PMG1 Duino User Manual

About this document

This document describes the generic information of PMG1 Duino framework.

Table of contents

1 Overview of PMG1 Duino

This PMG1 Duino framework provides a robust and ease-to-use platform based on Arduino environment that offers the following targets to the users.

(1) To provide an example of working and pre-configured framework based on the cost-effective and/or existing PMG1 kits, with the correct files included, essential tool chain and the correct compiler option set.

(2) To allow 'out-of-box' experimentation with minimum setup or prior knowledge, e.g. USB PD.

(3) As a demonstration in the example codes of how users can be used and applied to their applications.

(4) As a base from which real application can be created.

There are 2 kits supporting in this PMG1 Duino framework and the detail information of these kits can be found in the following sections, the kits are:

(1) CY7113: the Infineon official kit which supports single sink-only/source-only USB PD port based on CYPM1311. This kit doesn’t have Arduino pin-to-pin capability, and user/developer should do the jump wiring the connections to other modules manually.

(2) PMGDuino: the kit is designed for supporting dual DRP (Dual Role Power) USB PD ports based on CYPM1321. This kit has the Arduino pin-to-pin capability to Arduino MKR Zero and user/developer can stack other modules to the Arduino header directly.

The PMG1 Duino framework is based on Free RTOS, and user/developer would implement the code without any concern of breaking the routine functionality operation and event handling executing in the background, e.g. USB PD.

Both CY7113 and PMG1Duino supports up to 28V_{DC} V_{BUS} which compatible to USB PD revision 3.1.

1.1 Overview of supporting features

This section describes the supporting features of CY7113 and PMGDuino kits.

1.1.1 CY7113

- 1 Sink-Only/Source-Only USB PD port with built-in NFET gate drivers and dead-battery feature
- USB 2.0 Full-Speed interface[*1]
- 128k-Byte Flash memory embedded
- 32k-Byte RAM memory embedded
- Supports USB PD Revision 3.1
- Support up to 28V V_{BUS} Extended Power Range (EPR)
- Configurable[*2] V_{BUS} Over-Voltage Protection (OVP), Over-Current Protection (OCP), Short-Circuit Protection (SCP) and Reverse-Current Protection (RCP)[*3]
- VCONN FET with Over-Current Protection (OCP)
- Built-in 12-bit SAR ADC, up to 2 channels and 1MSPS converting rate[*4] are supported
- Built-in lower power comparators[*1]

1.1.2 PMG1Duino

- 2 Dual Power Role USB PD port with built-in NFET gate drivers and dead-battery feature
- USB 2.0 Full-Speed interface[*1]
- 128k-Byte Flash memory embedded
- 32k-Byte RAM memory embedded
- Supports USB PD Revision 3.1
- Support up to 28V V_{BUS} Extended Power Range (EPR)
- Configurable[*2] V_{BUS} Over-Voltage Protection (OVP), Over-Current Protection (OCP), Short-Circuit Protection (SCP) and Reverse-Current Protection (RCP)[*3]
- VCONN FET with Over-Current Protection (OCP)
- Built-in 12-bit SAR ADC, up to 7 channels and 1MSPS converting rate[*4] are supports
- Built-in lower power comparators[*1]

Note:
**1: Not supported in firmware base version V0.1.2 and plan to be supported in V0.2.0.*
**2: The threshold of OVP, OCP and SCP is fixed to the default value 120%, 120% and ≥6A respectively.*
**3: RCP is not enabled in firmware base version V0.1.2.*
**4: The converting rate of 1MSPS is depended on the number of ADC channels are deployed.*
**5: PMGDuino only supports OVP, OCP and SCP on provider power path (external power source to USB-C connector), user/developer should be aware the consumer power path is souring from a qualified power source.*

1.2 Introduction of supporting kits

The details hardware information is described in this chapter including component placement, pin assignment of headers and jumpers and the functionalities of each component.

The hardware schematic design can be found in the appendix of this document for the further reference.

1.2.1 CY7113

1.2.1.1 CY7113 board overview

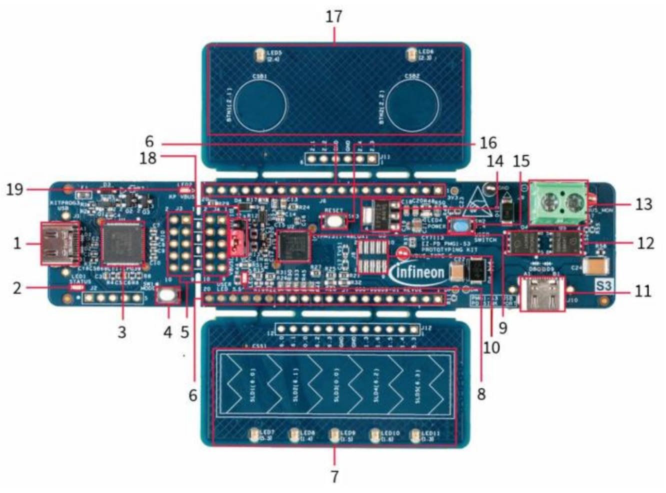


Figure 1 CY7113 component placement

Table 1 CY7113 Component placement list

Item	Description	Function	Note
1	USB-C connector	USB-C connector for debugger (KitProg3) board	
2	LED	Status indication of debugger board	
3	PSoC5LP MCU	MCU of debugger board	
4	Mode switch	Mode switch of debugger	
5	SWD header	SWD header of debugger board	J3/J4
6	GPIO header	GPIO/ADC header of CYPM1311	J6/J7
7	CAPSENSE™ board	CAPSENSE™ board	
8	LED	LED will blink in 1s period if under USB bootloader mode. Under user application mode, it can be used as LED_BUILTIN	
9	CYPM1311 MCU	USB PD MCU CYPM1311	
10	10-pin SWD/JTAG header	Reserved 10-pin SWD/JTAG header	
11	USB-C connector	USB-C connector for application board (CYPM1311)	J10
12	Load switch circuit	Back-to-back MOSFET switch circuit for V_{BUS}	
13	V_{BUS} power header	V_{BUS} power output header	
14	LDO circuit	LDO circuit for +3.3V system power regulating	
15	Button SW2	Function specific button for USB bootloader mode / user application mode switching	
16	Button SW3	Global reset button of MCU CYPM1311	
17	CAPSENSE™ board	CAPSENSE™ board	
18	Power selection jumper	Power selection jumper for system power (J5)	J5
19	Debugger Power LED	Power good indication LED of debugger board	

1.2.1.2 CY7113 board hardware information

Table 2 CY7113 jumper J5 (System power selection jumper) pin assignment

Item	Description	Function	Note
1	Pin 1-2	Select the V_{BUS} from USB-C as the system power source	
2	Pin 2-3	Select the debugger $+V_{DC}$ as the system power source	

Table 3 CY7113 header J3/J4 (SWD header) pin assignment

Pin #	Description	Function	Note
1	$+V_{DC}$	$+3.3V_{DC}$ Power pin	
2	KP_VBUS	KitProg3 VBUS / Programming V_{BUS}	
3	Grounding	Grounding pin	
4	I^2C SCL	I^2C clock signal pin for debugging purpose	
5	RST#	Global reset pin of CYPM1311 MCU	

Pin #	Description	Function	Note
6	I^{2}C SDA	I^{2}C data signal pin for debugging purpose	
7	SWD CLK	SWD clock signal input pin	
8	UART RX	UART RX signal pin for debugging purpose	
9	SWD IO	SWD data signal I/O pin	
10	UART TX	UART TX signal pin for debugging purpose	

Table 4 CY7113 header J6 (GPIO header) pin assignment

Pin #	Description	Function	Note
1	+3.3V	+3.3V$_{DC}$ power output pin	
2	VDDIO	Power pin of VDDIO of CYPM1311 (for coupling only)	
3	Grounding	Grounding pin	
4	CC1	USB PD CC1 signal pin for debugging purpose	
5	CC2	USB PD CC2 signal pin for debugging purpose	
6	P5_5	LED_BUILTIN pin	
7	P3_3	SW2 button for bootloader/application F/W switching	
8	RST#	Global reset pin of CYPM1311 MCU	
9	D0	GPIO D0 / UART RX	
10	D1	GPIO D1 / UART TX	
11	P3_3	Not applicable	
12	D2	GPIO D2 with PWM capability	
13	D3	GPIO D3	
14	D4	GPIO D4 with PWM capability	
15	A0/D5	ADC pin 0 / GPIO D5	
16	A1/D6	ADC pin 1 / GPIO D6	
17	D7	GPIO D7	
18	Grounding	Grounding pin	
19	D8	GPIO D8	
20	P5_5	Not applicable	

Table 5 CY7113 header J7 (GPIO header) pin assignment

Pin #	Description	Function	Note
1	VDDD	Power pin of VDDD of CYPM1311 (for coupling only)	
2	VCONN	VCONN power pin for debugging purpose	
3	Grounding	Grounding pin	
4	D22	GPIO pin 22	
5	D21	GPIO pin 21	
6	D20	GPIO D20 / I^{2}C SDA pin	
7	D19	GPIO D19 / I^{2}C SCL pin	

Pin #	Description	Function	Note
8	D18	GPIO D18 with PWM capability	
9	D17	GPIO D17 with PWM capability / SPI COPI pin	
10	D16	GPIO D16	
11	D15	GPIO D15 with PWM capability	
12	D14	GPIO D14	
13	D13	GPIO D13	
14	D12	GPIO D12	
15	Grounding	Grounding pin	
16	D11	GPIO D11 / SPI CIPO pin	
17	D10	GPIO D10 / SPI SCK pin	
18	VSYS_IN	Not applicable	
19	D9	GPIO D9	
20	P5_1	Not applicable. Reserved for CAPSENSE CMOD.	

