

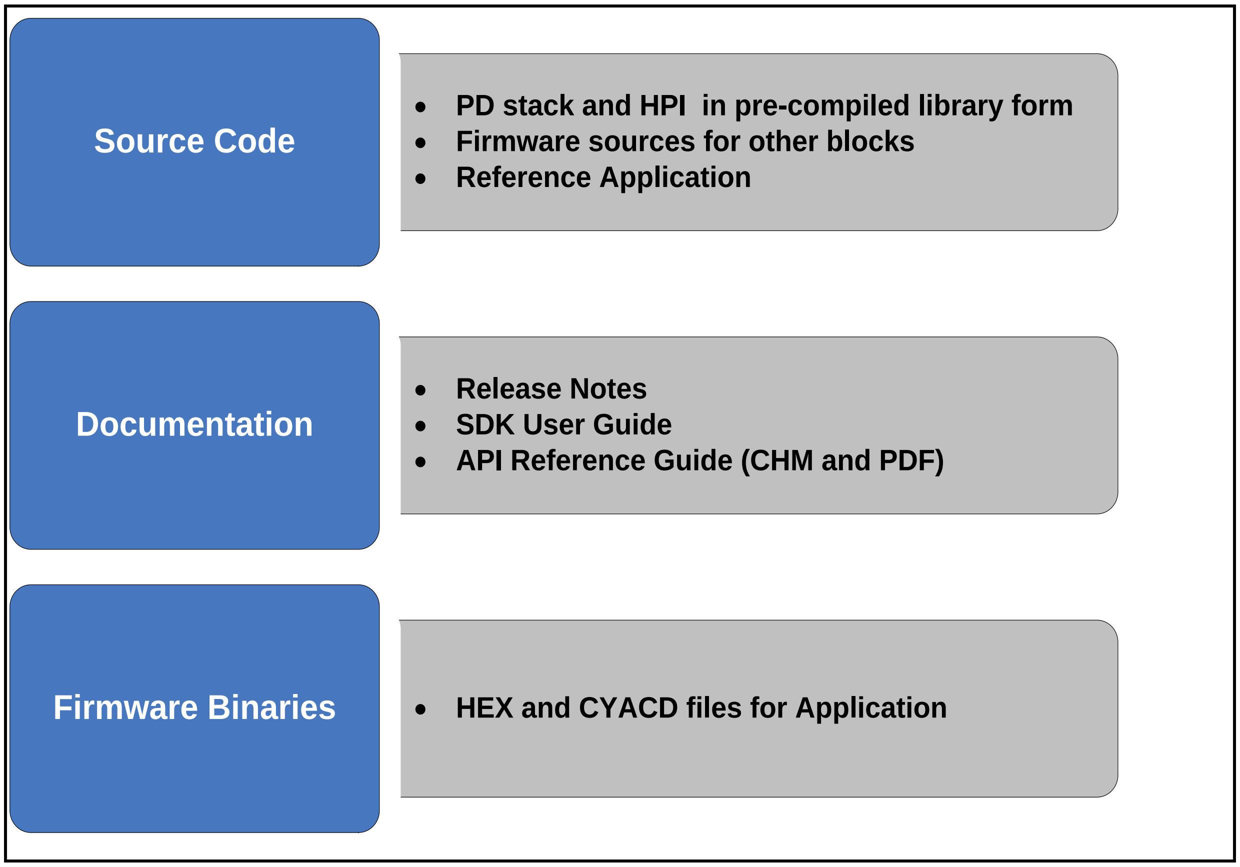
- Compliant to USB-PD specification Revision 3.0, Version 1.1
- Compliant to USB Type-C Specification Revision 1.2
- Support Type-C VBUS OVP and OCP.
- Support DisplayPort alternate mode as a DFP_U/DFP_D.
- Support Host Processor Interface (HPI) for runtime control of power profiles, modes etc.
- Support field firmware upgrades over I2C Interface.

In addition to the common features listed above, the CCG5 based notebook port controller solutions:

- Support BC 1.2 power source (DCP/CDP) operation for charging devices through Type-C to Type-A cable adapters.
- Support VConn OCP protection.

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

Figure 1: CCGx Host SDK Components


Source Code
PD stack and HPI in pre-compiled library form
Firmware sources for other blocks
Reference Application
Documentation
Release Notes
SDK User Guide
API Reference Guide (CHM and PDF)
Firmware Binaries
HEX and CYACD files for Application

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.

1.3.1 SDK Directory Structure

At the top level, the following folders are present:

- **Documentation**: The docs folder contains the EZ-PD™ CCGx Host SDK documentation, which includes release notes, user guide, and API reference guide.
- **Firmware**: The Firmware folder contains the firmware stack sources, reference projects, and pre-built firmware binaries targeted for the Kits and reference designs from Cypress.
    - **binaries**: The binaries folder contains the pre-built firmware binaries
    - **lib**: The lib folder contains the USB-PD stack and HPI module in 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.
    - **projects**: The projects folder contains the sources and PSoC Creator workspaces for the port controller designs.
    - **src**: The src folder contains the sources for the CCGx firmware stack organized by firmware module.

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

The src folder has the following sub-folders:

- **app:** The app folder contains the top-level application layer functionality that implements the required USB-PD controller functions. This includes functionality such as PDO evaluation and contract negotiation, VDM handling for both DFP and UFP roles, handling of control messages such as role swap; and alternate mode discovery and negotiation. The alternate mode specific implementation can be found in the app/alt_mode directory. It also includes handlers for legacy charger detection and Type-A port controller detection.
- **hpiss:** The hpiss folder contains the API interface definition for the Host Processor Interface implemented by the CCG5 firmware.
- **pd_common:** The pd_common folder contains the headers for the core Type-C and USB-PD stack for the CCGx device. This includes the HAL, the Type-C port manager, the USB-PD protocol layer, the USB-PD policy engine, and the Device Policy Manager.
- **pd_hal:** The pd_hal folder contains the low-level driver header and source files for USB-PD hardware block.
- **scb:** The scb folder contains the driver code for I2C slave mode operation using the Serial Controller Blocks (SCB) on the CCGx device. Since I2C slave mode is the most commonly-used interface for CCGx, a specially optimized driver is provided for the same.
- **system:** The system folder contains header and source files relating to the CCGx device hardware and registers, bootloader and flash access functions, low-level drivers for the GPIO blocks on the CCGx device, and a soft timer implementation that is used by the firmware stack.

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

Figure 2: CCGx Host SDK Directory Structure

```
EZ-PD CCGx Host SDK                           CCGx Host SDK Directory
├───CCG2                                      CCG2 Host Directory
│   ├───Documentation                         Documentation: User guide for CCG2 based Host
│   └───Firmware
│       ├───binaries                          Pre-built CCG2 firmware binary
│       └───projects                          Reference Project
└───CCGx                                      CCGx Host Directory
    ├───Documentation                         Documentation: User guide, API guide etc.
    └───Firmware
        ├───binaries                          Pre-built firmware binaries
        │
        ├───lib                               PD stack libraries (Reference only)
        │   ├───ccg3                          Libraries for the CCG3 device
        │   ├───ccg4pd3_singleport            Libraries for the single port CCG4 device
        │   ├───ccg4pd3_dualport              Libraries for the dual port CCG4 device
        │   ├───ccg5_singleport               Libraries for the single port CCG5 device
        │   └───ccg5_dualport                 Libraries for the dual port CCG5 device
        │
        ├───projects                          Reference Projects
        │   ├───CYPD3125-40LQXI_notebook      CCG3 based Notebook PD controller project
        │   ├───CYPD4126-24LQXI_notebook      CCG4 (24-QFN) based Notebook PD controller project
        │   ├───CYPD4126-40LQXI_notebook      CCG4 based single port Notebook PD controller project
        │   ├───CYPD4226-40LQXI_notebook      CCG4 based single port Notebook PD controller project
        │   ├───CYPD5125-40LQXI_notebook      CCG5 based single port Notebook PD controller project
        │   └───CYPD5225-96BZXI_notebook      CCG5 based dual port Notebook PD controller project
        │
        └───src                               Firmware Stack Sources (Reference only)
            ├───app                           Application layer
            ├───hpiss                         Host Processor Interface header file
            ├───pd_common                     PD stack headers
            ├───pd_hal                        PD block low level drivers
            ├───scb                           Serial communication block driver
            └───system                        Low level drivers, firmware update, timer
```

2. SDK Installation

2.1 SDK Installation

Once installed, the directory structure will be as shown in Figure 2.

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. 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 Tool Dependencies

2.2.1 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 CCGx devices, to develop and compile the firmware applications and optionally to program the devices using SWD. This version of the SDK requires PSoC Creator 4.2 build 641 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.

2.2.2 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^{2}C 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^{2}C for programming. Therefore, it will only work with the Cypress-provided kits or other hardware, which includes the Cypress USB – I^{2}C 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 SDK requires the latest EZ-PD Configuration Utility version 1.1, which includes support for programming and configuring CCG2, CCG3, CCG4 and CCG5 devices.

2.3 Hardware Dependencies

The CY4521 kit (http://www.cypress.com/documentation/development-kitsboards/cy4521-ez-pd-ccg2-evaluation-kit) can be used to evaluate the CCG2 firmware solution.

The CY4531 kit (http://www.cypress.com/documentation/development-kitsboards/cy4531-ez-pd-ccg3-evaluation-kit) can be used to evaluate the CCG3 firmware solution.

The CY4541 kit (http://www.cypress.com/documentation/development-kitsboards/cy4541-ez-pdtm-ccg4-evaluation-kit-guide) can be used to evaluate the CCG4 firmware solutions. The kit ships with the CYPD4225-40LQXI controller which supports only USB-PD 2.0 by default. The CCG4 device on the board can be swapped with CYPD4126-40LQXI or CYPD4226-40LQXI devices to work with the projects in the SDK.

Cypress does not provide a CCG5 Evaluation Kit at present. CCG5 solutions should use the reference schematics available in the CCG5 datasheet.

3. Getting Started with CCGx Host SDK

3.1 Using the Reference Projects

As Figure 2 shows, 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 based notebook port controllers (notebook, for short) applications.

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

1. **CYPD2122-24LQXI-notebook**: This project implements a single Type-C port controller for notebook platforms using the CYPD2122-24LQXI (CCG2) device.
2. **CYPD3125-40LQXI_notebook**: This project implements a single Type-C port controller for notebook platforms using the CYPD3125-40LQXI (CCG3) device.
3. **CYPD4126-24LQXI_notebook**: This project implements a single Type-C port controller for notebook platforms using the CYPD4126-24LQXI (CCG4) device.
4. **CYPD4126-40LQXI_notebook**: This project implements a single Type-C port controller for notebook platforms using the CYPD4126-40LQXI (CCG4) device.
5. **CYPD4226-40LQXI_notebook**: This project implements a dual Type-C port controller for notebook platforms using the CYPD4226-40LQXI (CCG4) device.
6. **CYPD5125-40LQXI_notebook**: This project implements a single Type-C port controller for notebook platforms using the CYPD5125-40LQXI device.
7. **CYPD5225-96BZXI_notebook**: This project implements a dual Type-C port controller for notebook platforms using the CYPD5225-96BZXI device.

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.

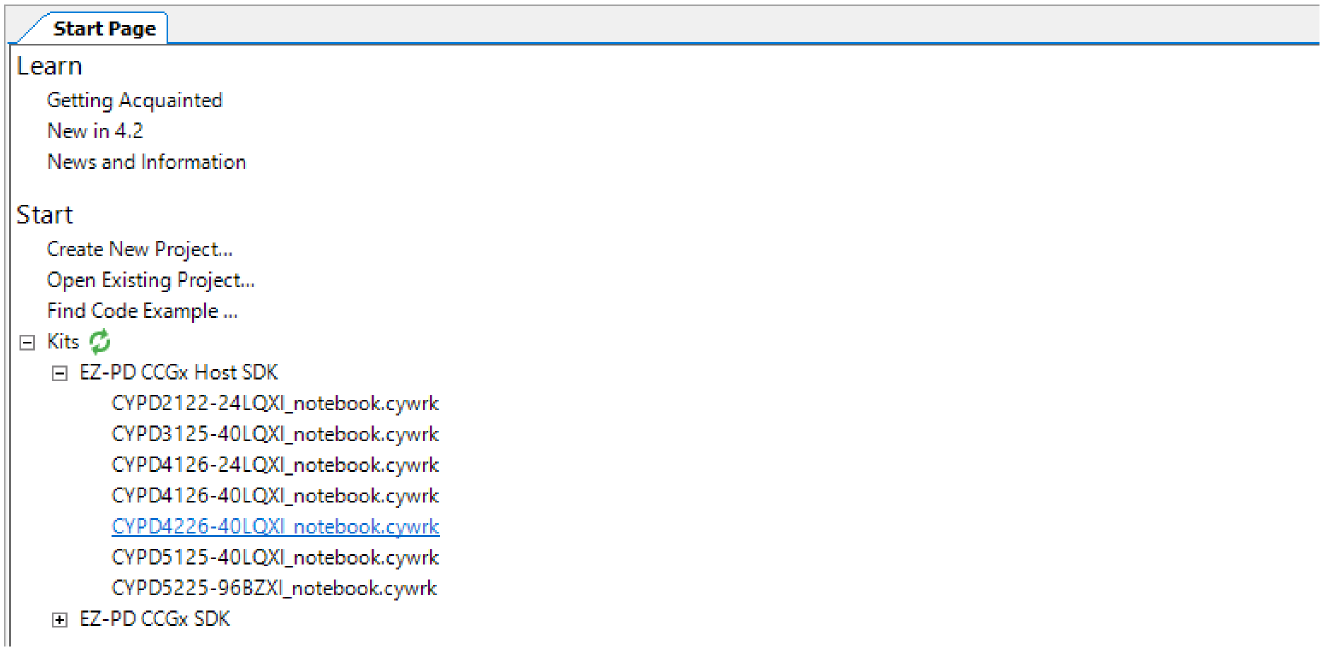
3.1.1 Copying the Project with PSoC Creator

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.

3.1.1.1 From Start Page

The SDK example projects are listed under ***Kits→ EZ-PD CCGx Host 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

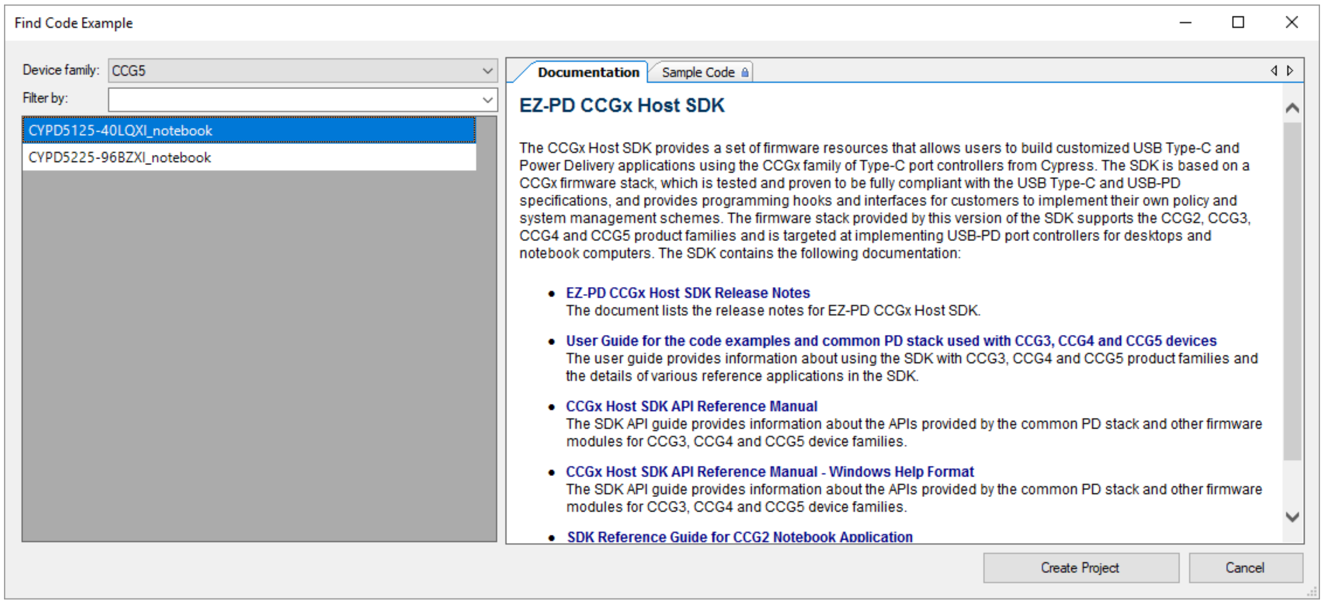

Start Page
Learn
Getting Acquainted
New in 4.2
News and Information
Start
Create New Project...
Open Existing Project...
Find Code Example ...
Kits
EZ-PD CCGx Host SDK
CYPD2122-24LQXI_notebook.cywrk
CYPD3125-40LQXI_notebook.cywrk
CYPD4126-24LQXI_notebook.cywrk
CYPD4126-40LQXI_notebook.cywrk
CYPD4226-40LQXI_notebook.cywrk
CYPD5125-40LQXI_notebook.cywrk
CYPD5225-96BZXI_notebook.cywrk
EZ-PD CCGx SDK

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

3.1.1.2 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: PSoC Creator Code Examples Dialog



EZ-PD CCGx Host SDK

The CCGx Host SDK provides a set of firmware resources that allows users to build customized USB Type-C and Power Delivery applications using the CCGx family of Type-C port controllers from Cypress. The SDK is based on a CCGx firmware stack, which is tested and proven to be fully compliant with the USB Type-C and USB-PD specifications, and provides programming hooks and interfaces for customers to implement their own policy and system management schemes. The firmware stack provided by this version of the SDK supports the CCG2, CCG3, CCG4 and CCG5 product families and is targeted at implementing USB-PD port controllers for desktops and notebook computers. The SDK contains the following documentation:

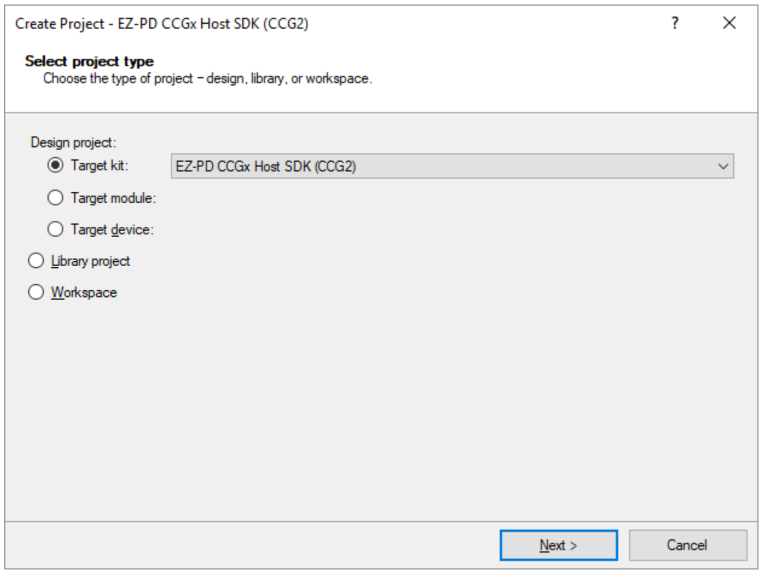
- **EZ-PD CCGx Host SDK Release Notes**
  The document lists the release notes for EZ-PD CCGx Host SDK.
- **User Guide for the code examples and common PD stack used with CCG3, CCG4 and CCG5 devices**
  The user guide provides information about using the SDK with CCG3, CCG4 and CCG5 product families and the details of various reference applications in the SDK.
- **CCGx Host SDK API Reference Manual**
  The SDK API guide provides information about the APIs provided by the common PD stack and other firmware modules for CCG3, CCG4 and CCG5 device families.
- **CCGx Host SDK API Reference Manual - Windows Help Format**
  The SDK API guide provides information about the APIs provided by the common PD stack and other firmware modules for CCG3, CCG4 and CCG5 device families.
- **SDK Reference Guide for CCG2 Notebook Application**

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 this behavior shall not have any impact.