Note:

**1: J6.15 (P2_2) and J6.16 (P2_1) are the share pins of CAPSENSE pins for CSD button 1 and 2.*

**2: J7.14(P6_0), J7.13(P6_1), J7.11(P6_2) and J7.12(P0_0) are the share pins of CAPSENSE pins of CSD slider 1-5.*

**3: Pin name with star mark has external 330-ohm pull-up resistor to system power (+3.3V_{DC}).*

**4: For fully deploying the pins in J6 and J7, 0-ohm resistors should be mounted to the resistors R64, R65, R67, R68, R69, R70 and R72.*

**5: For making SW2 working, a 4.7k-ohm resistor should be mounted to R49 and a 1uF capacitor should be mounted to C23.*

1.2.2 PMGDuino

1.2.2.1 PMGDuino board overview

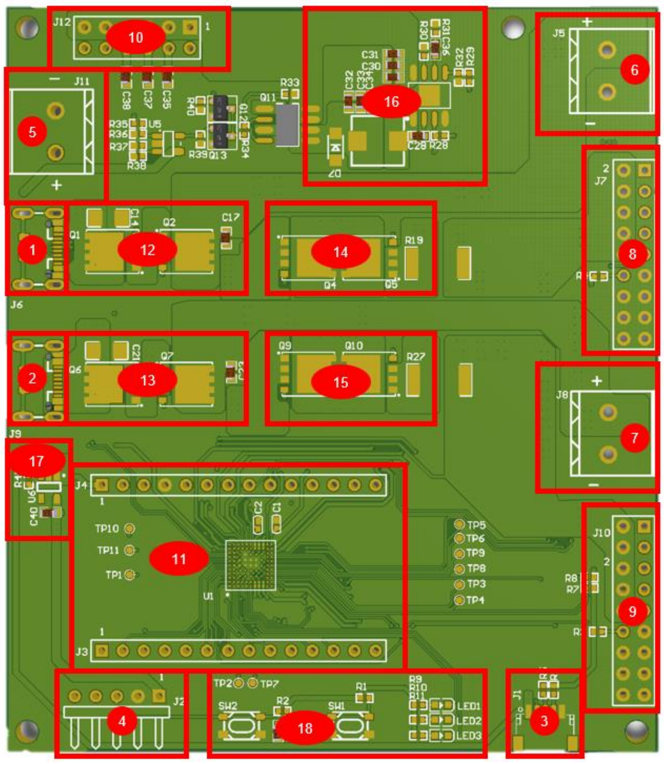


Figure 2 PMGDuino component placement

Table 6 PMGDuino Component placement list

Item	Description	Function	Note
1	USB-C connector	USB-C connector of port 0	J6
2	USB-C connector	USB-C connector of port 1	J9
3	I^{2}C connector	Additional I^{2}C connector for the connection to the external modules	J1
4	JTAG/SWD connector	Reserved JTAG/SWD connector	J2
5	+V$_{DC}$ power input header	+V$_{DC}$ power input for the system power, input range should be 5V-28V.	J11
6	Sink power output header	Sinking power output header for USB-C port 0	J5
7	Sink power output header	Sinking power output header for USB-C port 1	J8
8	Source power input header	Source power input header for USB-C port 0	J7
9	Source power input header	Source power input header for USB-C port 1	J10
10	System power source selection	System power source selection jump header	J12
11	PMG1 MCU and Arduino header	CYPM1321 MCU and Arduino pin-to-pin compatibility headers	U1/J3/J4
12	Back-to-back MOSFET switch circuit for sink power path	Back-to-back MOSFET switch circuit for sink power path of USB-C port 0	

Item	Description	Function	Note
13	Back-to-back MOSFET switch circuit for sink power path	Back-to-back MOSFET switch circuit for sink power path of USB-C port 1	
14	Back-to-back MOSFET switch circuit for source power path	Back-to-back MOSFET switch circuit for source power path of USB-C port 0	
15	Back-to-back MOSFET switch circuit for source power path	Back-to-back MOSFET switch circuit for source power path of USB-C port 1	
16	Buck circuit for system power	The buck circuit for the power supply	
17	$5V_{DC}$-to-$3.3V_{DC}$ regulator circuit	$5V_{DC}$-to-$3.3V_{DC}$ regulator circuit for the MCU, buttons and LEDs	
18	Buttons and LEDs	Buttons and LEDs for the specific function or for the user applications	SW1/SW2 LED1-3

1.2.2.2 PMGDuino board hardware information

This section lists the general descriptions, pin assignments and functionalities of each individual component, header and jumper on the board.

Table 7 PMGDuino connector J1 (I^2C) pin assignment

Pin #	Description	Function	Note
1	$+5V_{DC}$ power	$+5V_{DC}$ power output pin	
2	Reserved	Reserved pin, should leave as a float pin	
3	I^2C clock	I^2C clock signal input/output pin, an overlapped pin to the I^2C clock pin in header J4	
4	I^2C data	I^2C data signal input/output pin, an overlapped pin to the I^2C data pin in header J4	
5	Grounding	Grounding pin	

Table 8 PMGDuino header J2 (SWD) pin assignment

Pin #	Description	Function	Note
1	$+3.3V_{DC}$ power	$+3.3V_{DC}$ power input pin	
2	Grounding	Grounding pin	
3	Global reset pin	Global reset signal input pin of the MCU	
4	SWD clock	SWD clock signal input pin	
5	SWD data	SWD data signal I/O pin	

Table 9 PMGDuino header J3 (Arduino header) pin assignment

Pin #	Description	Function	Note
1	AREF	ADC reference input pin Not applicable in PMG1 Duino	
2	A0 / D15	ADC input A0 / GPIO D15 with PWM capability	
3	A1 / D16	ADC input A1 / GPIO D16	

Pin #	Description	Function	Note
4	A2 / D17	ADC input A2 / GPIO D17	
5	A3 / D18	ADC input A3 / GPIO D18	
6	A4 / D19	ADC input A4 / GPIO D19	
7	A5 / D20	ADC input A5 / GPIO D20	
8	A6 / D21	ADC input A6 / GPIO D21	
9	D0	GPIO D0 with PWM capability	
10	D1	GPIO D1 with PWM capability	
11	D2	GPIO D2 with PWM capability	
12	D3	GPIO D3 with PWM capability	
13	D4	GPIO D4 with PWM capability	
14	D5	GPIO D5 with PWM capability	

Table 10 PMGDuino header J4 (Arduino header) pin assignment

Pin #	Description	Function	Note
1	+5V_{DC}	+5V_{DC} power output pin	
2	VIN	+V_{DC} power input pin Up to 36V_{DC} tolerance	
3	+3.3V_{DC}	+3.3V power output pin	
4	Grounding	Grounding pin	
5	XRES	Global reset pin of MCU	
6	D14	GPIO D14 / UART TX pin	
7	D13	GPIO D13 / UART RX pin	
8	D12	GPIO D12 / I^2C SCL pin	
9	D11	GPIO D12 / I^2C SDA pin	
10	D10	GPIO D10 / SPI CIPO pin	
11	D9	GPIO D9 / SPI SCK pin	
12	D8	GPIO D8 / SPI COPI pin	
13	D7	GPIO D7	
14	D6	GPIO D6	

Table 11 PMGDuino jumper J12 (System power source selection jumper) pin assignment

Item	Description	Function	Note
1	Pin 1-2	System power soured from J11	
2	Pin 3-4	System power soured from VIN of J3	
3	Pin 5-6	System power soured from V_{BUS} of USB-C port 0	
4	Pin 7-8	System power soured from V_{BUS} of USB-C port 1	
5	Pin 9-10	System power soured from pin 1 and pin 3 of J7	
6	Pin 11-12	System power soured from pin 1 and pin 3 of J10	

Table 12 PMGDuino header J7/J10 (V_{BUS} sourcing header) pin assignment

Pin #	Description	Function	Note
1	Unregulated VIN	Unregulated power source from PMIC board Up to $36V_{DC}$ tolerance	
2	VIN	Regulated power input pin from PMIC board	
3	Unregulated VIN	Unregulated power source from PMIC board Up to $36V_{DC}$ tolerance	
4	VIN	Regulated power input pin from PMIC board	
5	I^2C SCL	I^2C SCL output pin for PMIC configuration	
6	VIN	Regulated power input pin from PMIC board	
7	I^2C SDA	I^2C SDA output pin for PMIC configuration	
8	VIN	Regulated power input pin from PMIC board	
9	INT#	Interrupt input pin from PMIC	
10	VIN	Regulated power input pin from PMIC board	
11	Grounding	Grounding pin	
12	IND	Indication pin, low if PWIC board is connected	
13	Grounding	Grounding pin	
14	Grounding	Grounding pin	
15	Grounding	Grounding pin	
16	Grounding	Grounding pin	
17	Grounding	Grounding pin	
18	Grounding	Grounding pin	

Table 13 PMGDuino button functionalities

Item	Description	Function	Note
1	SW1	Function specific button for triggering the F/W jumping to USB bootloader mode or user application mode	
2	SW2	Function specific button for global reset of CYPM1321	

Table 14 PMGDuino LED functionalities

Item	Description	Function	Note
1	LED1	Reserved LED for the user application (LED1)	
2	LED2	Function specific LED for USB-C connection status indication. It will blink in 1s period if under USB bootloader mode. It will blink in 10Hz if under user application mode and any USB-C contract is established.	
3	LED3	Reserved LED for the user application (BUILTIN_LED)	

Table 15 PMGDuino header J5/J8/J11 functionalities

Pin #	Description	Function	Note
1	J5	Sink power output header of USB-C port 0	
2	J8	Sink power output header of USB-C port 1	
3	J11	$+V_{DC}$ power input header, up to 36VDC tolerance	

1.2.3 RT6190 Buck-Boost board

1.2.3.1 RT6190 Buck-Boost board overview

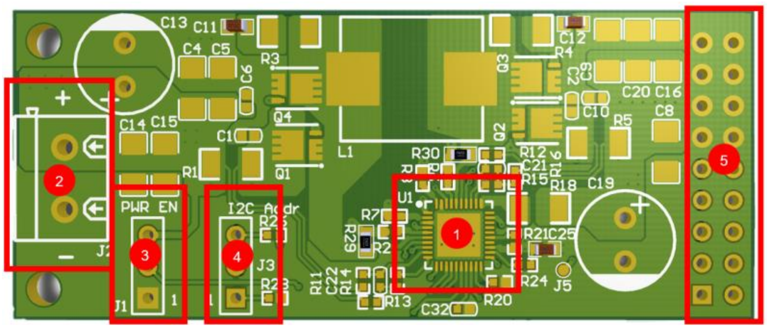


Figure 3 RT6190 Buck-Boost board component placement

Table 16 RT6190 Buck-Boost board Component placement list

Item	Description	Function	Note
1	Buck-Boost RT6190	PMIC Buck-Boost RT6190	U1
2	Unregulated VIN header	Unregulated VIN header, up to $+36V_{DC}$ tolerance	J2
3	Enable jumper	Reserved power output enable pin of RT6190	J1
4	I2C address selection jumper	Jumper for I2C device address selection of RTL6190	J3
5	Header for PMGDuino	Power output header to PMGDuino	J4

1.2.3.2 RT6190 Buck-Boost board hardware information

Table 17 RT6190 Buck-Boost board jumper J1 (Enable jumper) pin assignment

Pin #	Description	Function	Note
1	Enable by Power-On	RT6190 power output is enabled if pin 1-2 is shorted	
2	Enable pin	EN pin of RT6190	
3	Not applicable	This pin is grounding and should be connected	

Table 18 RT6190 Buck-Boost board header J3 (I^2C address selection) pin assignment

Item	Description	Function	Note
1	Pin 1-2	Pull-high and I^2C device address is 0x2C	Port 0
2	Pin 2-3	Pull-down and I^2C device address is 0x2D	Port 1

Table 19 RT6190 Buck-Boost board jumper J4 (Header for PMGDuino) pin assignment

Pin #	Description	Function	Note
1	Unregulated +V_{DC}	Unregulated +V_{DC} for PMGDuino board	
2	Regulated +V_{DC}	Regulated +V_{DC} for PMGDuino board	
3	Unregulated +V_{DC}	Unregulated +V_{DC} for PMGDuino board	
4	Regulated +V_{DC}	Regulated +V_{DC} for PMGDuino board	
5	I^2C SCL	I^2C SCL signal input for RT6190 configuration	
6	Regulated +V_{DC}	Regulated +V_{DC} for PMGDuino board	
7	I^2C SDA	I^2C SDA signal for RT6190 configuration	
8	Regulated +V_{DC}	Regulated +V_{DC} for PMGDuino board	
9	INT#	Interrupt signal output to PMGDuino board	
10	Regulated +V_{DC}	Regulated +V_{DC} for PMGDuino board	
11	Grounding	Grounding pin	
12	Grounding	Grounding pin	
13	Grounding	Grounding pin	
14	Grounding	Grounding pin	
15	Grounding	Grounding pin	
16	Grounding	Grounding pin	
17	Grounding	Grounding pin	
18	Grounding	Grounding pin	

2 PMG1 Duino quick getting started guide

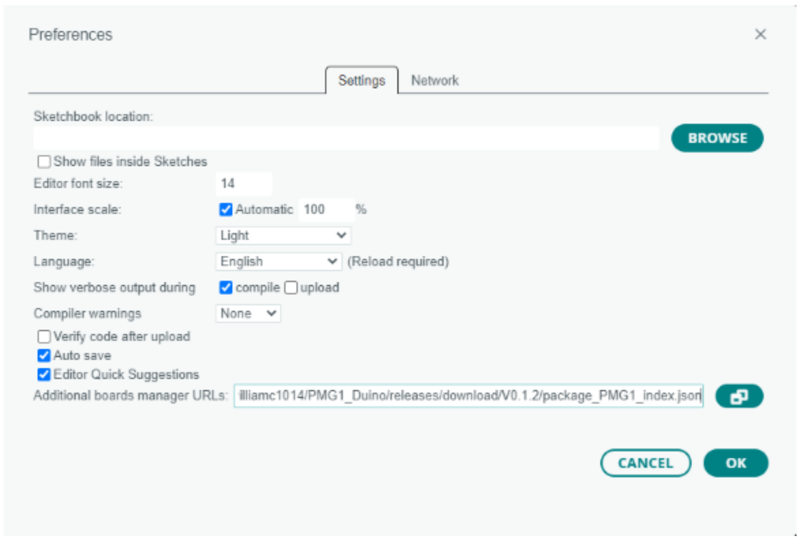
This section illustrates the step-by-step of how-to setup and install PMG1 Duino framework to your existing Arduino environment.

Note: PMG1 Duino currently only supports Windows platform in the version V0.1.2, for other platforms such as Linux and MacOS is planned to be supporting in the version V0.2.0

Steps:

1. Launch your *Arduino IDE*.
2. Open Arduino *Preference* setting dialog by selecting menu *File* and click on *Preference* item.
3. In the *Preference* dialog fill the URL link beneath to the filed *Additional boards manager URLs* and then click on *OK* button, as shown in the figure beneath.

   https://github.com/Infineon/PMG1-for-Arduino/releases/download/V0.1.2/package_PMG1_index.json

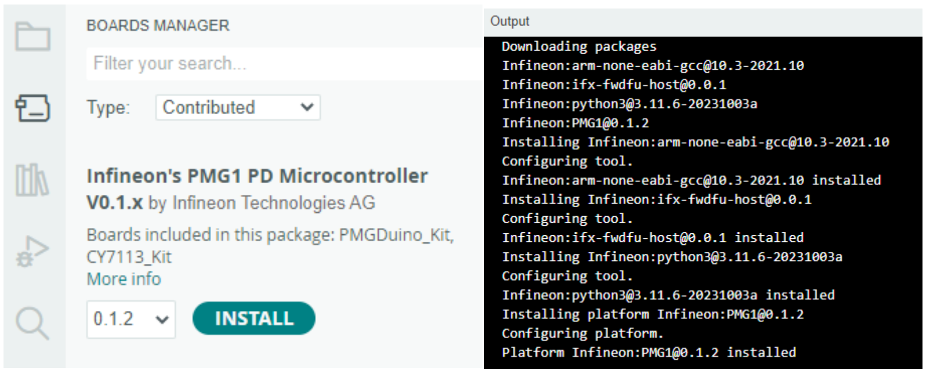

Preferences

Settings | Network

Sketchbook location:

☐ Show files inside Sketches
Editor font size: 14
Interface scale: ☑ Automatic 100 %
Theme: Light
Language: English (Reload required)
Show verbose output during ☑ compile ☐ upload
Compiler warnings None
☐ Verify code after upload
☑ Auto save
☑ Editor Quick Suggestions
Additional boards manager URLs: illiamc1014/PMG1_Duino/releases/download/V0.1.2/package_PMG1_index.json

4. Click on *Board Manager* icon in the left side bar and select the *Type* to *Contributed*.
5. Find the item named as *Infineon's PMG1 PD Microcontroller V0.1.x*, select the version (in this case, select V0.1.2) and click on *INSTALL* button. After doing this, you can see the content of the PMG1 Duino is getting started to download into your PC and you can find the similar result in the *Output* console as shown in the figure beneath.


BOARDS MANAGER

Filter your search...

Type: Contributed

Infineon's PMG1 PD Microcontroller V0.1.x by Infineon Technologies AG

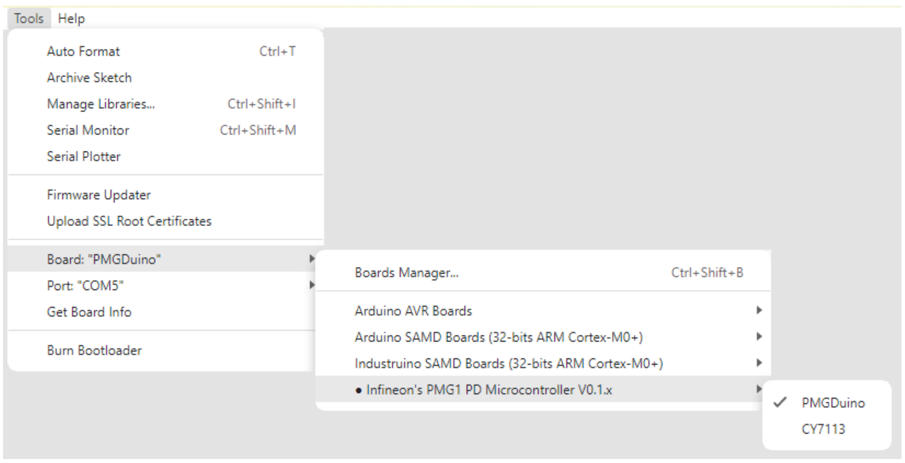
Boards included in this package: PMGDuino_Kit, CY7113_Kit

More info

0.1.2

```
Downloading packages
Infineon:arm-none-eabi-gcc@10.3-2021.10
Infineon:ifx-fwdfu-host@0.0.1
Infineon:python3@3.11.6-20231003a
Infineon:PMG1@0.1.2
Installing Infineon:arm-none-eabi-gcc@10.3-2021.10
Configuring tool.
Infineon:arm-none-eabi-gcc@10.3-2021.10 installed
Installing Infineon:ifx-fwdfu-host@0.0.1
Configuring tool.
Infineon:ifx-fwdfu-host@0.0.1 installed
Installing Infineon:python3@3.11.6-20231003a
Configuring tool.
Infineon:python3@3.11.6-20231003a installed
Installing platform Infineon:PMG1@0.1.2
Configuring platform.
Platform Infineon:PMG1@0.1.2 installed
```