Note: The ***noboot*** and bootloader projects shall not be present in the workspace created through the code example dialog, and shall require to be explicitly added to the workspace. These projects will still be available under the project root folder and you can use the **File → Add → Existing project** option to add them to the workspace.

The **File → New → Project** menu option also allows you to create a new project based on existing example projects. Choose the desired device from the list of EZ-PD CCGx Host SDK items as the Target Kit in the Create Project dialog as shown in Figure 5.

Figure 5: Selection of Target Kit for creation on new project


Create Project - EZ-PD CCGx Host SDK (CCG2)

Select project type
Choose the type of project – design, library, or workspace.

Design project:
- [✓ Target kit:] EZ-PD CCGx Host SDK (CCG2)
- Target module:
- Target device:
- Library project
- Workspace

Then choose Code Example and select the desired code example as shown in Figure 4.

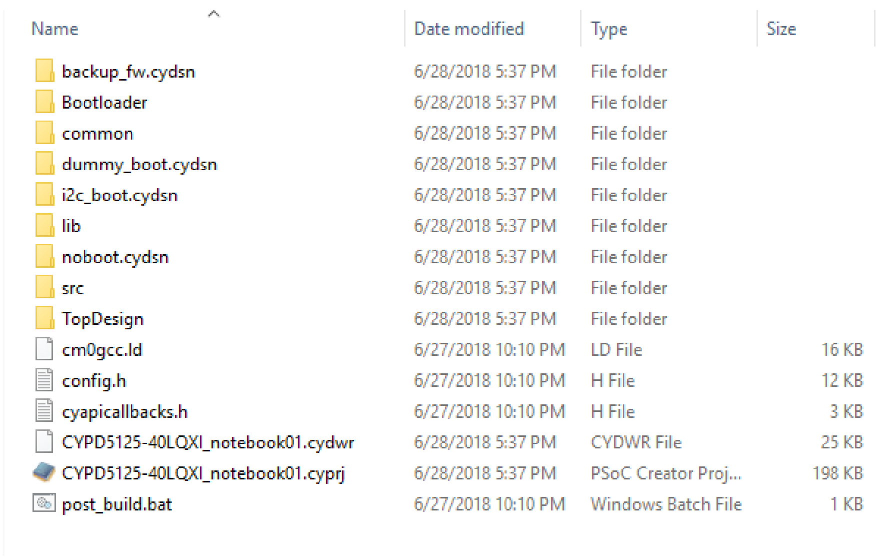
Note: When copying via Create Project option, a wrong device part number may get selected. In this case, the user is expected to change the part number to correct one using ***Device Selector Dialog***.

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 CYPD5125-40LQXI_notebook 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 **CYPD5125-40LQXI_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 (Creator 4.2 or later) is used to open the workspace.

Figure 6: Contents of the Reference Project Folder



Name	Date modified	Type	Size
backup_fw.cydsn	6/28/2018 5:37 PM	File folder	
Bootloader	6/28/2018 5:37 PM	File folder	
common	6/28/2018 5:37 PM	File folder	
dummy_boot.cydsn	6/28/2018 5:37 PM	File folder	
i2c_boot.cydsn	6/28/2018 5:37 PM	File folder	
lib	6/28/2018 5:37 PM	File folder	
noboot.cydsn	6/28/2018 5:37 PM	File folder	
src	6/28/2018 5:37 PM	File folder	
TopDesign	6/28/2018 5:37 PM	File folder	
cm0gcc.ld	6/27/2018 10:10 PM	LD File	16 KB
config.h	6/27/2018 10:10 PM	H File	12 KB
cyapicallbacks.h	6/27/2018 10:10 PM	H File	3 KB
CYPD5125-40LQXI_notebook01.cydwr	6/28/2018 5:37 PM	CYDWR File	25 KB
CYPD5125-40LQXI_notebook01.cyprj	6/28/2018 5:37 PM	PSoC Creator Proj...	198 KB
post_build.bat	6/27/2018 10:10 PM	Windows Batch File	1 KB

3. Once you open the workspace, note that there are three projects:

   a. ***CYPD5125-40LQXI_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 CCG5 device. More details on this project are provided in later sections.

   b. ***backup_fw.cydsn***: This is a limited feature version of the notebook PD port controller application using the CCG5 device. This is used as a secondary firmware binary which allows recovery in case the device firmware gets corrupted during an upgrade process.

   c. ***noboot.cydsn***: The main firmware project in **CYPD5125-40LQXI_notebook.*cydsn*** does not support runtime debugging through the SWD interface. The *noboot.cydsn* is a version of the same firmware application, which does not depend on the bootloader. This firmware overwrites the complete device flash and expects that the device will be programmed through SWD.

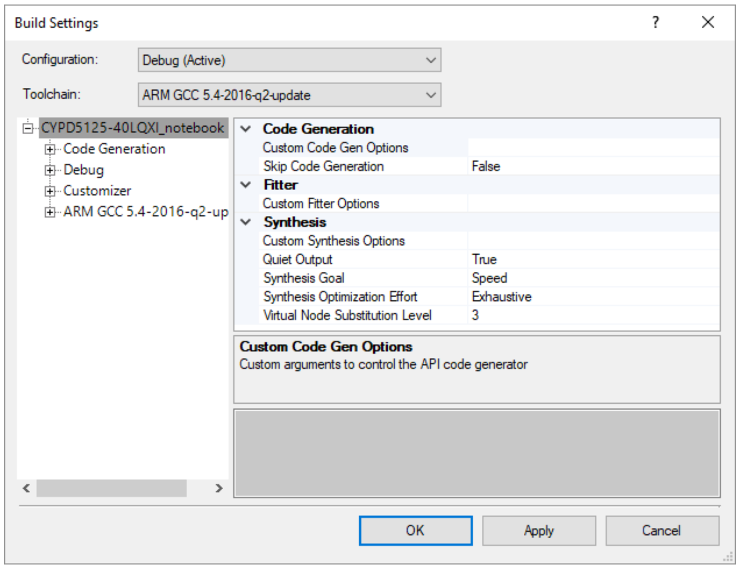
   NOTE*:** If the project was copied using Code Examples, then only the main project shall be seen on the workspace. The other two projects can be added to the workspace using ***Add Existing Project option.

   There is a fourth project available (***i2c_boot.cydsn***) in the project workspace directory. This is the bootloader project available for reference. The pre-built bootloader binary from the Bootloader directory should be used unless boot flow modifications are required. This project is not added to the workspace and can be added using ***Add Existing Project*** option.

4. The ***CYPD5125-40LQXI_notebook.cydsn*** project is set as the default project for the workspace. Choose the **Build CYPD5125-40LQXI_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 5.4-2016-q2-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



Configuration: Debug (Active)
Toolchain: ARM GCC 5.4-2016-q2-update

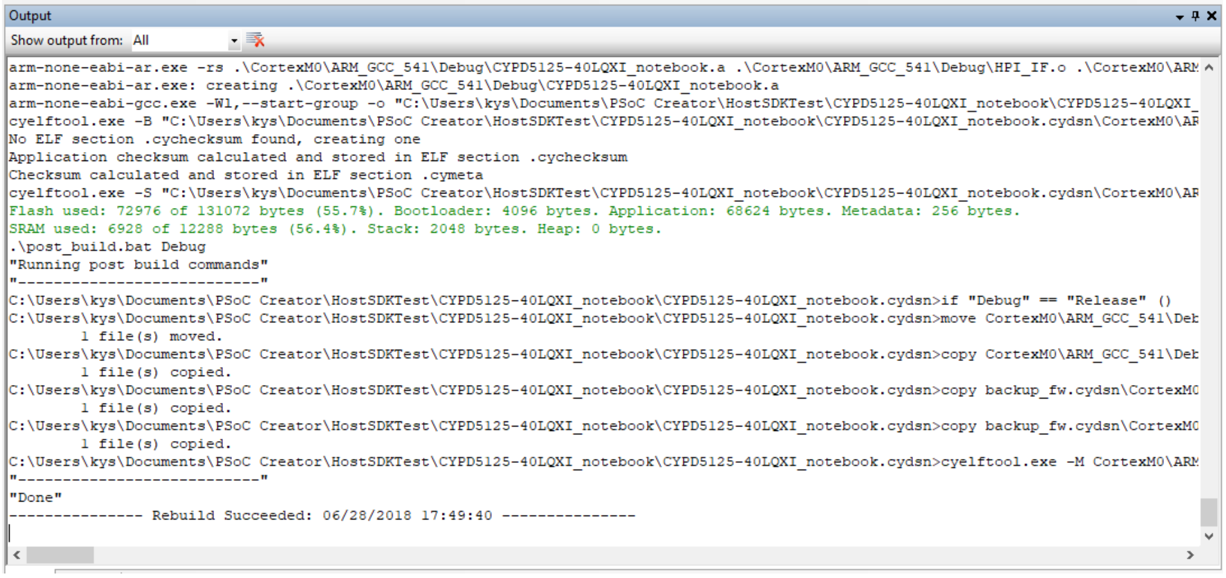
CYPD5125-40LQXI_notebook
- Code Generation
- Debug
- Customizer
- ARM GCC 5.4-2016-q2-up

Code Generation	
Custom Code Gen Options	
Skip Code Generation	False
Fitter	
Custom Fitter Options	
Synthesis	
Custom Synthesis Options	
Quiet Output	True
Synthesis Goal	Speed
Synthesis Optimization Effort	Exhaustive
Virtual Node Substitution Level	3

Custom Code Gen Options
Custom arguments to control the API code generator

6. You may receive a pop-up window asking for permission to make the project file writeable. 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 8 at the end of the build process.

Figure 8: Output Window after the Build is Complete

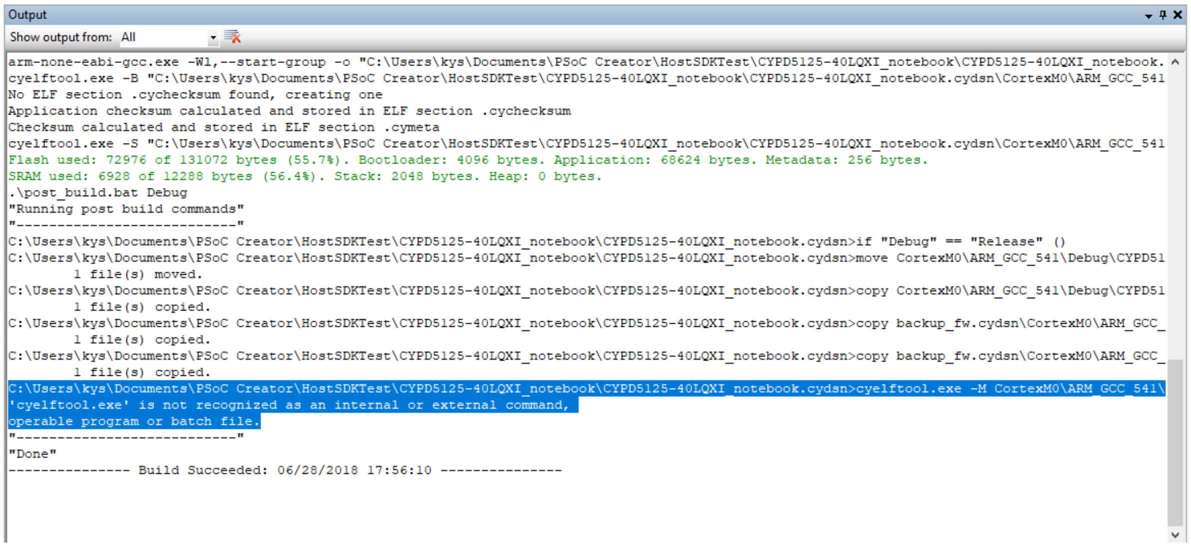


Show output from: All

```
arm-none-eabi-ar.exe -rs .\CortexM0\ARM_GCC_541\Debug\CYPD5125-40LQXI_notebook.a .\CortexM0\ARM_GCC_541\Debug\HPI_IF.o .\CortexM0\ARM
arm-none-eabi-ar.exe: creating .\CortexM0\ARM_GCC_541\Debug\CYPD5125-40LQXI_notebook.a
arm-none-eabi-gcc.exe -Wl,--start-group -o "C:\Users\kys\Documents\PSoC Creator\HostSDKTest\CYPD5125-40LQXI_notebook\CYPD5125-40LQXI_
cyelftool.exe -B "C:\Users\kys\Documents\PSoC Creator\HostSDKTest\CYPD5125-40LQXI_notebook\CYPD5125-40LQXI_notebook.cydsn\CortexM0\AR
No ELF section .cychecksum found, creating one
Application checksum calculated and stored in ELF section .cychecksum
Checksum calculated and stored in ELF section .cymeta
cyelftool.exe -S "C:\Users\kys\Documents\PSoC Creator\HostSDKTest\CYPD5125-40LQXI_notebook\CYPD5125-40LQXI_notebook.cydsn\CortexM0\AR
Flash used: 72976 of 131072 bytes (55.7%). Bootloader: 4096 bytes. Application: 68624 bytes. Metadata: 256 bytes.
SRAM used: 6928 of 12288 bytes (56.4%). Stack: 2048 bytes. Heap: 0 bytes.
.\post_build.bat Debug
"Running post build commands"
"---------------------------"
C:\Users\kys\Documents\PSoC Creator\HostSDKTest\CYPD5125-40LQXI_notebook\CYPD5125-40LQXI_notebook.cydsn>if "Debug" == "Release" ()
C:\Users\kys\Documents\PSoC Creator\HostSDKTest\CYPD5125-40LQXI_notebook\CYPD5125-40LQXI_notebook.cydsn>move CortexM0\ARM_GCC_541\Deb
        1 file(s) moved.
C:\Users\kys\Documents\PSoC Creator\HostSDKTest\CYPD5125-40LQXI_notebook\CYPD5125-40LQXI_notebook.cydsn>copy CortexM0\ARM_GCC_541\Deb
        1 file(s) copied.
C:\Users\kys\Documents\PSoC Creator\HostSDKTest\CYPD5125-40LQXI_notebook\CYPD5125-40LQXI_notebook.cydsn>copy backup_fw.cydsn\CortexM0
        1 file(s) copied.
C:\Users\kys\Documents\PSoC Creator\HostSDKTest\CYPD5125-40LQXI_notebook\CYPD5125-40LQXI_notebook.cydsn>copy backup_fw.cydsn\CortexM0
        1 file(s) copied.
C:\Users\kys\Documents\PSoC Creator\HostSDKTest\CYPD5125-40LQXI_notebook\CYPD5125-40LQXI_notebook.cydsn>cyelftool.exe -M CortexM0\ARM
"---------------------------"
"Done"
--------------- Rebuild Succeeded: 06/28/2018 17:49:40 ---------------
```

Note: When building CCG5 projects, you may see an error while the post build script which combines the backup and primary firmware images is running as shown in Figure 9. In such case, the HEX file generated will not be functional. This error can be resolved by adding the ***C:\Program Files (x86)\Cypress\PSoC Creator\4.2\PSoC Creator\bin*** (or actual path where PSoC Creator 4.2 is installed) folder the system environment path and restarting PSoC Creator.

Figure 9: Post-build script error in CCG5 reference projects



Output

Show output from: All

```
arm-none-eabi-gcc.exe -Wl,--start-group -o "C:\Users\kys\Documents\PSoC Creator\HostSDKTest\CYPD5125-40LQXI_notebook\CYPD5125-40LQXI_notebook.
cyelftool.exe -B "C:\Users\kys\Documents\PSoC Creator\HostSDKTest\CYPD5125-40LQXI_notebook\CYPD5125-40LQXI_notebook.cydsn\CortexM0\ARM_GCC_541
No ELF section .cychecksum found, creating one
Application checksum calculated and stored in ELF section .cychecksum
Checksum calculated and stored in ELF section .cymeta
cyelftool.exe -S "C:\Users\kys\Documents\PSoC Creator\HostSDKTest\CYPD5125-40LQXI_notebook\CYPD5125-40LQXI_notebook.cydsn\CortexM0\ARM_GCC_541
Flash used: 72976 of 131072 bytes (55.7%). Bootloader: 4096 bytes. Application: 68624 bytes. Metadata: 256 bytes.
SRAM used: 6928 of 12288 bytes (56.4%). Stack: 2048 bytes. Heap: 0 bytes.
.\post_build.bat Debug
"Running post build commands"
"---------------------------"
C:\Users\kys\Documents\PSoC Creator\HostSDKTest\CYPD5125-40LQXI_notebook\CYPD5125-40LQXI_notebook.cydsn>if "Debug" == "Release" ()
C:\Users\kys\Documents\PSoC Creator\HostSDKTest\CYPD5125-40LQXI_notebook\CYPD5125-40LQXI_notebook.cydsn>move CortexM0\ARM_GCC_541\Debug\CYPD51
        1 file(s) moved.
C:\Users\kys\Documents\PSoC Creator\HostSDKTest\CYPD5125-40LQXI_notebook\CYPD5125-40LQXI_notebook.cydsn>copy CortexM0\ARM_GCC_541\Debug\CYPD51
        1 file(s) copied.
C:\Users\kys\Documents\PSoC Creator\HostSDKTest\CYPD5125-40LQXI_notebook\CYPD5125-40LQXI_notebook.cydsn>copy backup_fw.cydsn\CortexM0\ARM_GCC_
        1 file(s) copied.
C:\Users\kys\Documents\PSoC Creator\HostSDKTest\CYPD5125-40LQXI_notebook\CYPD5125-40LQXI_notebook.cydsn>copy backup_fw.cydsn\CortexM0\ARM_GCC_
        1 file(s) copied.
C:\Users\kys\Documents\PSoC Creator\HostSDKTest\CYPD5125-40LQXI_notebook\CYPD5125-40LQXI_notebook.cydsn>cyelftool.exe -M CortexM0\ARM_GCC_541\
'cyelftool.exe' is not recognized as an internal or external command,
operable program or batch file.
"---------------------------"
"Done"
--------------- Build Succeeded: 06/28/2018 17:56:10 ---------------
```

7. Now, navigate to the project folder using Windows Explorer to locate the compiled firmware binaries. Navigate to the CYPD5125-40LQXI_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. ***CYPD5125-40LQXI_notebook.hex***: This is an SWD programmable binary file in the Intel Hex format that combines the bootloader as well as the notebook port controller firmware application.

    b. ***CYPD5125-40LQXI_notebook_2.cyacd***: This binary file contains the full-featured firmware application that can be loaded to the CCG5 device flash through the I2C interface. 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 CCG5 device.

    c. ***CYPD5125-40LQXI_notebook_1.cyacd***: This binary file contains the limited feature backup firmware application that can be loaded to the CCG5 device flash through the I2C interface. 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 CCG5 device.

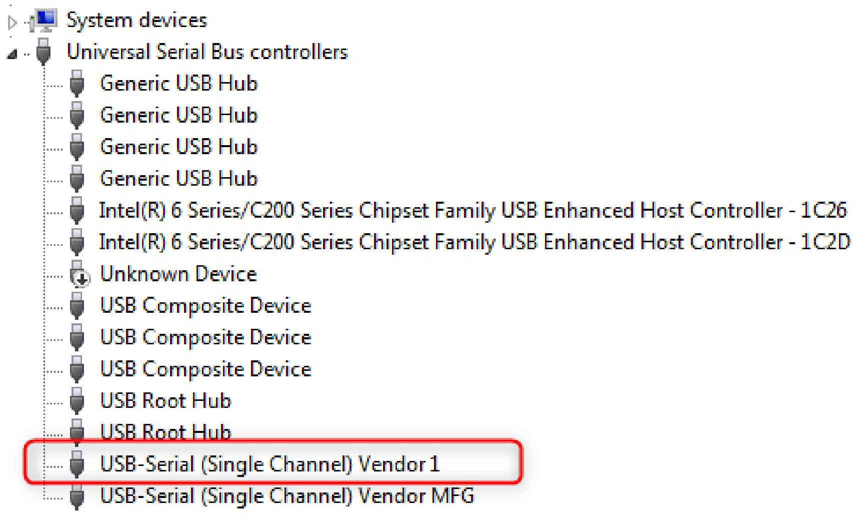
3.1.3 Programming CCGx using the EZ-PD Configuration Utility

If the CCG5 evaluation board is being used, the firmware binaries built using the above procedure can be loaded on to the CCG5 device using the EZ-PD Configuration Utility. This section provides step-by-step instructions for updating the firmware with the EZ-PD Configuration Utility. Refer to the EZ-PD Configuration Utility User Manual for more details.

1. Power up the CCG5 evaluation board using the 20V power adapter and connect a USB cable from the Mini-B connector to the host PC.

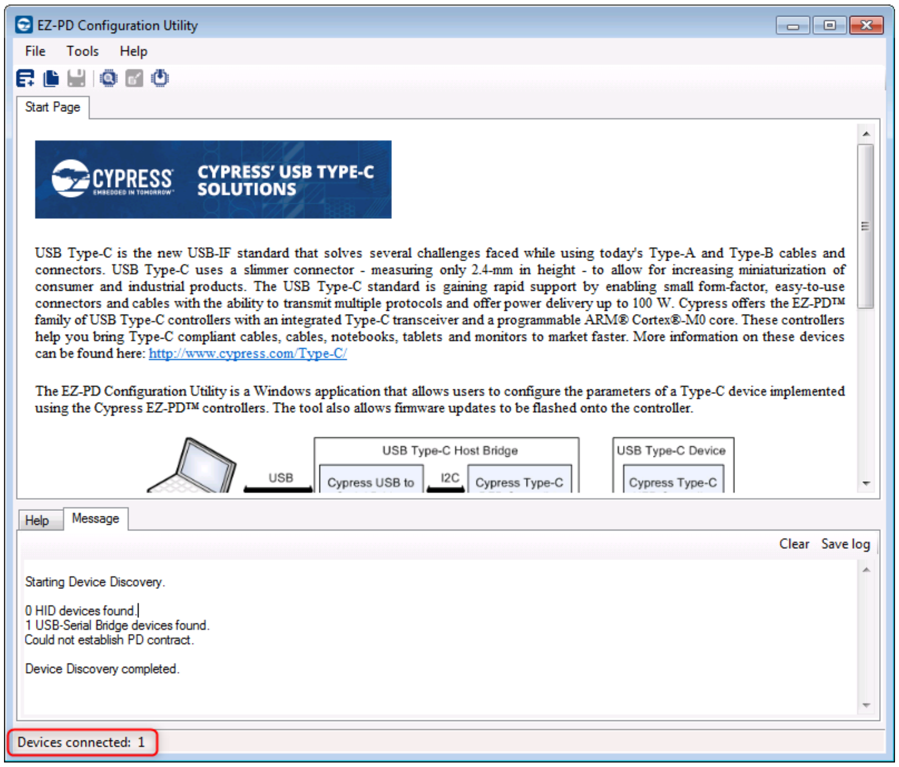
2. Wait for driver detection and binding for the USB-Serial controller on the board. The driver for this controller can be obtained by searching on Windows Update. Once the driver binding is successful, a “USB-Serial (Single Channel) Vendor 1” device will be listed under ‘Universal Serial Bus Controllers’ in the **Device Manager** window. See Figure 10 for the expected device listing.

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


System devices
Universal Serial Bus controllers
Generic USB Hub
Generic USB Hub
Generic USB Hub
Generic USB Hub
Intel(R) 6 Series/C200 Series Chipset Family USB Enhanced Host Controller - 1C26
Intel(R) 6 Series/C200 Series Chipset Family USB Enhanced Host Controller - 1C2D
Unknown Device
USB Composite Device
USB Composite Device
USB Composite Device
USB Root Hub
USB Root Hub
USB-Serial (Single Channel) Vendor 1
USB-Serial (Single Channel) Vendor MFG

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 a device connected on the lower border of the UI as shown in Figure 11.

Figure 11: Configuration Utility Detecting the Connected CCG5 evaluation board



USB Type-C is the new USB-IF standard that solves several challenges faced while using 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 allow for 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 with the ability to transmit multiple protocols and offer power delivery up to 100 W. Cypress offers the EZ-PD™ family of USB Type-C controllers with an integrated Type-C transceiver and a programmable ARM® Cortex®-M0 core. These controllers help you bring Type-C compliant cables, cables, notebooks, tablets and monitors to market faster. More information on these devices can be found here: http://www.cypress.com/Type-C/

The EZ-PD Configuration Utility is a Windows application that allows users to configure the parameters of a Type-C device implemented using the Cypress EZ-PD™ controllers. The tool also allows firmware updates to be flashed onto the controller.

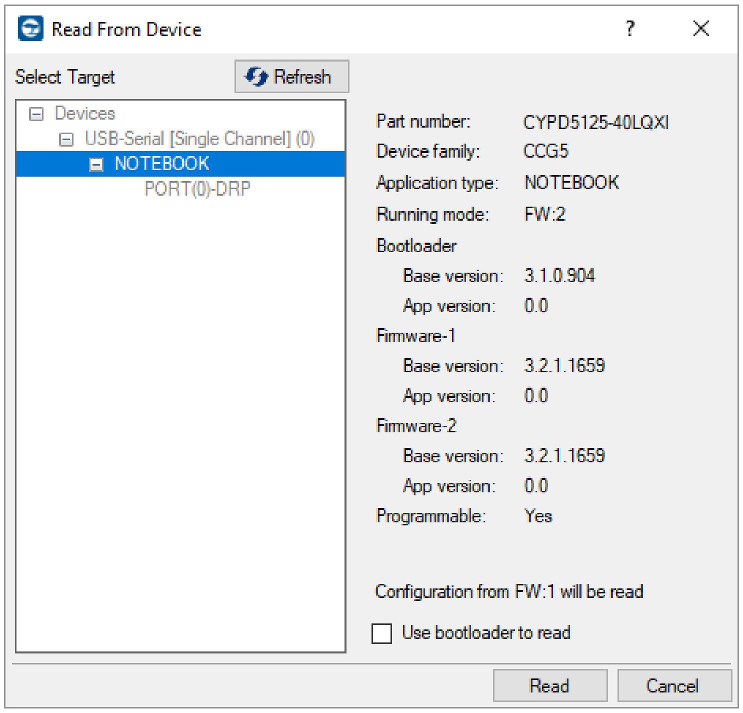
```
Starting Device Discovery.

0 HID devices found.
1 USB-Serial Bridge devices found.
Could not establish PD contract.

Device Discovery completed.
```

Devices connected: 1

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 required node (**Notebook**) in the device tree, the UI displays information about the CCGx device and the current firmware running on it.

Figure 12: Firmware Update Dialog



Part number:	CYPD5125-40LQXI
Device family:	CCG5
Application type:	NOTEBOOK
Running mode:	FW:2
Bootloader	
Base version:	3.1.0.904
App version:	0.0
Firmware-1	
Base version:	3.2.1.1659
App version:	0.0
Firmware-2	
Base version:	3.2.1.1659
App version:	0.0
Programmable:	Yes

Configuration from FW:1 will be read

☐ Use bootloader to read

7. Navigate to the folder containing the firmware binaries generated during the firmware build, and select the *CYPD5125-40LQXI_notebook_1.cyacd* and *CYPD5125-40LQXI_notebook_2.cyacd* files in the two firmware path options in the dialog.

8. Check the "*Use bootloader to flash"* option so that both banks of firmware can be updated in one step; and click on the ***Program*** button to start the firmware update. The full update will take about 30 seconds.

9. Once the update process is complete, use the **Tools** > **Read from Device** 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.

10. Since the Firmware-2 binary is the full-featured application, the CCG5 bootloader is designed such that it loads the Firmware-2 whenever it is present (independent of the firmware version).

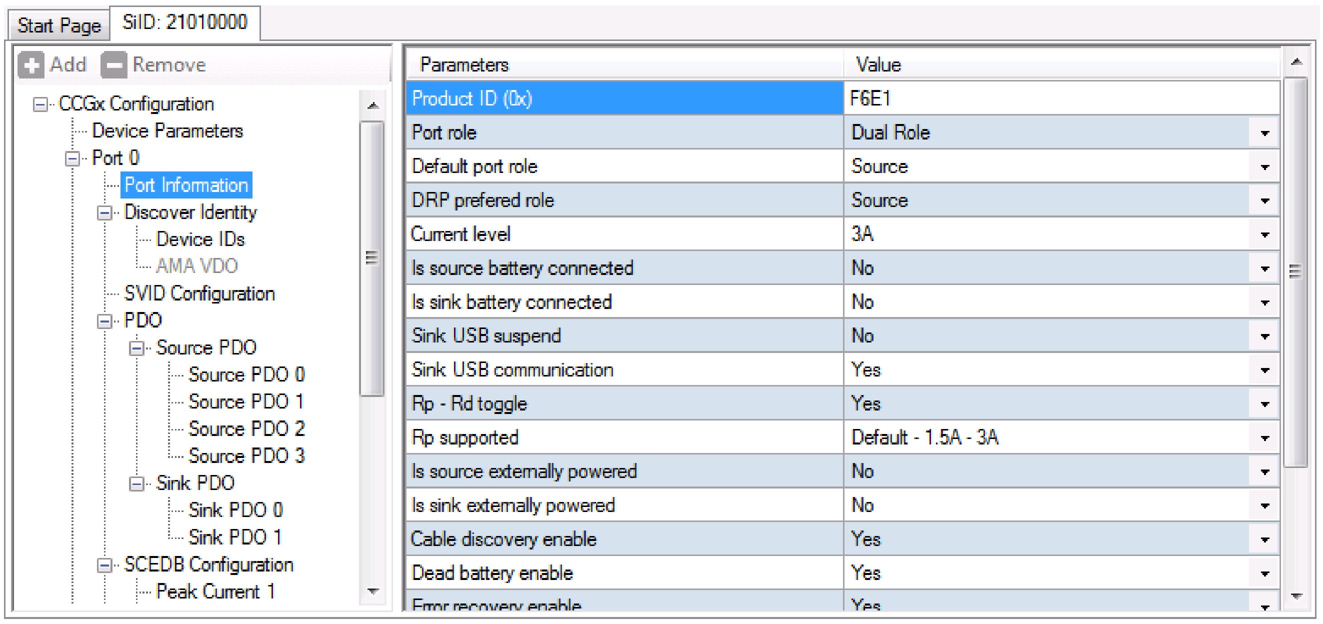
3.2 Updating CCGx Configuration

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 port roles supported by the device (Source/Sink/Dual Role).
- Enable/disable flags and parameters that control the various features like overvoltage, 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. The utility provides an interactive GUI through which all of the contents of the configuration table can be updated.

Figure 13: PD Port Configuration using EZ-PD Configuration Utility


Start Page
SiID: 21010000
Add
Remove

CCGx Configuration
- Device Parameters
- Port 0
  - Port Information
  - Discover Identity
    - Device IDs
    - AMA VDO
  - SVID Configuration
  - PDO
    - Source PDO
      - Source PDO 0
      - Source PDO 1
      - Source PDO 2
      - Source PDO 3
    - Sink PDO
      - Sink PDO 0
      - Sink PDO 1
  - SCEDB Configuration
    - Peak Current 1

Parameters	Value
Product ID (0x)	F6E1
Port role	Dual Role
Default port role	Source
DRP prefered role	Source
Current level	3A
Is source battery connected	No
Is sink battery connected	No
Sink USB suspend	No
Sink USB communication	Yes
Rp - Rd toggle	Yes
Rp supported	Default - 1.5A - 3A
Is source externally powered	No
Is sink externally powered	No
Cable discovery enable	Yes
Dead battery enable	Yes
Error recovery enable	Yes

Figure 13 shows a snapshot of the UI screens used for configuring the CCG5 firmware 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.

Table 1 shows the full list of configuration parameters relevant to the CCG5 notebook PD port controller solution. 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. 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 1: List of CCG5 Notebook Configuration Parameters

Configuration Parameter	Default Value	Change Allowed
Device Parameters		
Part Number	<As per Project>	Select the part number matching the application.
Manufacturer Info	"Cypress"	Can be changed. Valid only for PD 3.0 configurations.
Vendor ID	0x04B4	The USB Vendor ID assigned for the product.
Config table major version	2	The version of the configuration table format. Leave this with the default value.
Port Information		
Product ID	<As per Project>	The USB Product ID assigned for the product.
Port role	Dual role	Dual role for designs that allow charging through Type-C port, Source otherwise.
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	Can be changed.
Sink USB communication	Yes	Change not recommended.
Rp-Rd Toggle	Yes	Set to No if this is a Source only port.
Rp supported	Default, 1.5A and 3.0A	Can be changed
Is source externally powered	No	Can be changed based on system design.
Is sink externally powered	No	Can be changed based on system design.
Cable discovery enable	Yes	Change not recommended
Dead battery enable	Yes	Change not recommended
Error recovery enable	Yes	Change 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
Device IDs		
USB host support	Yes	Should be Yes for desktops and notebooks.
USB device support	No	Should be No for desktops and notebooks.
Modal operation supported	Yes	Set this to No if no UFP alternate modes are required.
USB Vendor Id	0x04B4	Can be changed
Product type (UFP)	AMA	Can be changed

Product type (DFP)	AMC	Change not recommended
USB-ID compliance XID	0	Can be changed
USB Product ID	<As per project>	This gets changed through Port Information node.
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	Not applicable for receptacle based designs.
VConn required	No	Should be No for receptacle based designs.
VBus required	Yes	Can be changed
USB Version	Gen1, Gen2 and USB 2.0	Can be changed
SVID Configuration		
SVID	0x8087	Can be changed. Remove this if Thunderbolt 3 is not supported.
Mode	1	Can be changed.
Source PDOs		
Source PDO 0	5 V @ 3A	Current can be changed
Source PDO 1	9 V @ 3A	Can be changed
Source PDO 2	15 V @ 3A	Can be changed
Source PDO 3	20 V @ 3A	Can be changed
Sink PDOs		
Sink PDO 0	5 V @ 0.9 A	Current can be changed.
Sink PDO 1	7 to 21 V @ 0.9 A	Can be changed.
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)	0	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	Supported	Can be changed
Touch temp	Default	Can be changed
Source Inputs-External supplies	Constrained 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
Source PD Power	60	Can be changed. Should match Source PDOs.
Peak Current		
Percentage overload (%)	0	Can be changed
Overload period (ms)	0	Can be changed
Duty cycle (%)	0	Can be changed
Vbus voltage droop	No	Can be changed
DP Mode Parameters		
Modes supported	CDEF	Can be changed
Mux Control	Controlled by CCGx	Can be changed based on hardware design.
Mode trigger	Automatic	Can be changed
Preferred DP Mode	4-lane DP	Not applicable for Notebook/Desktop designs.
Power Protection		
Over Voltage Protection	Enable	Can be changed
OVP Threshold	20%	Can be changed
OVP debounce period	10 us	Can be changed
OVP retry count	2	Can be changed
Over Current Protection	Enable	Can be changed
OCP threshold	20%	Can be changed
OCP debounce period	10ms	Can be changed
OCP retry count	2	Can be changed
OCP threshold-2	50	Not supported as of now
OCP debounce period – 2	1 ms	Not supported as of now
VConn OCP	Enable	Can be changed
Threshold	30%	Not applicable. OCP threshold is fixed.
Debounce period	1 ms	Change not recommended.
User Parameters		
Parameters 1 to 8	00	Can be changed to any value

The various fields are inter-related, and should be updated to be mutually consistent. 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. The utility issues a warning if the firmware version is older than the current version, and you have the option of aborting the configuration update at this stage.