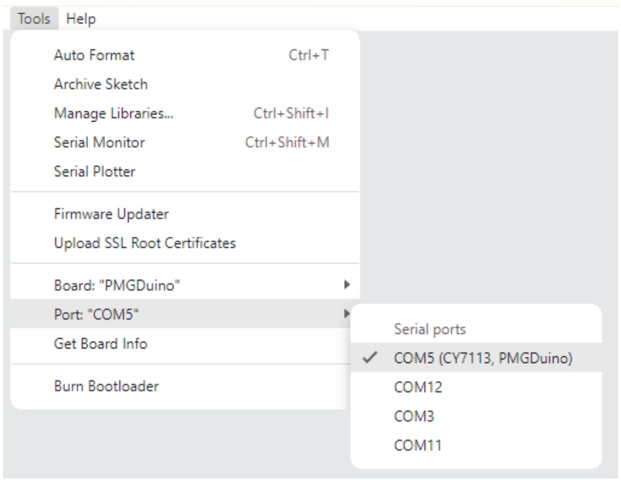
6. Now you can select the PMG1 Duino support kits for your development, by clicking on Tools in the menu, then Boards, find the *Infineon's PMG1 PD Microcontroller V0.1.x* and then select either *PMGDuino* or *CY7113* depended on which target kit being deployed.


Tools
Help
Auto Format Ctrl+T
Archive Sketch
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter
Firmware Updater
Upload SSL Root Certificates
Board: "PMGDuino"
Port: "COM5"
Get Board Info
Burn Bootloader
Boards Manager... Ctrl+Shift+B
Arduino AVR Boards
Arduino SAMD Boards (32-bits ARM Cortex-M0+)
Industruino SAMD Boards (32-bits ARM Cortex-M0+)
● Infineon's PMG1 PD Microcontroller V0.1.x
✓ PMGDuino
CY7113

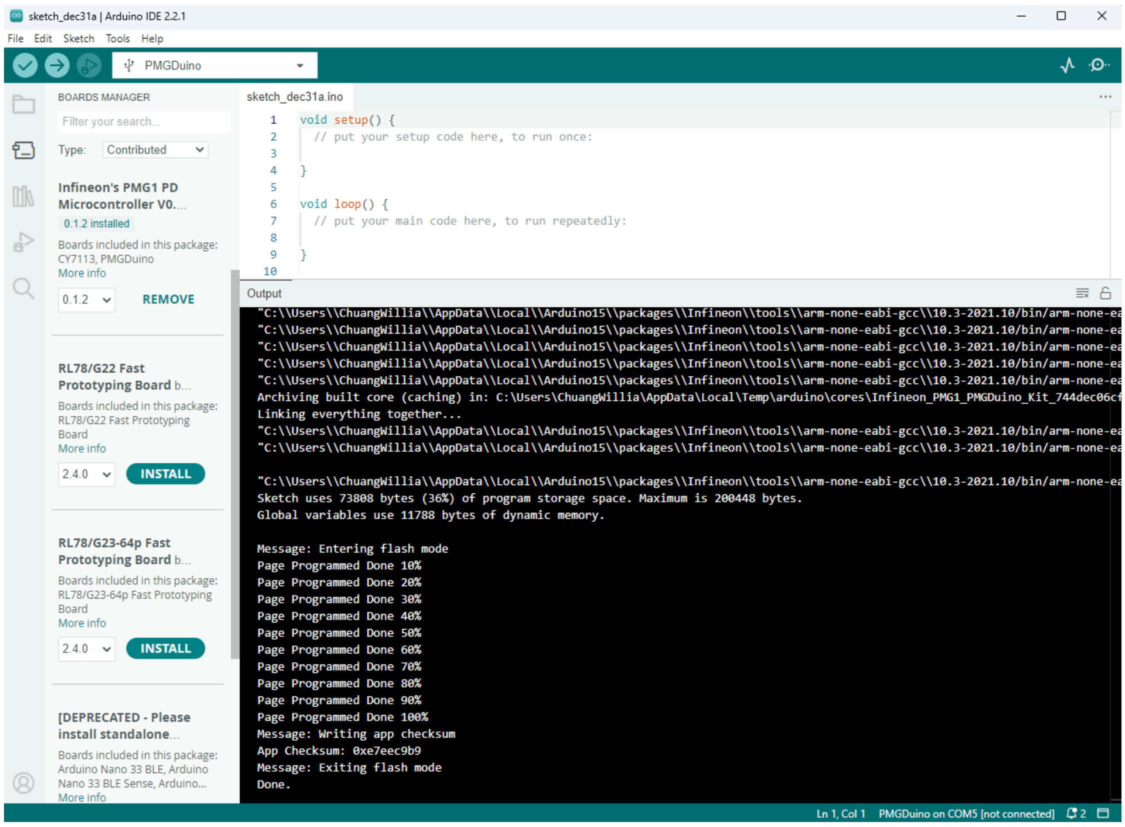
7. Connect your target kit to the PC:
   a. CY7113
      i. Connect your PC to USB-C port (J10)
      ii. Make sure the system power is sourcing from V_{BUS} (by shorting the pin 1 and pin 2 of J5)
      iii. Make sure the board is under USB bootloader mode (LED3 is blinking in 1s period), if it is not, press on SW2 to make it into the USB bootloader mode.
   b. PMGDuino
      i. Connect your PC to USB-C port 0 (J6)
      ii. Make sure the system power is powering to the board by set the jumper J12 to the proper power source, it is recommended to make the board be sourcing from V_{BUS} (by shorting the pin 5 and pin 6 of J12)
      iii. Make sure the board is under USB bootloader mode (LED2 is blinking in 1s period), if it is not, press on SW1 to make it into the USB bootloader mode.

Note: If there is any valid user application available in the MCU flash memory, the MCU will stay at USB bootloader mode for no longer than 300s (5 minutes), if the time is expired and keeping idle at USB bootloader mode, the MCU will jump to user application mode automatically. That means you will need to trigger the MCU to enter the USB bootloader mode again.

8. Select the serial COM port to the specific port by selecting *Tools* in the menu, go to *Port* and the select the specific port shown in the list as the example shown beneath.



9. Develop your code and once you finish it, click on *Upload* icon, the code will be compiled and uploaded to the target kit as the example shown beneath.


sketch_dec31a | Arduino IDE 2.2.1
File Edit Sketch Tools Help
PMGDuino
BOARDS MANAGER
Filter your search...
Type: Contributed
Infineon's PMG1 PD Microcontroller V0....
0.1.2 installed
Boards included in this package: CY7113, PMGDuino
More info
0.1.2
REMOVE
RL78/G22 Fast Prototyping Board b...
Boards included in this package: RL78/G22 Fast Prototyping Board
More info
2.4.0
INSTALL
RL78/G23-64p Fast Prototyping Board b...
Boards included in this package: RL78/G23-64p Fast Prototyping Board
More info
2.4.0
INSTALL
[DEPRECATED - Please install standalone...
Boards included in this package: Arduino Nano 33 BLE, Arduino Nano 33 BLE Sense, Arduino...
More info
sketch_dec31a.ino
Output
Ln 1, Col 1 PMGDuino on COM5 [not connected] 2

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

```
"C:\\Users\\ChuangWillia\\AppData\\Local\\Arduino15\\packages\\Infineon\\tools\\arm-none-eabi-gcc\\10.3-2021.10/bin/arm-none-ea
"C:\\Users\\ChuangWillia\\AppData\\Local\\Arduino15\\packages\\Infineon\\tools\\arm-none-eabi-gcc\\10.3-2021.10/bin/arm-none-ea
"C:\\Users\\ChuangWillia\\AppData\\Local\\Arduino15\\packages\\Infineon\\tools\\arm-none-eabi-gcc\\10.3-2021.10/bin/arm-none-ea
"C:\\Users\\ChuangWillia\\AppData\\Local\\Arduino15\\packages\\Infineon\\tools\\arm-none-eabi-gcc\\10.3-2021.10/bin/arm-none-ea
"C:\\Users\\ChuangWillia\\AppData\\Local\\Arduino15\\packages\\Infineon\\tools\\arm-none-eabi-gcc\\10.3-2021.10/bin/arm-none-ea
Archiving built core (caching) in: C:\Users\ChuangWillia\AppData\Local\Temp\arduino\cores\Infineon_PMG1_PMGDuino_Kit_744dec06cf
Linking everything together...
"C:\\Users\\ChuangWillia\\AppData\\Local\\Arduino15\\packages\\Infineon\\tools\\arm-none-eabi-gcc\\10.3-2021.10/bin/arm-none-ea
"C:\\Users\\ChuangWillia\\AppData\\Local\\Arduino15\\packages\\Infineon\\tools\\arm-none-eabi-gcc\\10.3-2021.10/bin/arm-none-ea

"C:\\Users\\ChuangWillia\\AppData\\Local\\Arduino15\\packages\\Infineon\\tools\\arm-none-eabi-gcc\\10.3-2021.10/bin/arm-none-ea
Sketch uses 73808 bytes (36%) of program storage space. Maximum is 200448 bytes.
Global variables use 11788 bytes of dynamic memory.

Message: Entering flash mode
Page Programmed Done 10%
Page Programmed Done 20%
Page Programmed Done 30%
Page Programmed Done 40%
Page Programmed Done 50%
Page Programmed Done 60%
Page Programmed Done 70%
Page Programmed Done 80%
Page Programmed Done 90%
Page Programmed Done 100%
Message: Writing app checksum
App Checksum: 0xe7eec9b9
Message: Exiting flash mode
Done.
```

3 Programming guide

This section describes the APIs (Application Program Interface) of the hardware interface and functionalities including the overview of each functionality, the limitations and supporting features of individual hardware functionalities and interfaces.