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.

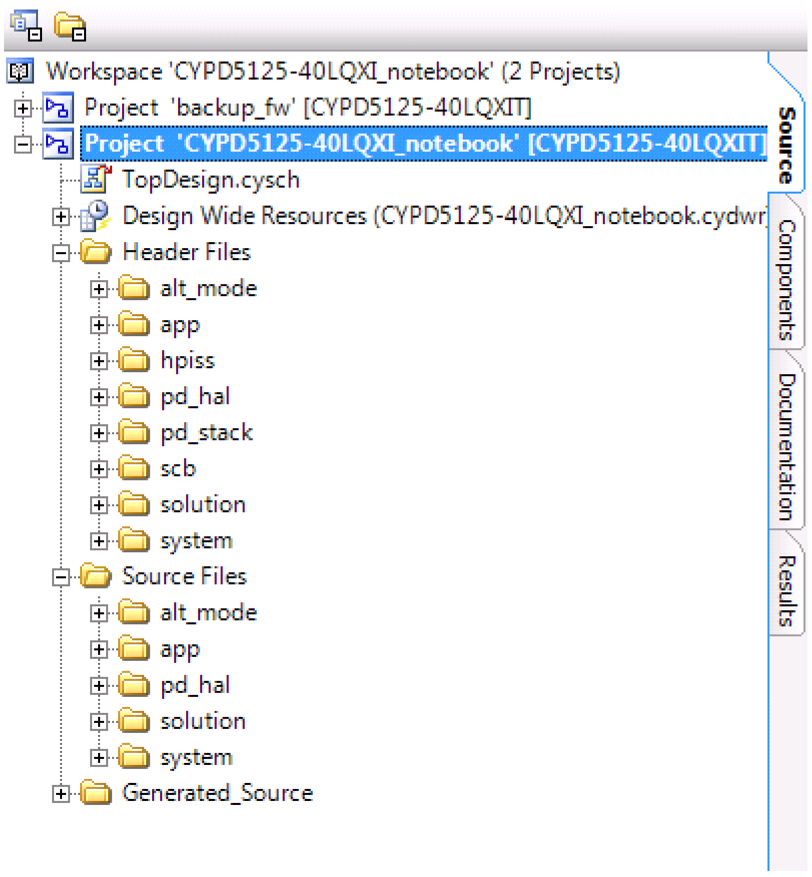
Note: As the firmware sources and reference projects are installed in the **Program Files** folder, it is not recommended that you make changes to the original installed version of these files. You can create a copy of the Firmware folder from the SDK installation, and use the copy for making any changes. The code example option of PSoC Creator will allow you to make copies of the projects. This will ensure that you have a clean version of the files that you can revert to as well. Refer to section 3.1.1 for 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 14. The figure uses the CYPD5125-40LQXI_notebook workspace as reference. The source and header files used in the solution are grouped into different folders.

Figure 14: CCG5 Notebook Solution Structure


Workspace 'CYPD5125-40LQXI_notebook' (2 Projects)
Project 'backup_fw' [CYPD5125-40LQXIT]
Project 'CYPD5125-40LQXI_notebook' [CYPD5125-40LQXIT]
TopDesign.cysch
Design Wide Resources (CYPD5125-40LQXI_notebook.cydwr)
Header Files
alt_mode
app
hpiss
pd_hal
pd_stack
scb
solution
system
Source Files
alt_mode
app
pd_hal
solution
system
Generated_Source

- **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.
  - **stack_params.h**: Configuration parameters used by the PD stack and application layers which manage the PD power negotiation. The content of this file is not expected to be changed by users.
  - **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).
  - **solution.h**: Header file providing constant, variable and function declarations for external hardware controls implemented by CCG5 firmware.
  - **instrumentation.h**: Header file providing declarations for instrumentation code which tracks firmware responsiveness and runtime stack usage.
  - **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.
  - **solution.c**: This source file contains the functions that control the data switch and/or external buck-boost regulators used in the system for data connection and power control.
  - **instrumentation.c**: This source file implements the instrumentation code that tracks firmware responsiveness and runtime stack usage.
  - **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 CCGx 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:
    - **pdss_mx_hal.c:** The hardware interface file to the PD block.
    - **hal_ccgx.c**: This source file implements various protection 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.
- **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.
- **hpiss**: Header file which defines the Host Processor Interface related functions.
- **scb**: Header file which defines the generic I2C slave driver used for HPI and other functions.
- **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 Compile Time Options

The 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 2: Selectable Firmware Features in config.h.

Table 2: Selectable Firmware Features in config.h

Pre-processor Switch	Description	Values
APP_VBUS_SRC_FET_ ON_P1	Function/Macro to turn on the port 1 source FET.	Map to a function or macro which disables the source (provider) FET associated with PD port 1.
APP_VBUS_SRC_FET_ OFF_P1	Function/Macro to turn off the port 1 source FET.	Map to a function or macro which enables the source (provider) FET associated with PD port 1.
APP_VBUS_SRC_FET_ ON_P2	Function/Macro to turn on the port 2 source FET. Only applicable for a dual-port solution.	Map to a function or macro which disables the source (provider) FET associated with PD port 2.
APP_VBUS_SRC_FET_ OFF_P2	Function/Macro to turn off the port 2 source FET. Only applicable for a dual-port solution.	Map to a function or macro which enables the source (provider) FET associated with PD port 2.
APP_VBUS_SNK_FET_OFF_P1	Function/Macro to turn on the port 1 sink FET.	Map to a function or macro which disables the sink (consumer) FET associated with PD port 1.

Pre-processor Switch	Description	Values
APP_VBUS_SNK_FET_ON_P1	Function/Macro to turn off the port 1 sink FET.	Map to a function or macro which enables the sink (consumer) FET associated with PD port 1.
APP_VBUS_SNK_FET_OFF_P2	Function/Macro to turn on the port 2 sink FET. Only applicable for a dual-port solution.	Map to a function or macro which disables the sink (consumer) FET associated with PD port 2.
APP_VBUS_SNK_FET_ON_P2	Function/Macro to turn off the port 2 sink FET. Only applicable for a dual-port solution.	Map to a function or macro which enables the sink (consumer) FET associated with PD port 2.
APP_DISCHARGE_FET_ON_P1	Function/Macro to turn on the port 1 VBus discharge circuit.	Map to a function or macro which enables the VBus discharge circuit associated with PD Port 1. The internal discharge FET on the CCG5 device can be used for this purpose.
APP_DISCHARGE_FET_OFF_P1	Function/Macro to turn off the port 1 VBus discharge circuit.	Map to a function or macro which disables the VBus discharge circuit associated with PD Port 1. The internal discharge FET on the CCG5 device can be used for this purpose.
APP_DISCHARGE_FET_ON_P2	Function/Macro to turn on the port 2 VBus discharge circuit. Only applicable for a dual-port solution.	Map to a function or macro which enables the VBus discharge circuit associated with PD Port 2. The internal discharge FET on the CCG5 device can be used for this purpose.
APP_DISCHARGE_FET_OFF_P2	Function/Macro to turn off the port 2 VBus discharge circuit. Only applicable for a dual-port solution.	Map to a function or macro which disables the VBus discharge circuit associated with PD Port 2. The internal discharge FET on the CCG5 device can be used for this purpose.
CCG_PROG_SOURCE_ENABLE	Select whether power source supports variable (not a discrete set) of VBus voltage settings.	Set to 1 if there is a programmable regulator to provide the power output. Set to 0 if the regulator only supports discrete voltage settings of 5V, 9V, 12V, 15V and 20V.
APP_VBUS_SET_VOLT_P1	Function/Macro to select the source voltage (VBus output) on PD port 1. Only required when CCG_PROG_SOURCE_ENABLE is set to 1.	Map to a function or macro which updates the on-board regulator to source the required output voltage.

Pre-processor Switch	Description	Values
APP_VBUS_SET_VOLT_P2	Function/Macro to select the source voltage (VBus output) on PD port 2. Only required when CCG_PROG_SOURCE_ENABLE is set to 1.	Map to a function or macro which updates the on-board regulator to source the required output voltage.
APP_VBUS_SET_5V_P1 APP_VBUS_SET_9V_P1 APP_VBUS_SET_12V_P1 APP_VBUS_SET_15V_P1 APP_VBUS_SET_20V_P1	Function/Macro to set the output voltage on Port 1 to the desired level. Only required when CCG_PROG_SOURCE_ENABLE is set to 0.	Map to functions or macros which select the output voltage from the on-board regulator. The implementation for unsupported voltages can be left as NOP.
APP_VBUS_SET_5V_P2 APP_VBUS_SET_9V_P2 APP_VBUS_SET_12V_P2 APP_VBUS_SET_15V_P2 APP_VBUS_SET_20V_P2	Function/Macro to set the output voltage on Port 2 to the desired level. Only required when CCG_PROG_SOURCE_ENABLE is set to 0.	Map to functions or macros which select the output voltage from the on-board regulator. The implementation for unsupported voltages can be left as NOP.
BC_1_2_SRC_ENABLE	Enable/disable BC 1.2 (CDP/DCP) source support on the USB ports.	Set to 1 to enable BC 1.2 power source support. Set to 0 to disable BC 1.2 power source support.
CCG_BC_12_IN_PD_ENABLE	Enable/disable CDP operation after USB-PD contract is in place.	Set to 1 to enable support for CDP negotiation even after PD contract is in place. Set to 0 (default) to disable support for CDP once PD contract is in place.
CCG5_CDP_WAIT_DURATION	Specify a timeout (in seconds) period from Type-C attach at which point BC 1.2 support will be disabled. This can be set to a non-zero value for power savings in cases where a Type-C to Type-A cable is left connected to the port without a device attached.	Timeout period in seconds. 0 (default) means no timeout.
SYS_DEEPSLEEP_ENABLE	Enable/disable putting the CCG5 device into a low power mode when idle.	Can be set to 0 to save flash space where required.
VBUS_OVP_ENABLE	Enable/disable detection and handling of VBus Over-Voltage faults.	Recommend setting this to 1 in all cases.
VBUS_OVP_MODE	Select mode of OVP detection and handling.	0 ➔ Not supported. 1 ➔ OVP detection by OV comparator and handling by firmware. 2 ➔ OVP detection by OV comparator and automated hardware based handling.

Pre-processor Switch	Description	Values
VBUS_OCP_ENABLE	Enable/disable detection and handling of VBus Over-Current faults.	1 ➔ Type-C VBUS OCP enable. 0 ➔ Type-C VBUS OCP disable.
VBUS_OCP_MODE	Select mode of OVP detection and handling.	0 ➔ Use external load switch 1 ➔ Not supported 2 ➔ OCP detection by internal CSA with automatic hardware based handling. 3 ➔ OCP detection by internal CSA with firmware based debounce and handling.
VCONN_OCP_ENABLE	Enable/disable detection and handling of VConn Over-Current faults.	1 ➔ Type-C VConn OCP enable. 0 ➔ Type-C VConn OCP disable.
DFP_ALT_MODE_SUPP	Enable/disable the alternate mode discovery and handling state machine when the port managed by CCG5 is a DFP.	1 ➔ Enable alternate modes when CCG5 is DFP. 0 ➔ Disable alternate modes when CCG5 is DFP.
DP_DFP_SUPP	Enable/disable DisplayPort source (DFP_U/DFP_D) functionality.	1 ➔ Enable DP source functionality. 0 ➔ Enable DP sink functionality.
TBT_DFP_SUPP	Enable/disable Thunderbolt alternate mode operation when CCG5 is a DFP. This can only be enabled in system designs that include a Thunderbolt controller from Intel.	1 ➔ Enable TBT alternate mode. 0 ➔ Disable TBT alternate mode.
UFP_ALT_MODE_SUPP	Enable/disable the alternate mode discovery and handling state machine when the port managed by CCG5 is a UFP.	1 ➔ Enable alternate modes when CCG5 is UFP. 0 ➔ Disable alternate modes when CCG5 is UFP.
TBT_UFP_SUPP	Enable/disable Thunderbolt alternate mode operation when CCG5 is a UFP. This can only be enabled in system designs that include a Thunderbolt controller from Intel.	1 ➔ Enable TBT alternate mode. 0 ➔ Disable TBT alternate mode.
RESET_ON_ERROR_ ENABLE	Selects whether to enable / disable CCG device reset on error (watchdog expiry or hard fault)	1 ➔ Enable reset option 0 ➔ Disable reset option
STACK_USAGE_CHECK_ENABLE	Enable/disable periodic checking of available margin in the runtime stack.	1 ➔ Enable stack margin checks 0 ➔ Disable stack margin checks

In addition to the user-selectable parameters listed above, the config.h file also defines a set of parameters that configure the behavior of the PD stack and other modules in the CCG5 firmware. Changes to these parameters are not recommended for optimal operation. These parameters are described in Table 3.

Table 3: Firmware configuration definitions in config.h

Pre-processor Switch	Description	Values
CCG_PD_REV3_ ENABLE	Whether to enable PD 3.0 support or not. PD 3.0 operation is recommended. More details in section 4.4.	1 ➔ PD 3.0 support enabled 0 ➔ PD 3.0 support disabled
CCG_FRS_RX_ENABLE	Enable/disable handling of Fast Role Swap requests from a PD 3.0 power source.	1 ➔ Enable FRS receive function. 0 ➔ Disable FRS receive function.
CCG_FRS_TX_ENABLE	Enable/disable generation of Fast Role Swap request as a PD 3.0 power source.	1 ➔ Enable FRS transmit function. 0 ➔ Disable FRS transmit function
MUX_INIT_DELAY_MS	Defines the delay required for the Type-C data switch to completely turn-on. This delay is applied between the MUX enable and VBus enable steps.	Valid values are 0 – 100 ms.

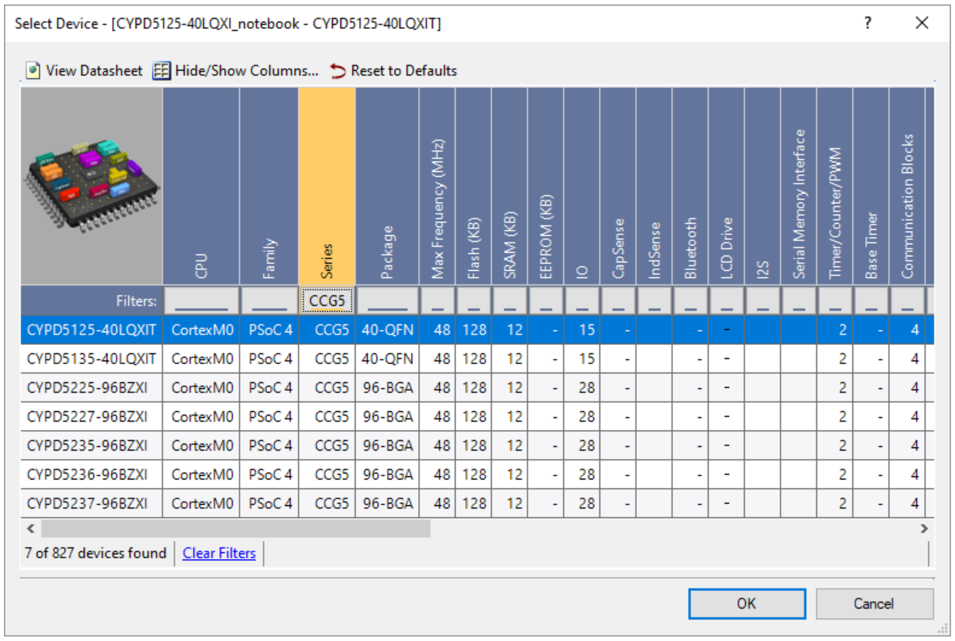
4.3 Part Number Update

The SDK workspaces are geared to work with appropriate applications. But there can be instances where the target part needs to be changed. The following are the steps to be done for changing the part.

4.3.1 Device Selector

The part number for the project in the workspace needs to be updated to match the new part. This can be achieved using the device selector. Right click the project on the Workspace Explorer window and select Device Selector. From the Device Selector dialog, change the part number. The part number selection can also be done when copying the example project as per section 3.1.1.2. If this is already done, then this step can be avoided.

Figure 15: Device Selector Dialog for changing Part Number


Select Device - [CYPD5125-40LQXI_notebook - CYPD5125-40LQXIT]
?
View Datasheet
Hide/Show Columns...
Reset to Defaults
OK
Cancel

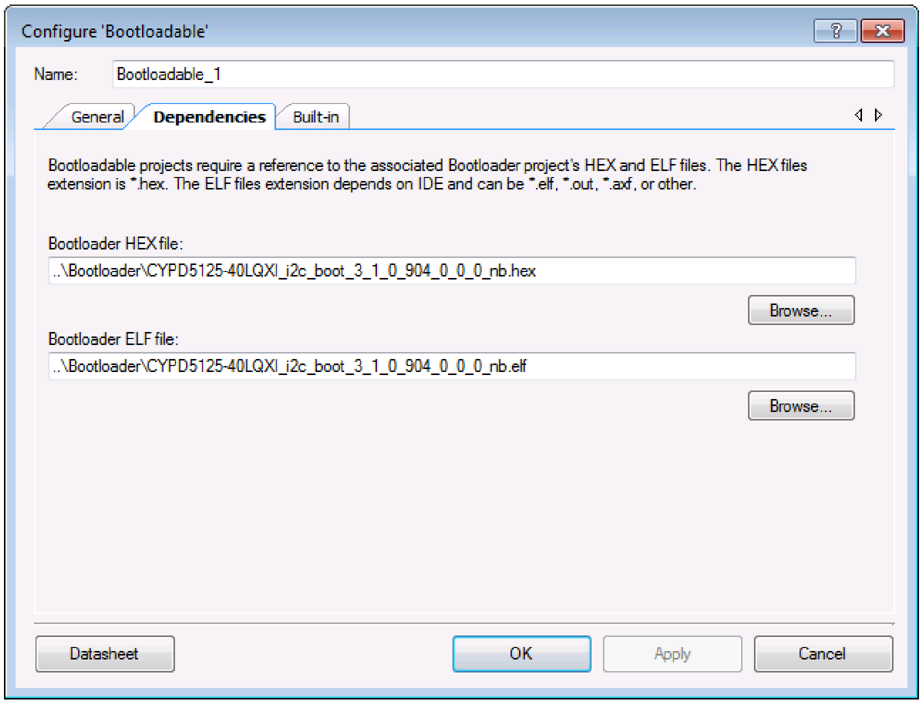
	CPU	Family	Series	Package	Max Frequency (MHz)	Flash (KB)	SRAM (KB)	EEPROM (KB)	IO	CapSense	IndSense	Bluetooth	LCD Drive	I2S	Serial Memory Interface	Timer/Counter/PWM	Base Timer	Communication Blocks
Filters:			CCG5															
CYPD5125-40LQXIT	CortexM0	PSoC 4	CCG5	40-QFN	48	128	12	-	15	-		-	-			2	-	4
CYPD5135-40LQXIT	CortexM0	PSoC 4	CCG5	40-QFN	48	128	12	-	15	-		-	-			2	-	4
CYPD5225-96BZXI	CortexM0	PSoC 4	CCG5	96-BGA	48	128	12	-	28	-		-	-			2	-	4
CYPD5227-96BZXI	CortexM0	PSoC 4	CCG5	96-BGA	48	128	12	-	28	-		-	-			2	-	4
CYPD5235-96BZXI	CortexM0	PSoC 4	CCG5	96-BGA	48	128	12	-	28	-		-	-			2	-	4
CYPD5236-96BZXI	CortexM0	PSoC 4	CCG5	96-BGA	48	128	12	-	28	-		-	-			2	-	4
CYPD5237-96BZXI	CortexM0	PSoC 4	CCG5	96-BGA	48	128	12	-	28	-		-	-			2	-	4

7 of 827 devices found | Clear Filters

4.3.2 Bootloader selection

When application part number is modified, the corresponding bootloader binaries should also be updated. In the schematics tab of the application double click the Bootloadable component. Update the bootloader binary files in the Dependencies Tab as shown in Figure 16. Choose the bootloader binary that matches the part number.

Figure 16: Bootloader binary file update option



Bootloadable projects require a reference to the associated Bootloader project's HEX and ELF files. The HEX files extension is *.hex. The ELF files extension depends on IDE and can be *.elf, *.out, *.axf, or other.

Bootloader HEX file:
..\Bootloader\CYPD5125-40LQXI_i2c_boot_3_1_0_904_0_0_0_nb.hex

Bootloader ELF file:
..\Bootloader\CYPD5125-40LQXI_i2c_boot_3_1_0_904_0_0_0_nb.elf

Since the CCG5 firmware is split across two applications (primary and backup firmware), the bootloader usage in these projects vary. The following sections describe the bootloader usage in each of these projects.

4.3.2.1 Bootloader in Backup Firmware application

The actual I2C boot-loader for the CCG5 device is used in the Backup firmware application. Fully tested pre-compiled boot-loader binaries for each of the CCG5 devices are made available in the corresponding Bootloader folders.

The ***i2c_boot.cydsn*** project which is part of the code example folder can be used to compile a new customized boot-loader where required. Figure 16 shows the bootloader binary selection option for the backup firmware project.

4.3.2.2 Bootloader in Primary Firmware Application

The primary firmware project uses a dummy boot-loader binary which is not added into the final HEX file that is generated. The post build script which runs at the end of the build process replaces the dummy boot-loader with the real boot-loader which it selects from the backup firmware binary.

A pre-compiled binary for the dummy boot-loader is provided in the Bootloader folder under each CCG5 project workspace. Since the dummy boot-loader is not really used in the final binaries, there is no need to change this binary.

4.3.3 Build and re-compile

Once the part number and bootloader binary files have been updated, the workspace is ready to be compiled and used normally. Once the part number has been updated, the firmware cannot be used correctly on the kit and requires the correct part to be used.

4.4 USB-PD Specification Revisions

The example applications provided for CCGx 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.

4.5 CYPD5125-40LQXI_notebook Application

The CCG5 (CYPD5125-40LQXI_notebook) application implements a Type-C PD port controller for desktop and notebook platforms. Table 4 summarizes the features supported by the primary and backup versions of this firmware solution.

Table 4: CCG5 Notebook firmware features

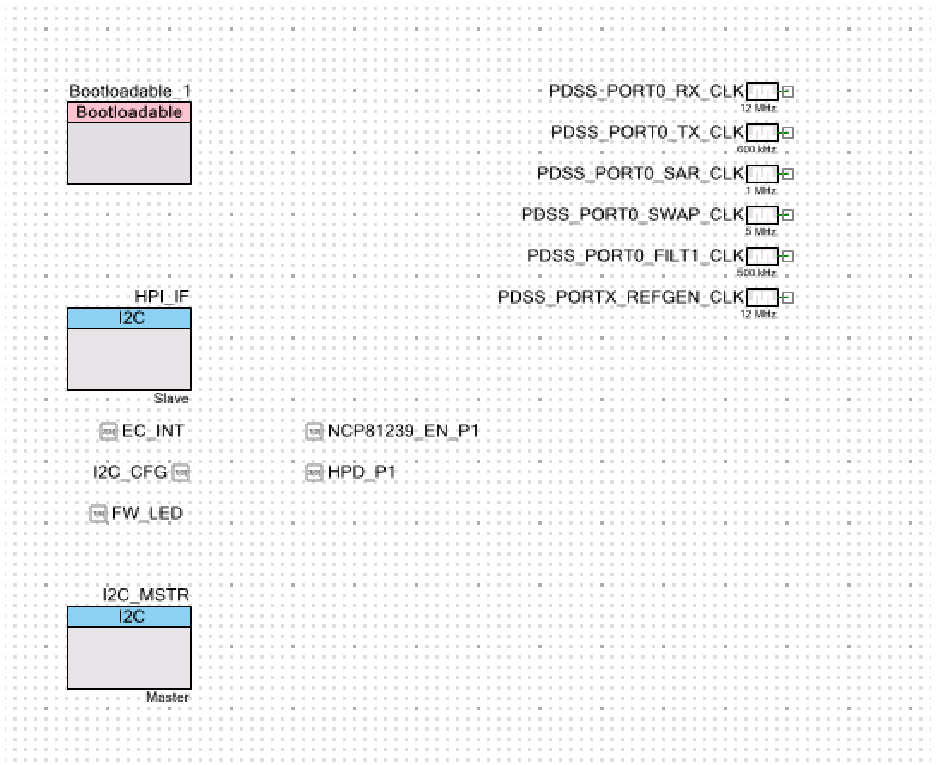
Feature	Support in Primary Firmware	Support in Backup Firmware
Dual Role Type-C v1.2 compliant port	Yes	Yes
Try.SRC configuration support	Yes	Yes
Try.SNK configuration support	Yes	Yes
USB-PD revision 2.0 support	Yes	Yes
USB-PD revision 3.0 support	Yes	No
Fast Role Swap Receive Support	Yes	No
DisplayPort source state machine	Yes	No
Thunderbolt (DFP/UFP) state machine	Yes	No
Firmware upgrade support through HPI	Yes	Yes
PD status and event reporting through HPI	Yes	Yes
PD command and VDM tunneling support through HPI	Yes	No
BC 1.2 (CDP) source support	Yes	No
VBus Over Voltage Protection	Yes	Yes
VBus Over Current Protection	Yes	Yes

VConn Over Current Protection	Yes	Yes
CC line Over-Voltage Protection	Yes	Yes
SBU line Over-Voltage Protection	Yes	Yes

The following sub-sections describe the project structure and implementation in more detail. The primary firmware implementation is described in detail, as the backup firmware is only a sub-set of the primary firmware.

4.5.1 PSoC Creator Schematic

Figure 17: PSoC Creator Schematic for CYPD5125 Notebook project



Open TopDesign.cysch file in the PSoC creator project. Schematic contains hardware resources used by power adapter such as clocks, voltage selection IOs etc. The schematic elements are split across multiple sheets, and Figure 17 shows all of the active elements together.

The selection of some of these elements is fixed due to the capabilities of the CCG5 device and the bootloader design. Table 5 points out the changes allowed in the schematic design.

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.

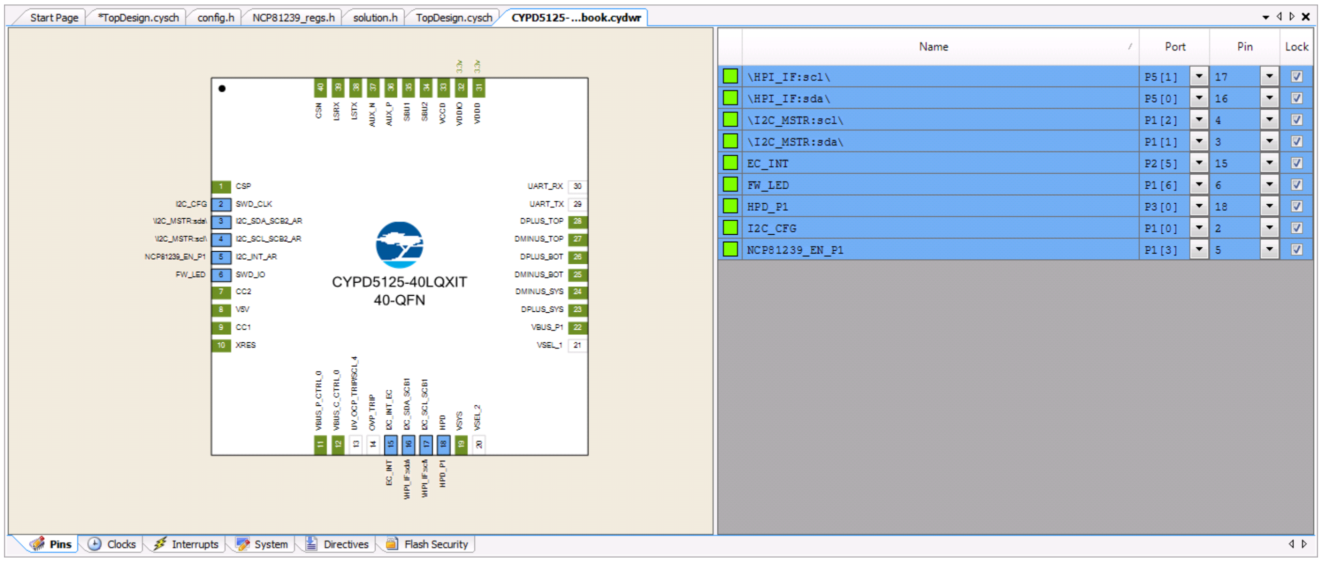
Table 5: Schematic Elements in CCG5 single-port Notebook Design

Schematic Element	Description	Changes allowed
Bootloadable_1	This is a software block which interacts with the boot-loader on the CCG5 device.	No changes are allowed.

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.
PDSS_PORT0_SWAP_CLK	This is an internal clock that is used by FR-SWAP transmit/receive part of the USB-PD block.	No changes are allowed.
PDSS_PORT0_FILT1_CLK	This is an internal clock that is used for the filters used for debouncing various comparator outputs in the USB-PD block.	No changes are allowed.
PDSS_PORTX_REFGEN_CLK	This is an internal clock that is used for the reference generator block of the USB-PD block.	No changes are allowed.
HPI_IF	This is an I^2C slave block through which the CCG5 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.
EC_INT	Output interrupt signal from CCG5 to the Embedded Controller.	Can be changed only in cases where the EC does not require an interrupt and will poll CCG5 for interrupt notifications.
I2C_CFG	Control signal used to define the I_2C slave address used in the HPI interface.	No changes are allowed as the HPI_IF is also used by the boot-loader which is fixed.
FW_LED	GPIO used to toggle an LED periodically to indicate firmware operation.	Can be changed or removed. The APP_FW_LED_ENABLE definition should be set to 0 if this function is being removed.
I2C_MSTR	I^2C master block used by the CCG5 to control the buck-boost regulator and data switch devices.	Can be changed/removed as required.
NCP81239_EN_P1	Output signal used to enable the on-board buck-boost regulator.	Can be changed/removed as required.
HPD_P1	DisplayPort HotPlug Detect signal output.	Can be removed if DP source function is not required. Changes are not 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 CCG5 device. Open the *CYPD5125-40LQXI_notebook.cydwr* file to see the DWR settings for the project.

Figure 18: Design Wide Resource (DWR) View



	Name	Port	Pin	Lock
🟩	\HPI_IF:scl\	P5[1]	17	✓
🟩	\HPI_IF:sda\	P5[0]	16	✓
🟩	\I2C_MSTR:scl\	P1[2]	4	✓
🟩	\I2C_MSTR:sda\	P1[1]	3	✓
🟩	EC_INT	P2[5]	15	✓
🟩	FW_LED	P1[6]	6	✓
🟩	HPD_P1	P3[0]	18	✓
🟩	I2C_CFG	P1[0]	2	✓
🟩	NCP81239_EN_P1	P1[3]	5	✓

As shown in Figure 18, 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 5.

4.6 *CYPD5225-96BZXI_notebook Application*

The dual-port CCG5 notebook application is very similar to the single-port application. The supported features are the same and are summarized in Table 4. The only changes between the projects are additional elements in the schematic design, compile-time configuration changes and changes to the PD stack and HPI libraries.

4.6.1 PSoC Creator Schematic

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 19 shows the various schematic elements used in the CCG5 dual-port notebook solution. The elements are almost the same as those listed in Table 5 with a few additions. The additions specific to the dual-port project are described in Table 6.

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.

Figure 19: PSoC Creator Schematic for CCG5 dual-port notebook application

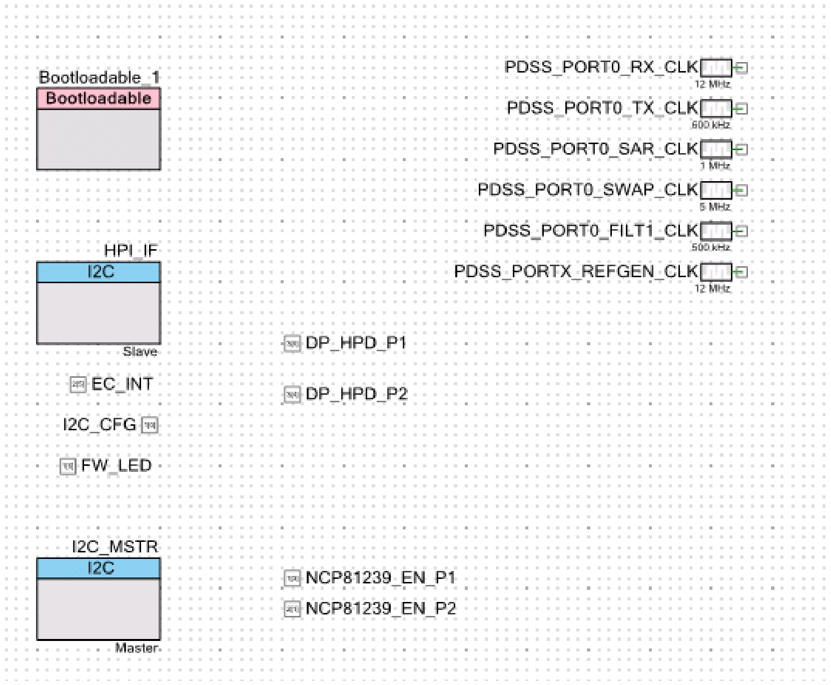


Table 6: Additional Schematic Elements in CCG5 dual-port notebook solution

Schematic Element	Description	Changes allowed
NCP81239_EN_P2	Output signal used to enable the on-board buck-boost regulator for the second PD port.	Can be changed/removed as required.
HPD_P2	DisplayPort HotPlug Detect signal output for the second PD port.	Can be removed if DP source function is not required. Changes are not allowed.

4.7 *CYPD4126-24LQXI_notebook application*

This reference project is a single-port notebook PD port controller application using the CYPD4126-24LQXI part from the CCG4 family. The functionality is like that of the CCG5 notebook applications with the following exceptions:

1. GPIOs are used for gate control of the Provider and Consumer FETs. An external FET will be required to generate the actual high voltage gate control signal.
2. GPIO based source voltage selection (5V, 9V, 15V and 20V) is used instead of I2C based regulator control.
3. The on-board ADC is used to implement the Over-Voltage detection and protection feature. Also, there is no capability to automatically disable power paths by the hardware on fault detection. Firmware intervention is required which means that latency of fault handling will be higher.
4. Over-Current detection is not supported by the CCG4 device. So, the application relies on a fault indication from external load switch to detect over-current (overload) faults.
5. There is no support for legacy charging (BC 1.2) protocols.