Functions *pulseIn*, *interrupts* and *noInterrupts* are not supported in PMG1 Duino framework.

3.1 GPIO

There are 23 digital GPIO pins available in CY7113 kit and 22 digital GPIO pins in PMGDuino kit.

All the digital GPIO can be configured as the digital input pin or digital output pin.

Some of them have PWM and Tone capabilities which can generate the PWM waveform with the specific frequency and duty period. Please find section 3.6 for detail information.

Both CY7113 and PMGDuino has one built-in LED which is not included in the GPIO pin list, user/developer can assign the LED status by using LED_BUILTIN pin name.

```
digitalWrite(LED_BUILTIN, 1);    // Turn off the built-in LED
digitalWrite(LED_BUILTIN, 0);    // Turn on the built-in LED
```

PMGDuino has an additional LED for the application beyond the LED_BUILTIN, user/developer can assign this LED status by using LED1 pin name.

```
digitalWrite(LED1, 1);    // Turn off the LED1
digitalWrite(LED1, 0);    // Turn on the LED1
```

Note: BUILTIN_LED and LED1 is configured as the output pin and can't be reconfigured to input mode, any mode change to these pins will be ignored.

User can use *pinMode* function to set the specific digital pin as either input mode or output mode.

```
pinMode(D0, OUTPUT); // Set digital pin D0 as a output pin
pinMode(D1, INPUT);  // Set digital pin D1 as a input pin
```

Additionally, PMG1 Duino offer another mode that can enable the internal pull-up resistor of the specific pin.

```
pinMode(D3, INPUT_PULLUP); // Set digital pin D3 as a input pin with internal pull-high resistor enabled
```

For assigning the state of the specific digital pin, user/developer can use *digitalWrite* function to assign the pin state to the specific pin.

```
digitalWrite(D0, HIGH);  // Set digital pin D0 to high state
digitalWrite(D0, LOW);   // Set digital pin D0 to low state
digitalWrite(D0, 1);     // Set digital pin D0 to high state
digitalWrite(D0, 0);     // Set digital pin D0 to low state
```

User/developer can use *digitalRead* to get the pin state from the specific digital pin.

```
int pinStateD2 = digitalRead(D2); // Get the pin state from digital pin D2
int pinStateD3 = 0; // Claim a variable to store the pin state
pinStateD3 = digitalRead(D3); // Get the pin state from digital pin D2
```

For shift in or shift out the bits through digital pins, user/developer can use *shiftIn* and *shiftOut* function respectively for this purpose.

```
shiftIn(D1, D0, LSBFIRST); // Shift in the bits in LSB first bit-order. Data pin: D0, clock pin: D1

shiftOut(D1, D0, MSBFIRST); // Shift out the bits in MSB first bit-order. Data pin: D0, clock pin: D1
```

There is no additional delay between the clock asserting to high and deserting to low in the implementation of functions *shiftIn* and *shiftOut*, the clock rate will be very high as the MCU execution rate is up to 48MHz therefore it is recommended to use another approach for shift in/out the bits. The example code is shown below:

```
byte altShitIn(byte clockPin, byte dataPin, byte bitOrder, int pluseWidthUs)
{
        byte retVal = 0;
        byte i = 0;

        for (i = 0; i < 8; ++i)
        {
                digitalWrite(clockPin, HIGH);
                delayMicroseconds(pluseWidthUs);
                if (bitOrder == LSBFIRST)
                        retVal |= digitalRead(dataPin) << i;
                else
                        retVal |= digitalRead(dataPin) << (7-i);
                digitalWrite(clockPin, LOW);
                delayMicroseconds(pluseWidthUs);
        }
        return retVal;
}

byte altShitOut(byte clockPin, byte dataPin, byte bitOrder, byte val, int pluseWidthUs)
{
        byte i = 0;

        for (i = 0; i < 8; i ++)
        {
                if (bitOrder == LSBFIRST)
                {
                        digitalWrite(dataPin, (val & 0x01));
                        val = val >> 1;
                }
                else
                {
                        digitalWrite(dataPin, (val & 0x80) != 0);
                        val = val << 1;
                }
                digitalWrite(clockPin, HIGH);
                delayMicroseconds(pluseWidthUs);
                digitalWrite(clockPin, LOW);
                delayMicroseconds(pluseWidthUs);
        }
}
```

There is an implementation that user/developer can register the interrupt event for the specific digital pin by using *attachInterrupt* function.

Firstly, user/developer should implement an interrupt service function for the pin interrupt events. For example.

```
void intr_d1( )
{
        // Implement you service call here for the pin interrupt event
}
```

Then user/developer can use *attachInterrupt* function to register event for the specific pin with the specific trigger condition.

```
attachInterrupt(D1, intr_d1, LOW); // Triggered at low state and intr_d1 will be called

attachInterrupt(D1, intr_d1, CHANGE); // Triggered at state change and intr_d1 will be called

attachInterrupt(D1, intr_d1, FALLING); // Triggered at high-to-low transition and intr_d1 will be called

attachInterrupt(D1, intr_d1, RISING); // Triggered at low-to-high transition and intr_d1 will be called
```

If the event detection to the specific pin is no longer required, user/developer can use *deachInterrupt* to unhook the registered interrupt service function from the event and meanwhile the event detection will be disabled.

```
deachInterrupt(D1); // Disable the event detection and unhook the interrupt service function
```

3.2 UART (Serial)

Both CY7113 and PMGDuino has a UART (Serial) interface.

The UART (Serial) interface supports the bitrates of 300bps, 600bps, 1200bps, 1800bps, 2400bps, 4800bps, 7200bps, 9600bps, 14400bps, 19200bps, 38400bps, 57600bps or 115200bps.

The following formats are supported in PMG1 Duino framework: *SERIAL_5N1, SERIAL_6N1, SERIAL_7N1, SERIAL_8N1, SERIAL_5N2, SERIAL_6N2, SERIAL_7N2, SERIAL_8N2, SERIAL_5E1, SERIAL_6E1, SERIAL_7E1, SERIAL_8E1, SERIAL_5E2, SERIAL_6E2, SERIAL_7E2, SERIAL_8E2, SERIAL_5O1, SERIAL_6O1, SERIAL_7O1, SERIAL_8O1, SERIAL_5O2, SERIAL_6O2, SERIAL_7O2 or SERIAL_8O2*. The default data format is *SERIAL_8N1*.

This UART (Serial) interface has a ring buffer for the input/output data buffering which the size is 32 bytes.

The USB-to-Serial function is not implemented in PMG1 Duino version V0.1.2 therefore the following statements are not supported.

```
if (Serial) {…}
if (!Serial) {…}
while(Serial) {…}
while(!Serial) {…}
```

The first thing of deploying UART (Serial) interface is to initiate the interface by calling the *Serial.begin* function, for example.

```
Serial.begin(9600); // Initiate the Serial interface with 9600bps bitrate and SERIAL_8N1 format
```

Or

Serial.begin(115200, SERIAL_8E1); // Initiate the Serial interface with 9600bps and SERIAL_8E1 format

User/developer can output data by using *Serial.write* function or as there is a print class implemented in this framework, the *print* function is also available for outputting the data, for example.

Serial.write(0xAA); // Outputs 1 byte 0xAA data to Serial interface

Serial.print("Hello World")' // Outputs string "Hello World" to Serial interface

User/developer can read the inputted data by using *Serial.read* function, for example.

int inData = Serial.read( ); // Read 1 byte data from Serial ring buffer

Or user/developer can just peek the inputted data without removing it from the Serial ring buffer by using *Serial.peek* function, for example.

int peekData = Serial.peek( ); // Only peek the data without remove it from the buffer

It is also helpful to get the number of bytes that available for the reading access and/or for the writing access to this UART (Serial) interface by using *Serial.available* and *Serial.availableForWrite* functions respectively. For example.

int num4Read = Serial.available( ); // Get how many bytes available in Serial ring buffer for reading

int num4Write = Serial.availableForWrite( ); // Get how many byte left in Serial ring buffer for writing

It is also useful if the data stored in the ring buffer is no longer needed, user/developer can call *Serial.flush* to flush out all stored data in the ring buffer. For example.

Serial.flush( ); // Clean out all stored data in ring buffer and waiting for Serial resume to idle

If this UART (Serial) in no longer required, it can be dismissed by calling the function *Serial.end*, and all occupied digital I/O will be reset to default input mode (Hi-Z).

Serial.end( ); // Dismiss Serial interface

The function *Serial.serialEvent* is not supported/implemented in this PMG1 Duino framework.

All other member functions such link *println*, *readBytes* or *parseInt* etc. are all supported in this framework, please check Arduino Language Reference for the detail information.

3.3 I²C (Wire)

Both CY7113 and PMGDuino has a Wire interface.

This Wire interface supports either bitrates of 100kbps or 400kHz.

A 32-byte ring buffer is implemented in this interface for buffering the input and/or output data.

User/developer should use *Wire.begin* function to initiate this Wire interface before this interface is going to deploying in your application and should use *Wire.end* function to dismiss this interface and the digital I/O from the interface.

If there is no device address is assigned while calling *Wire.begin* function, the value of 0xFF will be a default device address and Wire interface works as a master Wire device during the operation.

User/developer can use function *Wire.setClock* to set the bitrate of Wire interface, if this function is not involved in your design, the bitrate of 400kbps will be a default bitrate for Wire interface.