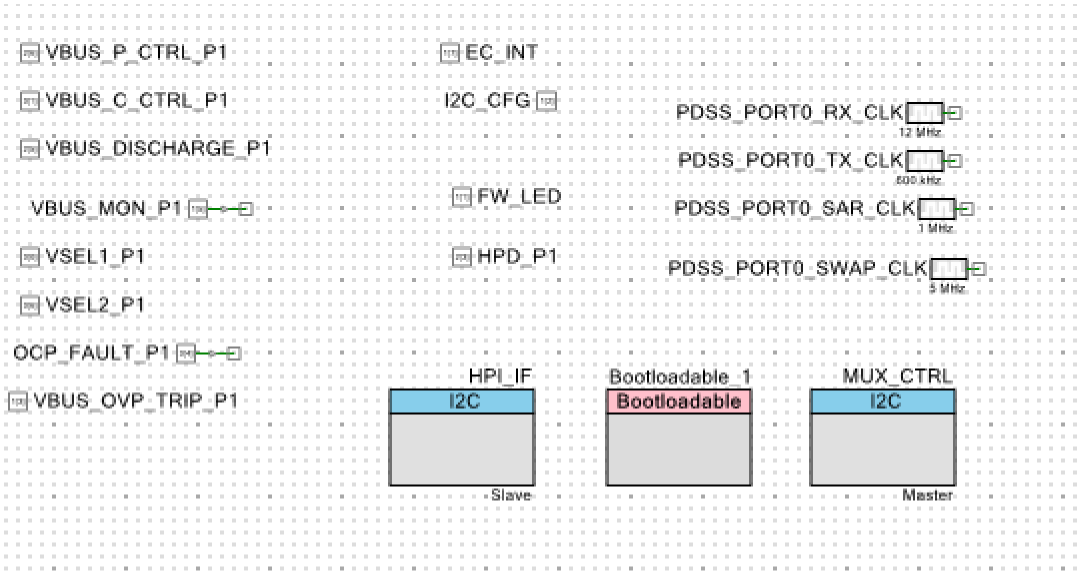
Table 7 shows the features supported by each of the CCG4 notebook applications.

Table 7: CCG4 Notebook Application Features

Feature	Supported
Dual Role Type-C v1.2 compliant port	Yes
Try.SRC configuration support	Yes
Try.SNK configuration support	Yes
USB-PD revision 2.0 support	Yes
USB-PD revision 3.0 support	Yes
Fast Role Swap Receive Support	Yes
DisplayPort source state machine	Yes
Thunderbolt (DFP/UFP) state machine	No
Firmware upgrade support through HPI	Yes
PD status and event reporting through HPI	Yes
PD command and VDM tunneling support through HPI	Yes
BC 1.2 (CDP) source support	No
VBus Over Voltage Protection	Yes
VBus Over Current Protection	No – Can be implemented using external load switch.
VConn Over Current Protection	No

4.7.1 PSoC Creator Schematic

Figure 20: PSoC Creator Schematic for CYPD4126-24LQXI_notebook project



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 20).

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 8 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.

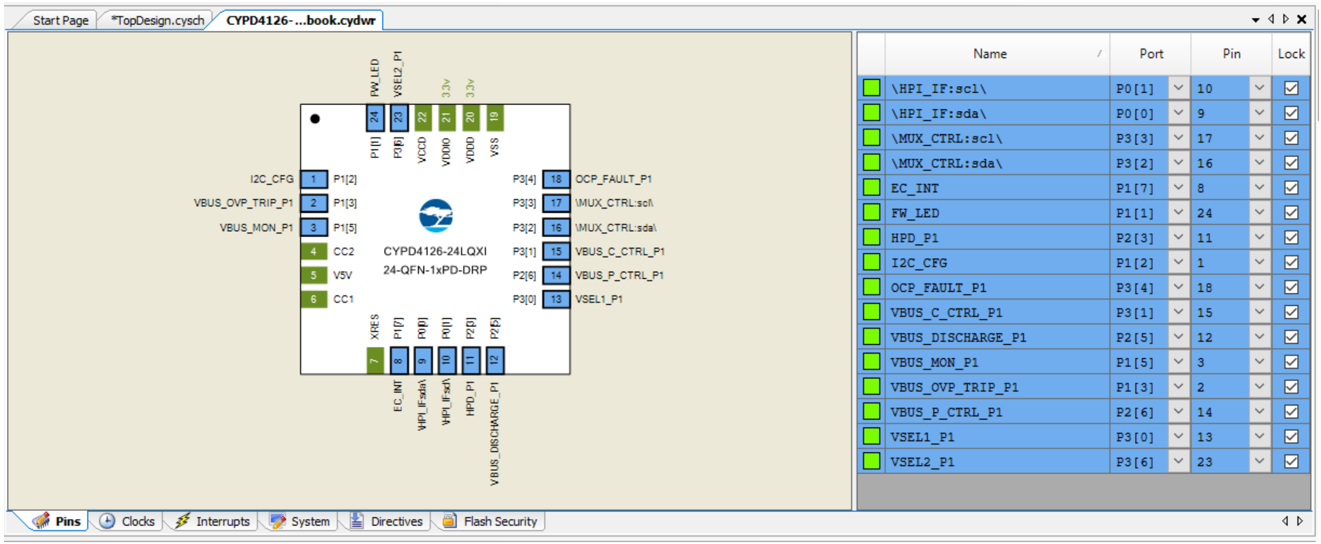
Table 8: Schematic Elements in CCG4 Notebook Design

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_PORT0_SWAP_CLK	This is an internal clock that is used by the Fast Role Swap detect logic to time the incoming Fast Role Swap request.	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	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.
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.

Schematic Element	Description	Changes Allowed
VBUS_P_CTRL_P1 VBUS_C_CTRL_P1	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	Output pin 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	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	Input pin that notifies CCG4 that an overcurrent condition has been detected.	This 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	Output pin from CCG4 that are used for a fast turn-off of the VBus supply in case of overvoltage.	This 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 ***CYPD4126-24LQXI_notebook.cydwr*** file to see the DWR settings for the project.

Figure 21. DWR Project Settings for CYPD4126-24LQXI_notebook



	Name	Port	Pin	Lock
🟩	\HPI_IF:scl\	P0[1]	10	✓
🟩	\HPI_IF:sda\	P0[0]	9	✓
🟩	\MUX_CTRL:scl\	P3[3]	17	✓
🟩	\MUX_CTRL:sda\	P3[2]	16	✓
🟩	EC_INT	P1[7]	8	✓
🟩	FW_LED	P1[1]	24	✓
🟩	HPD_P1	P2[3]	11	✓
🟩	I2C_CFG	P1[2]	1	✓
🟩	OCP_FAULT_P1	P3[4]	18	✓
🟩	VBUS_C_CTRL_P1	P3[1]	15	✓
🟩	VBUS_DISCHARGE_P1	P2[5]	12	✓
🟩	VBUS_MON_P1	P1[5]	3	✓
🟩	VBUS_OVP_TRIP_P1	P1[3]	2	✓
🟩	VBUS_P_CTRL_P1	P2[6]	14	✓
🟩	VSEL1_P1	P3[0]	13	✓
🟩	VSEL2_P1	P3[6]	23	✓

As shown in Figure 21, 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 8.

4.7.2 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. ***CYPD4126-24LQXI_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 CCG4 evaluation boards. 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.

4.7.2.1 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 9.

Table 9: 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

Option	Description	Values
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

4.7.2.2 Source Voltage Selection

Refer to the **APP_VBUS_SET_XX_P1** macros in the ***CYPD4126-24LQXI_notebook.cydsn/config.h*** file to implement the source voltage selection scheme.

On the CCG4 eval boards, 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 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.

4.7.2.3 FET Control

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

- **APP_VBUS_SRC_FET_ON_P1** – Turn provider FET ON
- **APP_VBUS_SRC_FET_OFF_P1** – Turn provider FET OFF
- **APP_VBUS_SNK_FET_ON_P1** – Turn consumer FET ON
- **APP_VBUS_SNK_FET_OFF_P1** – Turn consumer FET OFF
- **APP_DISCHARGE_FET_ON_P1** – Turn VBus Discharge FET ON
- **APP_DISCHARGE_FET_OFF_P1** – Turn VBus Discharge FET OFF

4.7.2.4 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^{2}C 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.

4.7.2.5 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.

4.8 CYPD4126-40LQXI_notebook application

This reference project is a single-port notebook PD port controller application using the CYPD4126-40LQXI part from the CCG4 family. The functionality is like that of the CYPD4126-40LQXI notebook application with only one change:

1. The CYPD4126-40LQXI has the capability to automatically control the GPIOs used to turn the provider and consumer FETs on/off. For this reason, the **`VBUS_FET_INTERNAL_CTRL`** parameter is set to 1 in this project.

For the project schematic, DWR settings, compile time settings and solution level code changes; refer to the CYPD4126-24LQXI_notebook project.

4.9 CYPD4226-40LQXI_notebook application

This reference project is a dual-port notebook PD port controller application using the CYPD4226-40LQXI part from the CCG4 family. The functionality supported on each port is identical to that supported in the CYPD4126-40LQXI_notebook project.

4.9.1 PSoC Creator Schematic

Figure 22 shows the schematic associated with the CYPD4226-40LQXI_notebook project. The schematic elements are similar to those in the CYPD4126-24LQXI_notebook project; with the port specific elements being replicated for the second PD port.

Table 10 shows the additional schematic elements in the CYPD4226-40LQXI_notebook project as compared to the CYPD4126-24LQXI_notebook and CYPD4126-40LQXI_notebook projects.

Figure 22: PSoC Creator Schematic for CYPD4226-24LQXI_notebook project

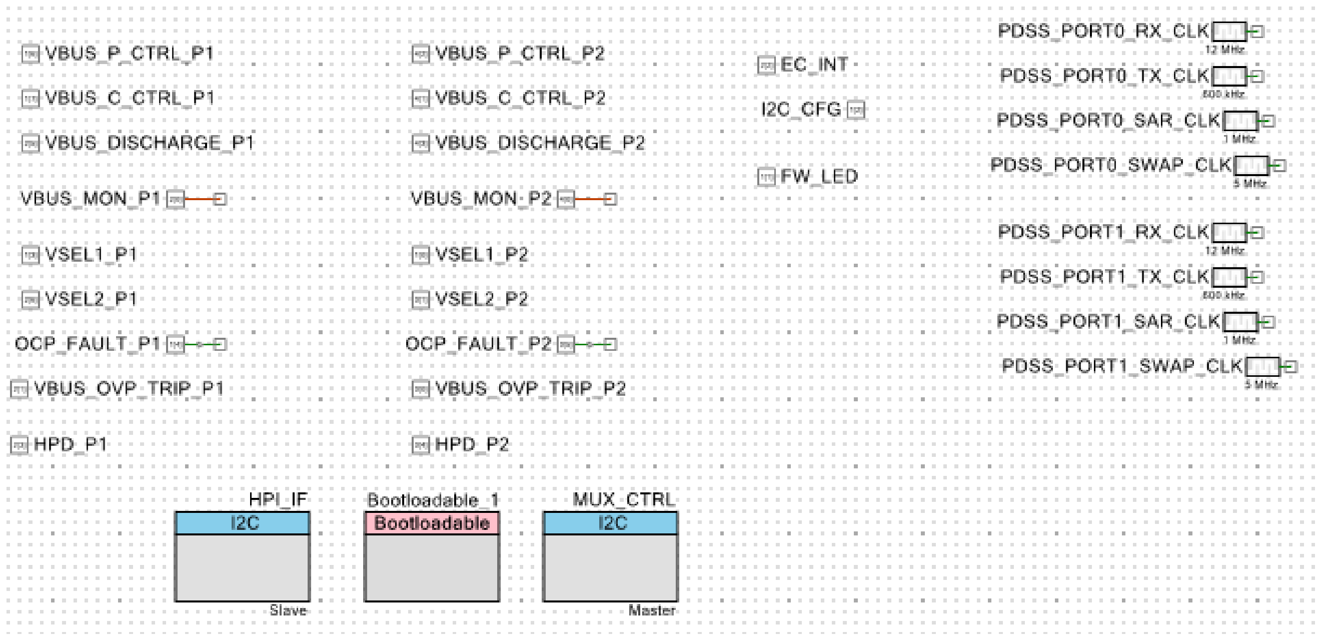


Table 10: Additional Schematic Elements in dual-port CCG4 Notebook Design

Schematic Element	Description	Changes 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.
PDSS_PORT1_SWAP_CLK	This is an internal clock that is used by the Fast Role Swap detect logic to time the incoming Fast Role Swap request.	No changes are allowed.
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.
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_P2 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.

Schematic Element	Description	Changes Allowed
VBUS_DISCHARGE_P2	Output pin 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_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_P2	Input pin that notifies CCG4 that an overcurrent condition has been detected.	This 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_P2	Output pin from CCG4 that are used for a fast turn-off of the VBus supply in case of overvoltage.	This 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.

4.9.2 Updating Code to Match the Schematic

Refer to Section 4.7.2 for details on how to modify the reference project code to match your design and schematic changes.

4.10 CYPD3125-40LQXI_notebook application

This reference project implements a single port notebook PD port controller using the CYPD3125-40LQXI device from the CCG3 family. Table 11: CCG3 Notebook Application FeaturesTable 11 summarizes the features supported by the CCG3 notebook application.

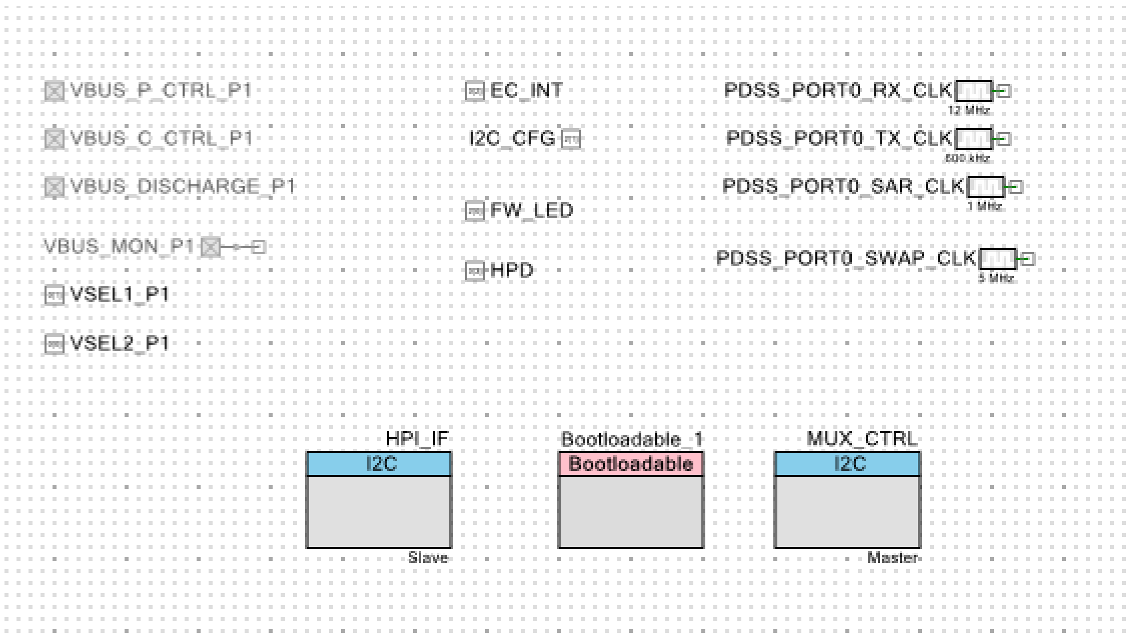
Table 11: CCG3 Notebook Application Features

Feature	Supported
Dual Role Type-C v1.2 compliant port	Yes
Try.SRC configuration support	Yes
Try.SNK configuration support	Yes
USB-PD revision 2.0 support	Yes
USB-PD revision 3.0 support	Yes
Fast Role Swap Receive Support	Yes
DisplayPort source state machine	Yes
Thunderbolt (DFP/UFP) state machine	No
Firmware upgrade support through HPI	Yes
PD status and event reporting through HPI	Yes

PD command and VDM tunneling support through HPI	Yes
BC 1.2 (CDP) source support	No
VBus Over Voltage Protection	Yes
VBus Over Current Protection	Yes
VConn Over Current Protection	No

4.10.1 PSoC Creator Schematic

Figure 23: 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.cysc*h file, which is part of each Creator project. Double-click on this file to open the schematic editor window (see Figure 23).

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 12 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.

Note: The VBUS_P_CTRL_P1, VBUS_C_CTRL_P1, VBUS_DISCHARGE_PI and VBUS_MON_P1 elements are shown greyed out because we use the dedicated hardware features of the CCG3 device for gate driver control, VBus discharge control and VBus measurement. Hence, the corresponding I/Os do not need to be instantiated in the project.

Table 12: 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.
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 by the Fast Role Swap detect logic to time the incoming Fast Role Swap request.	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.

4.10.2 Updating Code to Match the Schematic

Table 13 shows the compile-time selectable features supported by this application.

Refer to Section 4.7.2 for details on modifying the solution level code to match any hardware changes.

Table 13: 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_MODE	Select the OVP handling mechanism.	0 is reserved value 1 for firmware based FET turn-off on OV detection. 2 for hardware based FET turn-off on OV detection.
VBUS_OCP_ENABLE	Enable OverCurrent Protection on VBus supply when CCG3 is acting as the power source. 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	Select handling mechanism for Over-Current faults detected by the firmware.	2 for hardware based FET turn-off on OC detection. 3 for firmware based debounce and FET turn-off on OC detection.
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

4.11 *CYPD2122_24LQXI_notebook application*

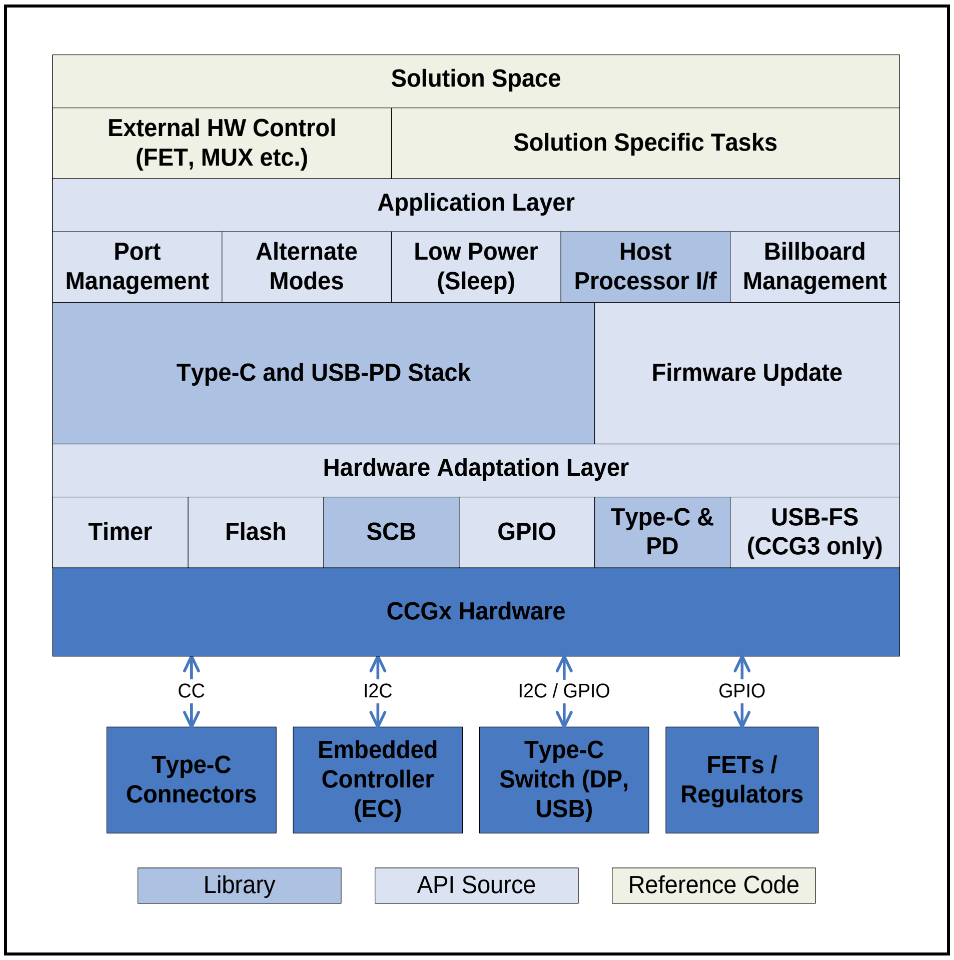
Please refer to the ***Notebook Reference Design and API Guide*** document in the ***CCG2/Documentation*** folder for details of this reference application.

5. Firmware Architecture

5.1 Firmware Blocks

The CCGx firmware architecture allows users to implement a variety of USB-PD applications using the CCG devices and a fully tested firmware stack. A block diagram of the CCGx firmware architecture is shown in Figure 24.

Figure 24: 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 and timer module.
- 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 Power Adapter and Power Bank PD port controller applications, the firmware update will be done from the port partner side through the CC interface.
- 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. This interface is not applicable for Power Adapters and Power Banks.
- 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.

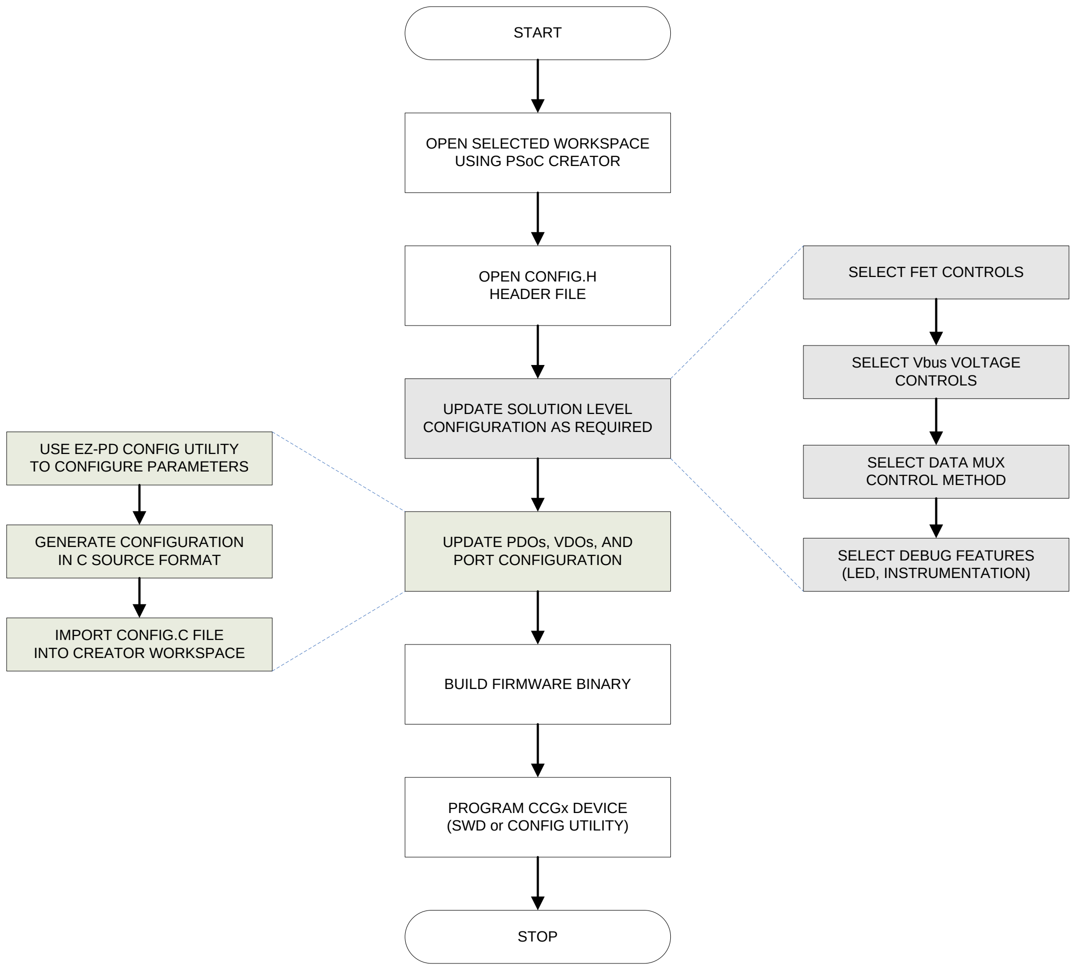
5.2 SDK Usage Model

Users of the CCG 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 25. 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 25. SDK Usage Flow


START
OPEN SELECTED WORKSPACE USING PSoC CREATOR
OPEN CONFIG.H HEADER FILE
UPDATE SOLUTION LEVEL CONFIGURATION AS REQUIRED
SELECT FET CONTROLS
SELECT Vbus VOLTAGE CONTROLS
SELECT DATA MUX CONTROL METHOD
SELECT DEBUG FEATURES (LED, INSTRUMENTATION)
UPDATE PDOs, VDOs, AND PORT CONFIGURATION
USE EZ-PD CONFIG UTILITY TO CONFIGURE PARAMETERS
GENERATE CONFIGURATION IN C SOURCE FORMAT
IMPORT CONFIG.C FILE INTO CREATOR WORKSPACE
BUILD FIRMWARE BINARY
PROGRAM CCGx DEVICE (SWD or CONFIG UTILITY)
STOP

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 firmware upgrade interface. The following table denotes the version structure and format.

Table 14: CCGx Firmware Version Structure

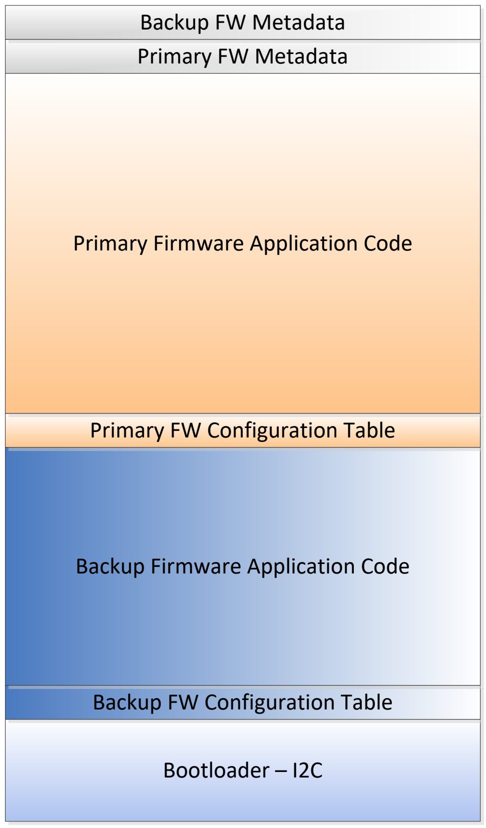
Bit Field	Name	Description
[15:0]	Base FW Build number	This field corresponds to base firmware version and shall be automatically incremented during nightly build. This field should not be manually edited. This field is expected to be reset on every SNPP release cycle and not modified throughout the 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” Power Bank: “pb” 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*

5.4.1 CCG5 Flash Memory Map

CCG5 has 128-KB flash memory that is designated to store a bootloader along with the primary and backup firmware applications. Each of the primary and backup firmware binaries are associated with corresponding configuration table and metadata. The flash memory map for the device is shown in Figure 26.

Figure 26: CCG5 Flash Memory Map


Backup FW Metadata
Primary FW Metadata
Primary Firmware Application Code
Primary FW Configuration Table
Backup Firmware Application Code
Backup FW Configuration Table
Bootloader – I2C

The bootloader is used to update firmware and configuration over the I2C-interface. It is allocated a fixed area. This memory area can only be written to from the SWD interface. CCG5 bootloader uses 5KB of memory.

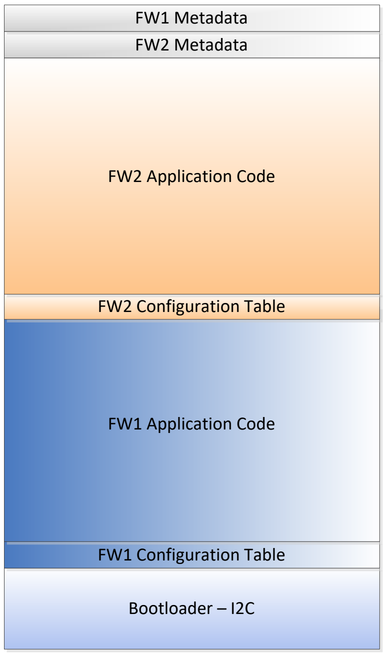
The configuration table holds the default PD configuration for the CCG application and is located at a fixed offset from start of the firmware binary. The size of configuration table for each application is 1 KB.

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

5.4.2 CCG3/CCG4 Flash Memory Map

CCG3 and CCG4 devices have 128-KB flash memory that is designated to store a bootloader along two similar copies of the notebook application. Each copy of the binary is associated with corresponding configuration table and metadata. The flash memory map for the device is shown in Figure 26.

Figure 27: CCG3/CCG4 Flash Memory Map



The bootloader is used to update firmware and configuration over the I2C-interface. It is allocated a fixed area. This memory area can only be written to from the SWD interface. CCG4 bootloader uses 5KB of memory and CCG3 bootloader uses 6KB of memory.

The configuration table holds the default PD configuration for the CCG application and is located at a fixed offset from start of the firmware binary. The size of configuration table for each application is 1 KB.

The metadata area holds metadata about the firmware binary. 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.

In the case of the CCG5 device, the primary firmware always has higher priority and will be loaded if present. The bootloader only loads the backup firmware if the primary has been corrupted or when there is an explicit request to load the backup.

In the case of CCG3 and CCG4 devices, both firmware binaries are equivalent. Hence, the bootloader keeps track of the most recently updated binary (by looking the metadata row that was written last) and loads it when valid.

Figure 28 shows the bootloader flow diagram for the CCG5 device. Figure 29 shows the bootloader flow diagram for CCG3 and CCG4 devices.

Figure 28: CCG5 Bootloader Flow Diagram

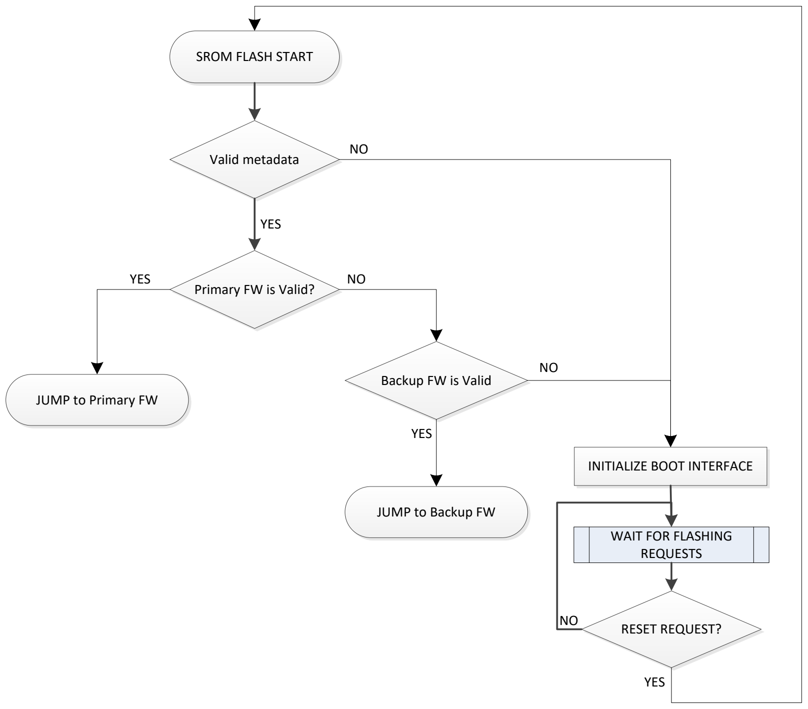
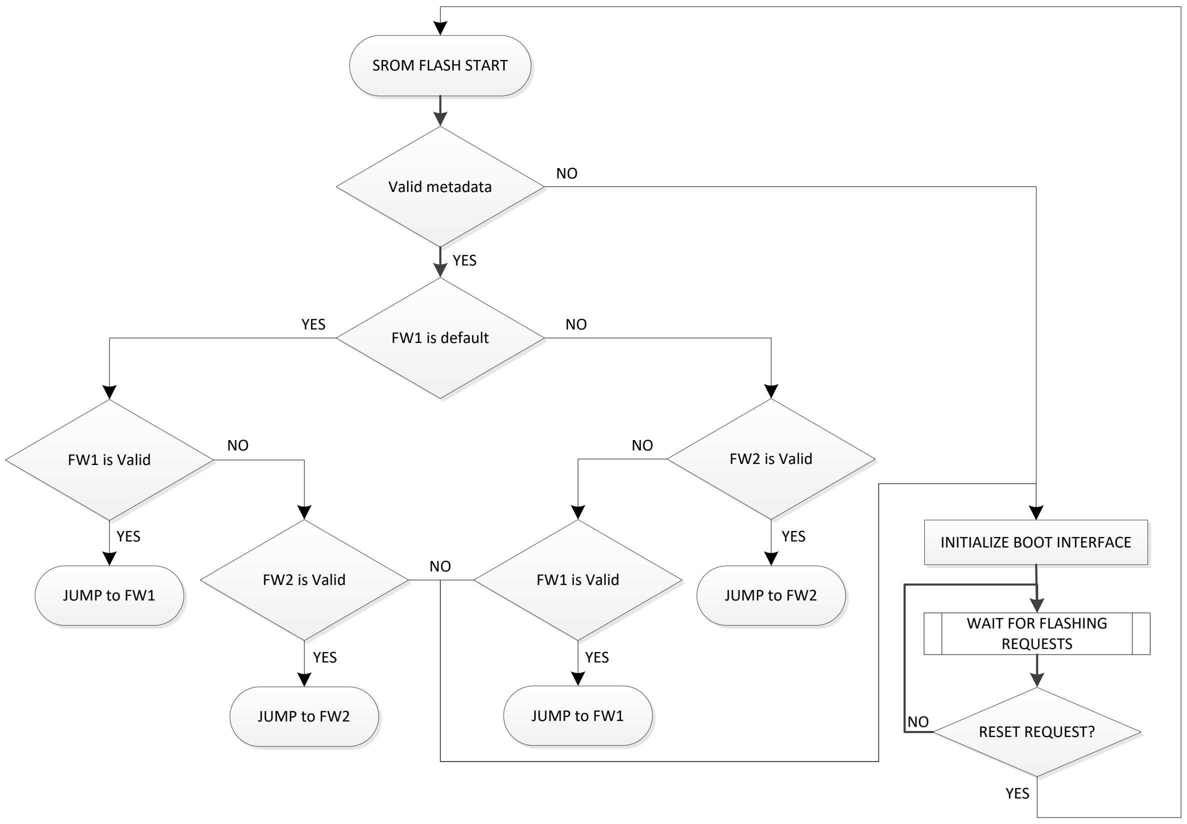


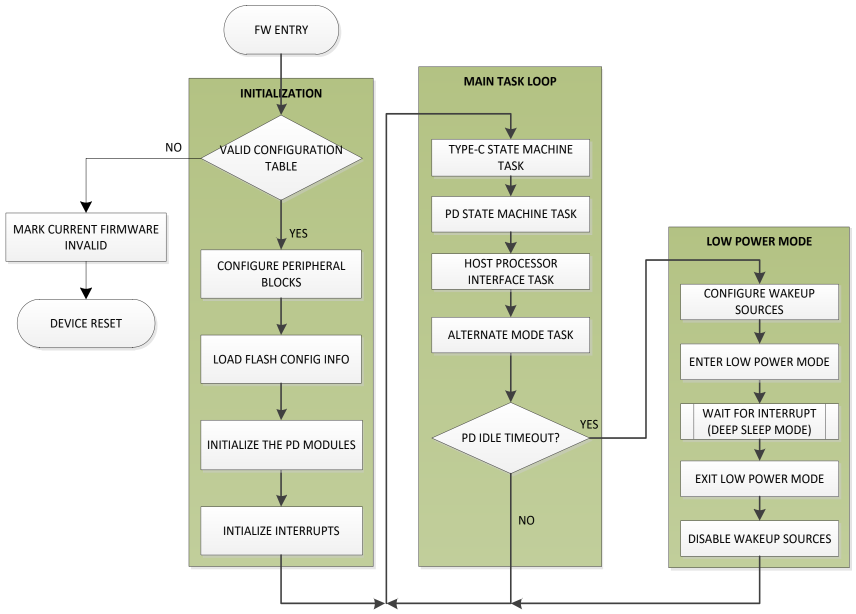
Figure 29: CCG3/CCG4 Bootloader Flow Diagram



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.