It is commonly known that user/developer should use function *Wire.beginTransmission* to initiate a new transmission activity and should use function *Wire.endTransmission* to end the transmission activity and release the I^{2}C bus.

During any transmission activity, user/developer can use *Wire.write* to queue the outputting data into the I2C ring buffer and the data will be transmitted to the I^{2}C bus while *Wire.endTransmission* is called.

User/developer can use *Wire.requestFrom* and *Wire.read* to request the data input from the external device and read the inputting data if any available data checked can be read by calling *Wire.available*.

The following example shows the simple process of accessing the data to a I^{2}C 24LC256 EEPROM device.

```
int devAddress = 0x50; // Device address
int byteAddress = 0; // Data address to be accessing in the EEPROM
byte byteData = 0;
Wire.begin( ); // Initiate Wire interface as a master device

Wire.setClock(100000); // Set Wire interface bitrate to 100kbps

// Write out the data to EEPROM
Wire.beginTransmission(devAddress); // Begin the transmission with device address 0x50
Wire.write(byteAddress >> 8); //Write out the EEROM address MSB byte
Wire.write(byteaddress & 0xFF); // Write out the EEROM address LSB byte
Wire.write(0xAA); // Write 0xAA to the EEPROM
Wire.endTransmission( ); // End the transmission and the data will be outputting to I2C bus
delay(5); // delay 5ms to wait the data being transmitted accomplished

// Read a byte from EEPROM
Wire.beginTransmission(devAddress); // Begin the transmission with device address 0x50
Wire.write(byteAddress >> 8); //Write out the ROM address MSB byte
Wire.write(byteaddress & 0xFF); // Write out the ROM address LSB byte
Wire.endTransmission( ); // End the transmission and the data will be outputting to I2C bus
Wire.requestFrom(devAddress, 1); // Request 1 byte data from EEPROM

If (Wire.available( ))
        byteData = Wire.read( ); // Read 1 byte data from Wire buffer

Wire.end( ); // Dismiss Wire interface
```

User/developer can also register the Wire events and hook the callback functions while the peripheral device is requesting data, or the data is received from the peripheral device by using *Wire.onRequest* function or *Wire.onReceive* function respectively.

```
void wireDataReceive(int numByteReceived)
{
        // Handled the received data here
}

void  wireDataRequested( )
{
        // Handle the data request event here
}
```

```
// example of initiate the event triggering of Wire interface
Wire.begin(devAddress); // Initiate the Wire interface with device address devAddress
Wire.onReceive(wireDataReceive); // Hook the interrupt service function to data received event
Wire.onRequest(wireDataRequest); // Hook the interrupt service function to data requesting event
```

Functions *setWireTimeout*, *clearWireTimeoutFlag* and *getWireTimeoutFlag* are not supported in PMG1 Dunio version V0.1.2 and planned to be supported in V0.2.0. In version V0.1.2 the timeout time is set to fixed 25ms and no timeout flag can be retrieved for the timeout status.

3.4 SPI

Both CY7113 and PMGDuino has a SPI interface.

The supporting bitrate of this SPI interface can be maximum 1.5Mbps and minimum 23.4375kbps, default bitrate is 1Mbps and the possible bitrates are calculated by the following equation:

$Bitrate_{SPI}$ *= 3MHz / DivNum, DivNum is the clock division number* 2^n*, n can be any integer value 1~7*

This SPI interface supports all 4 SPI data modes which native Arduino supports including *SPI_MODE0*, *SPI_MODE1*, *SPI_MODE2* and *SPI_MODE3*. And the bit-order can be configured as MSB bit first (*MSBFIRST*) or LSB bit first (*LSBFIRST*). The default setting of data mode and bit-order are *SPI_MODE0* and *MSBFIRST* (MSB bit first).

User/developer can use functions *setBitOrder*, *setClockDivider*, *setDataMode* for run-time configuration to the bit-order, clock rate (bitrate) and data mode setting of this SPI interface respectively.

User/developer should use the functions *begin*, *setbitOrder* (optional), *setClockDivder*(optional), *setDataMode* (optional), *beginTransaction*, *transfer* or *transfer16*, *endTransaction* and *end* (if SPI interface is no longer required), in the sequence to make sure that the SPI transaction working properly. The example code can be found in the following example code.

```
byte spiData[11] = {'H', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd'};
byte dataRecevied_8 = 0;
int dataReceived_16 = 0;

SPI.begin( ); // Initiate and configure the pin mode of SPI interface
SPI.beginTransaction(1000000, MSBFIRST, SPI_MODE0); // Initiate the SPI interface with the SPI settings
SPI.transfer(spiData, 11); // Send out 11-byte data
dataRecevied_8  = SPI.transfer(0xAA); // Send out 1 byte 0xAA data and get back the received data
dataRecevied_16 = SPI.transfer16(0x55AA); // Send out 1 word 0x55AA data and get back the received data
SPI.endTransaction( ); // Dismiss SPI interface
SPI.end( ); // Release digital pins and reset to input Hi-Z mode
```

There is no buffering mechanism implemented in this SPI interface therefore every time the user/developer call the function *SPI.transfer* the MCU execution will be block till all the data transaction is accomplished.

Function *SPI.usingInterrupt* is not supported in PMG1 Duino implementation.

3.5 ADC

CY7113 supports 2 12-bit SAR ADC input channels and PMGDuino supports 7 12-bit SAR ADC channels in PMG1 Duino implementation.

User/developer can use function *analogReference* to select the reference input source:

- *AR_DEFAULT/AR_INTERNAL*: Reference source from embedded 1.2V_{DC} bandgap circuit, provides the 0V_{DC} ~ 2.4V_{DC} (0x000 ~ 0xFFF) input voltage conversion range.
- *AR_VDD*: Reference source from 3.3V_{DC} VDDA power rail, provides the 0V_{DC} ~ 3.3V_{DC} (0x000 ~ 0xFFF) input voltage conversion range.

User/developer can use the following example code to configure the ADC reference input source.

```
analogRegerence(AR_DEFAULT);
```

Or

```
analogReference(AR_VDD);
```

User/developer can use function *analogRead* to read back the voltage on the specific analog pin (A0~A1 for CY7113, A0~A6 for PMGDuino).

The pin will be configured as an analog input pin, do the ADC conversion on the pin, reset the pin to digital input Hi-Z mode and return the value when this function is called.

While doing the *analogRead*, the MCU will be hold and blocked until the conversion is accomplished.

```
int adcVoltageVal = 0; // Variable for converted ADC value restore, must be an integer type for 12-bit result
adcVoltageVal  = (analogRead(A0) & 0xFFF); // Conversion and read the voltage level on pin A0
adcVoltageVal  = (analogRead(A1) & 0xFFF); // Conversion and read the voltage level on pin A1
```

Function *analogReadResolution* is not supported/implemented in PMG1 Duino.

3.6 PWM and analogWrite

There are 6 digital pins supports PWM functionality in CY7113 and 7 digital pins supports PWM functionality in PMGDuino.

The maximum frequency of PWM functionality is up to 8MHz (8000000Hz), if any assigned working frequency higher than 8MHz will be ignored and no PWM function will be enabled on the pin.

The duty cycle of the PWM is given to 50% and cannot be changed.

User/developer can use functions tone and noTone to enable or disable the PWM functionality on the specific digital pin respectively.

The following code shows an example of enabling or disabling the PWM on the specific digital pin.

```
tone(D2, 10000); // Enable the PWM on digital pin D2

noTone(D2); // Disable the PWM on digital pin D2
```

User/developer can use *analogWrite* to output a continues PWM waveform with the specific duty cycle on the digital pin with PWM capability.

The output PWM is given the frequency of 490Hz, and the duty cycle is in range of 0~255 with scale step of 1/256.

The code beneath shows the example of how-to use *analogWrite* function.

```
analogWrite(D2, 64); // Enable PWM function with 25% duty cycle of 490Hz PWM

analogWrite(D2, 255); // Enable PWM function with 100% duty cycle of 490Hz PWM
```

3.7 USB Power Delivery

Note: PMG1 Duino version V0.1.2 supports USB PD revision 3.1 and planned to support USB PD revision 3.2 in PMG1 Duino version V0.2.0.

CY7113 provides a USB-C port and PMGDuino provide 2 USB-C port based on their hardware capability.

Both CY7113 and PMGDuino supports USB PD SPR and EPR mode, the working V_{BUS} power rate can be up to $20V_{DC}$/5A in SPR mode and $28V_{DC}$/5A in EPR mode (with 5A USB Type-C cable).

There is a 5V/3A fixed PDO defined in the list of SPR PDO list by default (PDO0 in Sink PDO and Source PDO) and a 28V/5A fixed PDO defined for EPR mode by default.

User/developer can add or remove any custom PDO to the PDO list in SPR PDO index 1-6. There is no customization option for EPR mode to add and/or remove any PDO in EPR PDO list.

An implementation of controlling the external RT6190 PMIC is including in PMG1 Duino and the output voltage of this RT6190 PMIC can be configured to the specific required voltage level (up to 36V with scale step 20mV/Step) with input voltage in range of 3V-36V automatically depended on USB PD negotiation result.