5.6.1 Fault Handling

Each CCGx device family supports different forms of fault detection and handling capabilities. Table 15 summarizes the various kinds of fault detection and handling supported in various applications in the SDK.

Table 15: Summary of fault handling features in notebook applications

Type of fault	CCG4 Support	CCG3 Support	CCG5 Support	Comments
VBus Over-Voltage	Use external resistor divider and ADC for detection. Firmware based handling with ~50us latency (typical).	Internal resistor divider and dedicated OV comparator. Hardware can turn FET off on fault with programmable debounce.		Hard Reset and recovery will be attempted till a configurable number of retries. Firmware suspends the port and waits for physical disconnection after all retries have elapsed.
VBus Over-Current	No on-device support. Firmware supports interrupt input from external load switch. Firmware based handling with debounce in ms units.	High-side current sensing across a sense resistor. Firmware based FET turn-off after debounce in ms units.		

Type of fault	CCG4 Support	CCG3 Support	CCG5 Support	Comments
VConn Over-Current	Not supported	Not supported	Hardware based detection and automatic VConn switch disable.	Firmware exits any alternate modes which require VConn to be present. VConn will not be re-enabled until there is a new Type-C connection.
CC line Over-Voltage	Not supported	Not supported	Hardware based detection and CC line disable.	Firmware suspends the port and waits for physical disconnection.
SBU line Over-Voltage	Not supported	Not supported	Hardware based detection and SBU MUX disconnection.	

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 APIs.

6.1 Data Structures

Table 16 lists the important data structures used by the APIs.

Table 16: List of important data structures

Data structure	Description
dpm_pd_cmd_buf_t	Data structure holding the command parameters for DPM commands. Refer to section 6.3.6.3 for usage.
dpm_status_t	The data structure holds the status information for the device policy manager. This data is retrieved by dm_get_info function. The application / solution layer can retrieve the DPM status using this. The data should not be modified outside of DPM. Refer to section 6.3.6.4 for usage.
pd_do_t	Union to hold a PD data object. All USB-PD data objects are 4-byte values which are interpreted according to the message type, length and object position. This union represents all possible interpretations of a USB-PD data object. Refer to the USB-PD specification for details on each field.
pd_config_t	The data structure holds the device configuration data as stored by the Ez-PD Configuration Utility. These parameters are located at fixed offset from start of the firmware so that the device can be configured without having to recompile and modify the firmware binary. The structure consists of header fields holding header information as well as checksum followed by port specific configuration data.
pd_port_config_t	The data structure holds the port specific configuration data stored as part of pd_config_t structure. Refer to the API Reference Guide for more details.
ovp_settings_t	The data structure holds the over voltage protection settings selected via configuration utility. This allows the selection on fault thresholds, debounce period and retry count a. Refer to the API Reference Guide for more details.
ocp_settings_t	The data structure holds the over voltage protection settings selected via configuration utility. This allows the selection on fault thresholds, debounce period and retry count a. Refer to the API Reference Guide for more details.
app_cbk_t	This is a function pointer array for all PD stack handlers. The structure allows to override / customize handling for various PD stack controls from the solution space.

	Refer to Section 6.3.7.2 for usage.
app_resp_t	The data structure holds the response information for callbacks from the PD stack.
app_status_t	Various state machine parameters stored for rereference from the generic application implementation for the DPM handlers registered via app_cbk_t.

6.2 API Summary

6.2.1 Device Policy Manager (DPM) API

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

Table 17: 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.
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_def_cable_cap	Update the default cable current characteristics. The default spec limit is 3A. But in captive cable designs, this can be overridden to match actual design.	def_cur: New default current setting in 10mA units.	None.
dpm_get_def_cable_cap	Get the current default cable capabilities current setting.	None	Default cable current setting in 10mA units.

Function	Description	Parameters	Return
dpm_update_snk_wait_cap_period	Update the sink wait for source capabilities period (tTypeCSnkWaitCap)	period: Wait period in ms.	None
dpm_get_snk_wait_cap_period	Get the current tTypeCSnkWaitCap setting for the stack.	None	Wait period in ms.
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_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_is_rdo_valid	Generic stack implementation to verify an RDO with the current source capabilities.	port: Port to be checked. rdo: The RDO to be verified.	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_get_pd_port_status	Get the current USB-PD port status.	port: Port to be queried	32bit port status.
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

Function	Description	Parameters	Return
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
dpm_prot_reset_rx	Resets protocol layer TX and RX counter for a specific sop type.	port: Port to be updated sop: SOP type to do the reset.	Call status
dpm_prot_reset_rx	Resets protocol layer RX only counter for a specific sop type.	port: Port to be updated sop: SOP type to do the reset.	Call status
dpm_set_alert	Sets alert ADO on fault.	port: Port to be updated alert_ado: The ADO object to be updated	Call status
dpm_clear_hard_reset_count	Clears the hard reset count.	port: Port to be updated	Call status
dpm_set_fault_active	Indicate that there is a fault condition active.	port: Port to be updated	Call status
dpm_clear_fault_active	Clear all fault conditions	port: Port to be updated	Call status
dpm_refresh_src_cap	Regenerate the source capabilities from the available PDOs and PDO mask. Also update current setting based on the cable characteristics.	port: Port to be updated	0 – If failed 1 – If successful
dpm_refresh_snk_cap	Regenerate the sink capabilities from the available PDOs and PDO mask.	port: Port to be updated	0 – If failed 1 – If successful
dpm_update_mux_enable_wait_period	Update the delay to be used between MUX enable and VBus enable.	period: Delay in ms.	None

6.2.2 Application Layer API

Table 18 lists the application layer APIs provided by the CCGx Host SDK. These function declarations and definitions can be found under the ***src/app*** folder.

Table 18: 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

Function	Description	Parameters	Return
app_get_status	Get the current application status information.	port: Port to be queried.	Pointer to application status structure.
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

Table 19 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 19: 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.

6.2.3 Alternate Mode API

This section documents the alternate mode related API provided in the CCGx Host SDK. These APIs are defined in the sources under ***src/app/alt_mode*** and are summarized in Table 20.

Table 20: 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.

Function	Description	Parameters	Return
vdm_task_mngr	Alt. mode manager state machine task. This is called from app_task.	port: Port to be serviced.	None
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.

6.2.4 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.

6.2.4.1 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 21.

Table 21: 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.
gpio_clear_intr	Clear any interrupts associated with the given pin.	port_pin: CCG pin identifier.	None

6.2.4.2 I2C API

The Serial Communication Block component in PSoC Creator can be used with the CCG device. However, the SDK provides a dedicated I^{2}C slave mode driver, which is optimized for HPI implementation. These API definitions are provided in ***src/scb/i2c.c*** and are summarized in Table 22.

Table 22: 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_deinit	De-initialize the specified I2C slave block.	scb_index: SCB index to be used.	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

6.2.4.3 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 23.

Table 23: 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 data_in_place: Specifies whether the data to be written to flash should be used in place, or a copy should be made on stack.	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.

6.2.4.4 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 24.

Table 24: 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

6.2.5 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 25.

Table 25: 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	None	Current firmware mode.

Function	Description	Parameters	Return
	device.		

6.3 API Usage Examples

This section provides a few examples for the usage of the APIs documented under Section 6.2. Refer 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:

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

6.3.1 Boot API Usage

The bootloader and firmware application communication in the CCGx Host 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.

6.3.1.1 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 ();
}
```

6.3.1.2 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();
}
```

6.3.1.3 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 6.2.5 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();
}
```

6.3.2 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 Host SDK. The list of GPIO APIs is provided in section 6.2.4.1.

6.3.2.1 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);
}
```

6.3.2.2 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);
}
```

6.3.3 Timer API Usage

The CCGx Host 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);
}
```

6.3.4 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);
}
```

6.3.5 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.

6.3.6 DPM API Usage

6.3.6.1 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);
}
```

6.3.6.2 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;
                }
        }
}
```

6.3.6.3 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;
    }
```

6.3.6.4 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.

6.3.6.5 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 6.3.6.4, 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);
        }
}
```

6.3.6.6 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.

6.3.7 Solution Level Examples

6.3.7.1 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);
}
```

6.3.7.2 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. */
    eval_fr_swap,           /* Evaluate received FR swap */
    vbus_get_value          /* Retrieve the VBUS voltage */
};
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));
}
```

6.3.7.3 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 6.3.7.2).

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.

6.3.8 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.

6.3.8.1 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.

6.3.8.1.1 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 26: 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.

6.3.8.1.2 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.

6.3.8.1.2.1 DFP Handling

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

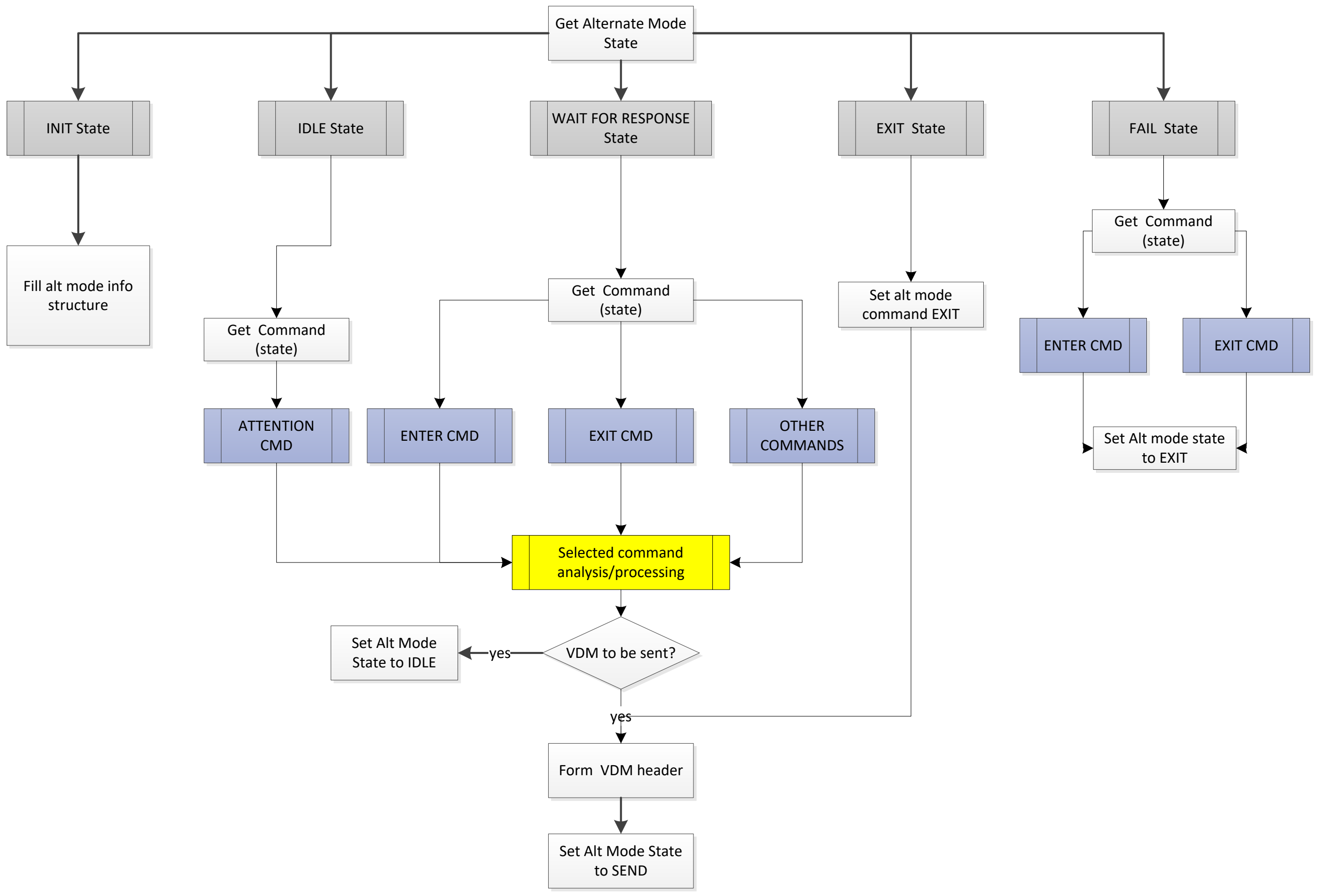
1. The INIT state is used to initiate alternate mode handling. This state uses in two cases:
   a. When DFP alternate mode registrations is successful and we need to initiate alternate mode handling.
   b. When Alt modes manager initiates asynchronous entry of the alternate mode.
   c. During initialization 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



6.3.8.1.2.2 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:

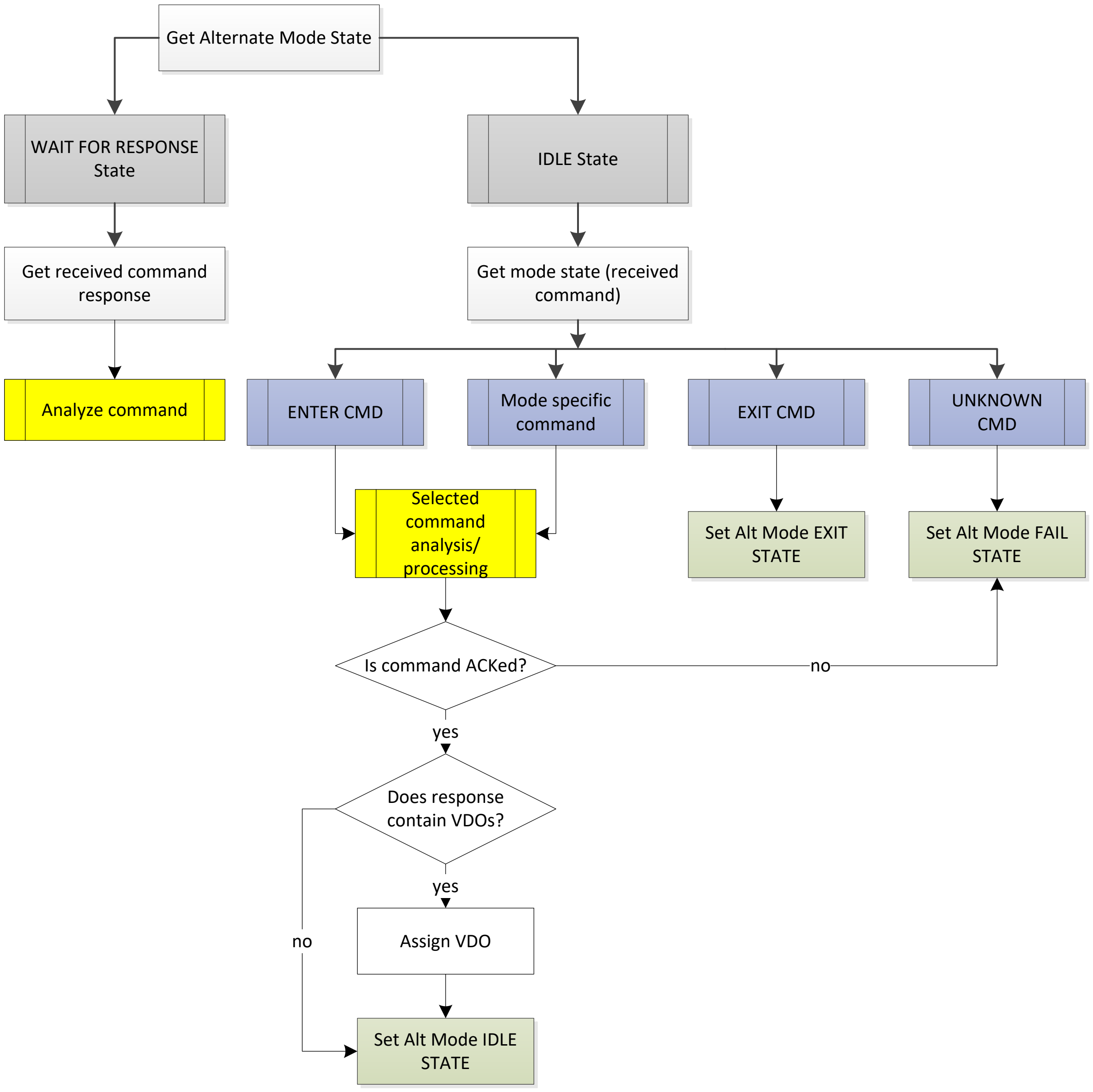
   a. Get internal alternate mode depending on command from VDM header

   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



6.3.8.2 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.

7. Revision History

7.1 Document Revision History

Document Title: **Cypress EZ-PD™ CCGx Host SDK User Guide** Document Number: 002-24327			
Revision	**Issue Date**	**Origin of Change**	**Description of Change**
**	29/June/2018	KYS	Initial release