User/developer should initiate the USB PD instance by calling the function *begin* which will initiate the variables and status from the USB PD engine. The example is shown below.

```
usbpd0.begin( ); // Initiate the USB PD port 0 interface class, available for both CY7113 and PMGDuino

usbpd1.begin( ); // Initiate the USB PD port 1 interface class, available for PMGDuino only
```

There is also a function that can *usbpdx.end* the USB PD instance but this function is a dummy function which have any implementation in this function. The examples are:

```
usbpd0.end( ); // Dismiss USB PD port 0 interface class, a dummy function which no effect by calling this

usbpd1.end( ); // Dismiss USB PD port 1 interface class, a dummy function which no effect by calling this
```

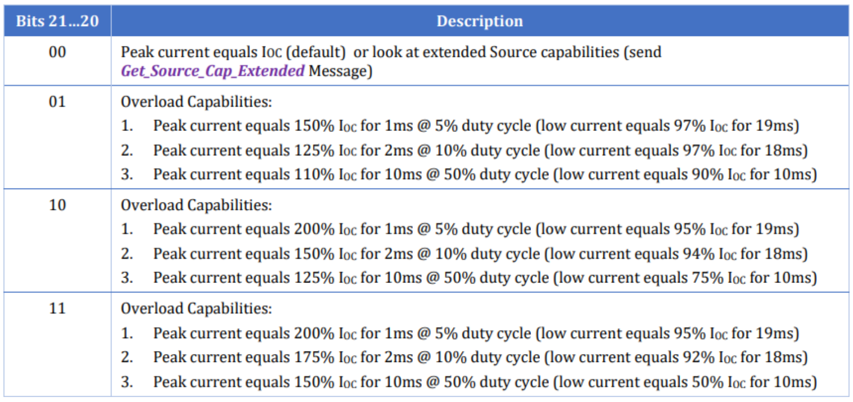
User/developer can add and change the SPR Source PDO in the source PDO list by calling the functions *addFixedSrcPdo* and *addVarSrcPdo*. After the modification to the source PDO is done, user/developer should call function *updateSrcPdo* for updating the modification to the USB PD engine. After function *updateSrcPdo* or *updateSnkPdo* are called, the USB PD will update the source PDO to the partner port and do the power negotiation again according to the requirement in the USB PD specification. The example of modifying source PDO is shown beneath.

According to USB PD specification, each PDO should have its unique voltage level, that means there should not have any individual PDO have same voltage level to others.

It is also required that voltage level of fixed PDO should be assigned from lowest to highest voltage level and the AVS PDO should be in the last PDO in the PDO list.

```
// Add a Fixed PDO to source PDO index 1, fixed voltage 9V, maximum current 3A
// Prototype (PMGDuino only)
// addFixedSrcPdo(byte pdoIndex, int voltage, int maxCurrent, byte peakCurrent)
usbpd0.addFixedSrcPdo(1, 9000, 3000, 0);
// Add a Fixed PDO to source PDO index 2, fixed voltage 15V, maximum current 3A
usbpd0.addFixedSrcPdo(2, 15000, 3000, 0);
// Add a Fixed PDO to source PDO index 3, fixed voltage 20V, maximum current 3A
usbpd0.addFixedSrcPdo(3, 20000, 3000, 0);
// Add a AVS PDO to source PDO index 4, variable voltage range 9-20V, maximum current 3A
// Prototype (PMGDuino only)
// addVarSrcPdo(byte pdoIndex, int minVoltage, int maxVoltage, int maxCurrent)
usbpd0. addVarSrcPdo(4, 9000, 20000, 3000);
// Update PDO changes to the USB PD engine
usbpd0.updateSrcPdo( );
```

Peak Current of argument in function *addFixedSrcPdo* should be set to the value range of 0-3 corresponding to USB PD specification, the meaning of each value can be referred to the figure 4 beneath. Typically, it is recommended to set it to 0.



Bits 21…20	Description
00	Peak current equals I_{OC} (default) or look at extended Source capabilities (send *Get_Source_Cap_Extended* Message)
01	Overload Capabilities: 1. Peak current equals 150% I_{OC} for 1ms @ 5% duty cycle (low current equals 97% I_{OC} for 19ms) 2. Peak current equals 125% I_{OC} for 2ms @ 10% duty cycle (low current equals 97% I_{OC} for 18ms) 3. Peak current equals 110% I_{OC} for 10ms @ 50% duty cycle (low current equals 90% I_{OC} for 10ms)
10	Overload Capabilities: 1. Peak current equals 200% I_{OC} for 1ms @ 5% duty cycle (low current equals 95% I_{OC} for 19ms) 2. Peak current equals 150% I_{OC} for 2ms @ 10% duty cycle (low current equals 94% I_{OC} for 18ms) 3. Peak current equals 125% I_{OC} for 10ms @ 50% duty cycle (low current equals 75% I_{OC} for 10ms)
11	Overload Capabilities: 1. Peak current equals 200% I_{OC} for 1ms @ 5% duty cycle (low current equals 95% I_{OC} for 19ms) 2. Peak current equals 175% I_{OC} for 2ms @ 10% duty cycle (low current equals 92% I_{OC} for 18ms) 3. Peak current equals 150% I_{OC} for 10ms @ 50% duty cycle (low current equals 50% I_{OC} for 10ms)

Figure 4 Fixed Power Source Peak Current Capability

User/developer can remove any of existing source PDO from the list by calling function *removeSrcPdo*. Similarly, user/developer should call function *updateSrcPdo* to make it effect to the USB PD engine and a new USB PD negotiation process will be triggered. The example is shown beneath.

```
// Remove source PDO index 3
// Prototype (available for PMGDuino only): removeSrcPdo(byte pdoIndex)
usbpd0.removeSrcPdo(3);
// Update PDO changes to the USB PD engine
usbpd0.updateSrcPdo( );
```

Note: If any new value setting to the existing PDO, the previous PDO setting will be overwritten by the new values.

Similar with the implementation of handling source PDO, there are functions *addFixSnkPdo*, *addVarSnkPdo*, *removeSnkPdo* and *updateSnkPdo* for the same purpose but for the power sinking from the partner port. The example is shown below.

```
// Add a Fixed PDO to sink PDO index 1, fixed voltage 9V, operation current 3A
// Prototype: addFixedSnkPdo(byte pdoIndex, int voltage, int operationCurrent)
usbpd0.addFixedSnkPdo(1, 9000, 3000);
// Add a Fixed PDO to sink PDO index 2, fixed voltage 15V, operation current 3A
usbpd0.addFixedSnkPdo(2, 15000, 3000);
// Add a Fixed PDO to sink PDO index 3, fixed voltage 20V, operation current 3A
usbpd0.addFixedSnkPdo(3, 20000, 3000);
// Add a AVS PDO to sink PDO index 4, variable voltage range 9-20V, operation current 3A
// Prototype: addVarSnkPdo(byte pdoIndex, int minVoltage, int maxVoltage, int operationCurrent)
usbpd0. addVarSnkPdo(4, 9000, 20000, 3000);
// Update PDO changes to the USB PD engine
usbpd0.updateSnkPdo( );
```

```
// :
// :
// :
// Remove sink PDO index 3
// Prototype: removeSnkPdo(byte pdoIndex)
usbpd0.removeSnkPdo(3);
// Update PDO changes to the USB PD engine
usbpd0.updateSnkPdo( );
```

Note:

(1) If any new value setting to the existing PDO, the previous PDO setting will be overwritten by the new values.

(2) Any changes to PDO index 0 (5V/3A) will be ignore.

User/developer can get the number of PDO in the PDO list by calling *getSrcPdoNum* for source PDO list (SPR PDOs) and *getSnkPdoNum* for sink PDO list (SPR PDOs). The example code can be found beneath.

```
// Prototype (available for PMGDuino only): usbpd0.getSrcPdoNum(void)
// Prototype (available for PMGDuino only): usbpd1.getSrcPdoNum(void)
// Prototype (available for both PMGDuino and CY7113): usbpd0.getSnkPdoNum(void)
// Prototype (available for PMGDuino only): usbpd1.getSnkPdoNum(void)
byte numSrcPDO_P0 = usbpd0.getSrcPdoNum( ); // Get the number of PDO in source PDO list of port 0
byte numSnkPDO_P0 = usbpd0.getSnkPdoNum( ); // Get the number of PDO in sink PDO list of port 0
byte numSrcPDO_P1 = usbpd1.getSrcPdoNum( ); // Get the number of PDO in source PDO list of port 1
byte numSnkPDO_P1 = usbpd1.getSnkPdoNum( ); // Get the number of PDO in sink PDO list of port 1
```

User/developer can get the negotiation result and find which PDO is requested (being as a power source role) or accepted (being as a power sink role) by calling function *getCurrentSrcRdo* and *getCurrentSnkPdo* respectively. The return value of 0-6 is the PDO index for SPR mode and 7-12 is for the EPR mode PDO index. The example code is shown as beneath.

```
byte pdoIndex = 0; // Variable for RDO or PDO storing
usbpd0.updateStatus( ); // update the USB-C port 0 status
if (usbpd0.contractExisted)
{
        // A USB PD device is connected to the port
        if (usbpd0.powerRole)
                pdoIndex = usbpd0.getCurrentSrcRdo( ); // Port is as power source role
        else
                pdoIndex = usbpd0.getCurrentSnkPdo( ); // Port is as power sink role
}
```

User/developer can update and get the port status of the USB Type-C and USB PD by calling function *updateStatus* which is described in the later description.

User/developer can enable and disable the source and sink EPR PDOs being presented to the partner port by calling the function *enableSrcEprPdo* and *enableSnkEprPdo* respectively. The example code is shown below.

```
// Enable or disable the EPR sink PDO of the port
// Prototype: enableSnkEprPdo(bool enable)
usbpd0.enableSnkEprPdo(true); // make sink EPR PDO be presented to partner port
usbpd0.enableSinkEprPdo(false); // make sink EPR PDO not be presented to partner port
```

```
// Enable or disable the EPR source PDO of the port
// Prototype: enableSrcEprPdo(bool enable)
usbpd0.enableSrcEprPdo(true); // make source EPR PDO be presented to partner port
usbpd0.enableSrcEprPdo(false); // make source EPR PDO not be presented to partner port
```

User/developer can retrieve the source PDO capability and sink PDO capability list by calling function *getPartnerPortSrcPdo* and *getPartnerPortSnkPdo* respectively. Please note that if the specific port of PMG1 MCU is working as power sink role, the source PDO capability list will be updated to member *iPartnerSrcPdo* when the power negotiation is accomplished – calling *getPartnerPortSrcPdo* function after the power negotiation is accomplished when the PMG1 port is working as power sink role will retrigger another power negotiation, therefore it is recommended to check the working power role before calling this function.

After the function is called, the number of available source/sink PDO will be updated to member *iPartnerSrcPdoCnt* and *iPartnerSnkPdoCnt* respectively, and the source/sink PDO content will be updated member *iPartnerSrcPdo* and *iPartnerSnkPdo* respectively. The example code is shown as beneath.

```
// Get the partner port sink PDO capability list
// Prototype: getPartnerSnkPdo(void)
usbpd0.getPartnerPortSnkPdo( );
while (!usbpd0.iPartnerSnkPdoUpdated) { } // Delay for a while for the USB PD traffic to be accomplished
for (byte i = 0; i < usbpd0.iPartnerSnkPdoCnt; i ++)
        Serial.print(usbpd0.iPartnerSnkPdo[i].val);

// Get the partner port source PDO capability list
// Prototype: getPartnerSrcPdo(void)
usbpd0.getPartnerPortSrcPdo( );
while (!usbpd0.iPartnerSrcPdoUpdated) { } // Delay for a while for the USB PD traffic to be accomplished
for (byte i = 0; i < usbpd0.iPartnerSrcPdoCnt; i ++)
        Serial.print(usbpd0.iPartnerSrcPdo[i].val);
```

The status of if any USB Type-C device is attached to the port, its plugin orientation, if any USB PD is connected to the port, power role of the port after the power negotiation, and the type of the attached/connected device. The example code shows the example of how-to get the status of the port.

```
// Get the USB-C port 0 status, available for both CY7113 and PMGDuino
// Prototype:usbpd0.updateStatus(void)
usbpd0.updateStatus( );

if (usbpd0.attach) { /* A device is attached to the port */ }
if (usbpd0.polarity) { /* the orientation is non-flip */ }
if (usbpd0.contractExisted) { /* A USB PD is connected to the port */ }

if (usbpd0.powerRole == CY_PD_PRT_ROLE_SOURCE) { /* port is power source role */ }
else if (usbpd0.powerRole == CY_PD_PRT_ROLE_SINK) { /* port is power sink role */ }
else { /* Not applicable */ }

if ( usbpd0.attachDeviceType == 1) { /* Hub device type */}
else if (usbpd0.attachDeviceType == 2) { /* Peripheral device type */}
else if (usbpd0.attachDeviceType == 3) { /* Power sink device type or passive cable */}
else if (usbpd0.attachDeviceType == 4) { /* Active cable */}
else if (usbpd0.attachDeviceType == 5) { /* Alternative Mode Accessory device type */}
else if (usbpd0.attachDeviceType == 6) { /* VCONN powered device type */}
else { /*Unknown device type */}

// Example of getting the USB-C port 1 status, only available for PMGDuino
usbpd1.updateStatus( );
```

User/developer can request the power role change in the run-time (only available for PMGDuino) by calling function *doPwrRoleSwap*, after calling this function the port power role swap negotiation will be proceeded and if the power role swap is accepted by the partner port, the new power negotiation will be proceeded. The example code can be found below.

```
usbpd0.updatStatus( ); // check the port power role
if (usbpd0.powerRole == 0)
{
        // if the current power role is as sink power role, then request the port power role changing
        // to source power role
        // Prototype: doPwrRoleSwap(void)
        usbpd0.doPwrRoleSwap( );
}
```

User/developer can register USB Type-C and USB PD events and hook the callback function for the customization implementations.

User/developer can use function getEprModeActive to retrieve if the EPR mode is active. The example code is:

```
// Prototype: bool getEprModeActive(void)
if (usbpd0.getEprModeActive( ))
        { /* EPR mode is entered and active */}
else
        { /* EPR mode is not active */ }
```

Functions *registerEvent* and *unRegisterEvent* is designed for the user/developer to register USB Type-C and USB PD events and hook the callback function to the specific event. The example code of hoe-to use *regiserEvent* function is shown as beneath.

```
// Callback function of the USB PD negotiation completet
void usbPdNegotiationComplete(byte portIndex)
{
        // Implement your customization code here
        if (portIndex == 0)
                { /* customization for port 0 */}
        else
                { /* customization for port 1 */}
}

// Register USB PD power negotiation complete event
// Prototype: usbpd0.register(byte eventIndex, void (*func)(byte portIndex))
usbpd0.register(APP_EVT_PD_CONTRACT_NEGOTIATION_COMPLETE, usbPdNegotiationComplete);
// :
// :
// unregister USB PD power negotiation complete event if it is no longer required anymore
usbpd0.unRegisterEvent(APP_EVT_PD_CONTRACT_NEGOTIATION_COMPLETE);
```

User/developer should be aware of the implementation of callback function for the USB Type-C and USB PD events shouldn't take too long or block the MCU execution as it is related to the timing requirements of USB PD specification.

The following table lists the USB Type-C and USB PD events that user/developer can use for their customizations.

Table 20 USB Type-C and USB PD event list

Item	Event Name	Description	Note
1	APP_EVT_UNEXPECTED_VOLTAGE_ON_VBUS	Unexpected high voltage seen on VBUS	
2	APP_EVT_TYPE_C_ERROR_RECOVERY	Type-C error recovery initiated	
3	APP_EVT_CONNECT	Type-C connect detected	
4	APP_EVT_DISCONNECT	Type-C disconnect(detach) detected	
5	APP_EVT_HARD_RESET_RCVD	Hard reset received	
6	APP_EVT_HARD_RESET_COMPLETE	Hard reset processing completed	
7	APP_EVT_PKT_RCVD	New PD message received	
8	APP_EVT_PR_SWAP_COMPLETE	PR_SWAP process completed	
9	APP_EVT_PD_CONTRACT_NEGOTIATION_COMPLETE	Contract negotiation completed	
10	APP_EVT_VBUS_OVP_FAULT	VBUS over voltage fault detected	
11	APP_EVT_VBUS_OCP_FAULT	VBUS over current fault detected	
12	APP_EVT_TYPEC_STARTED	PD port enable (start) completed	
13	APP_EVT_VBUS_UVP_FAULT	VBUS undervoltage fault detected	
14	APP_EVT_VBUS_SCP_FAULT	VBUS short-circuit fault detected	
15	APP_EVT_TYPEC_ATTACH	Type-C attach event	
16	APP_EVT_PD_SINK_DEVICE_CONNECTED	Sink device connected	
17	APP_EVT_PR_SWAP_ACCEPTED	PR-SWAP accepted by EZ-PD(TM) PMG1 or port partner	
18	APP_EVT_EPR_MODE_ENTER_RECEIVED	EPR mode enters command received from sink port	
19	APP_EVT_EPR_MODE_ENTER_SUCCESS	EPR mode enter successfully, source response with enter succeeded	
20	APP_EVT_EPR_MODE_ENTER_FAILED	EPR mode enter failed, source response with fail	
21	APP_EVT_EPR_MODE_EXIT	EPR mode exit received	

4 Example codes

There are example codes available for the USB Type-C and USB PD applications, please find them in the GitHub page for the reference.

- Example: usb_pd_sink_pdo_led_indication

  This example code is demonstrated the sink capabilities and retrieve the power from the external USB PD power source.

  The Built-in LED will blink in the specific period depended on the accepted sink PDO when the USB-C port 0 is connected to the external USB PD power source.

  The PMG1 kit system power source is recommended to be supplied by the USB-C port 0 V_{BUS} (short pin 7-8 of J12 if using PMGDuino kit).

  This example is available for bot CY7113 and PMGDuino kit.

- Example: usb_pd_simple_snk_pdo_example_port0 and usb_pd_simple_snk_pdo_example_port1

  These examples are demonstrated the very simple sink PDO capabilities and retrieve the power from the external USB PD power source.

  Example usb_pd_simple_snk_example_port0 is available for both CY7113 and PMGDuino; if PMGDuino kit is used, it is recommended to souring the system power from USB-C port 0 V_{BUS} (short pin 7-8 of J12).

  Example usb_pd_simple_snk_pdo_example_port1 is only available for PMGDuino; it is recommended to supply the system power from USB-C port 1 V_{BUS} (short pin 5-6 of J12) if this example is deploying.

- Example: usb_pd_simple_src_example_port0 and usb_pd_simple_src_example_port1

  These examples are demonstrated the very simple source PDO capabilities and supply the power to the external USB PD device.

  Both examples are only available for PMGDuino kit only.

  When working with these examples, RT6190 PMIC board is recommended to be deployed as the power source.

  It is recommended that supply the system power by the V_{BUS} of either USB-C port 0 when working with example usb_pd_dynamic_src_pdo_control_port0 (short pin 9-10 of J12) or USB-C port 1 when working with example usb_pd_dynamic_src_pdo_control_port1 (short pin 11-12 of J12)

  When working with example usb_pd_simple_src_example_port0, RT6190 PMIC board should be connected to J7 of PMGDuino board, and short pin 1-2 of jumper J3 and pin 2-3 of jumper J1 of RT6190 PMIC board for selecting the I^2C device address to 0x2C.

  When working with example usb_pd_simple_src_example_port1, RT6190 PMIC board should be connected to J10 of PMGDuino board, and short pin 2-3 of jumper J3 and pin 2-3 of jumper J1 of RT6190 PMIC board for selecting the I^2C device address to 0x2D.

  An external power source should be connected to header J2 of RT6190 PMIC board and the recommended input voltage range should be 5V-28V, the current capability should be depended on the application (as the maximum current claimed in the source PDOs).

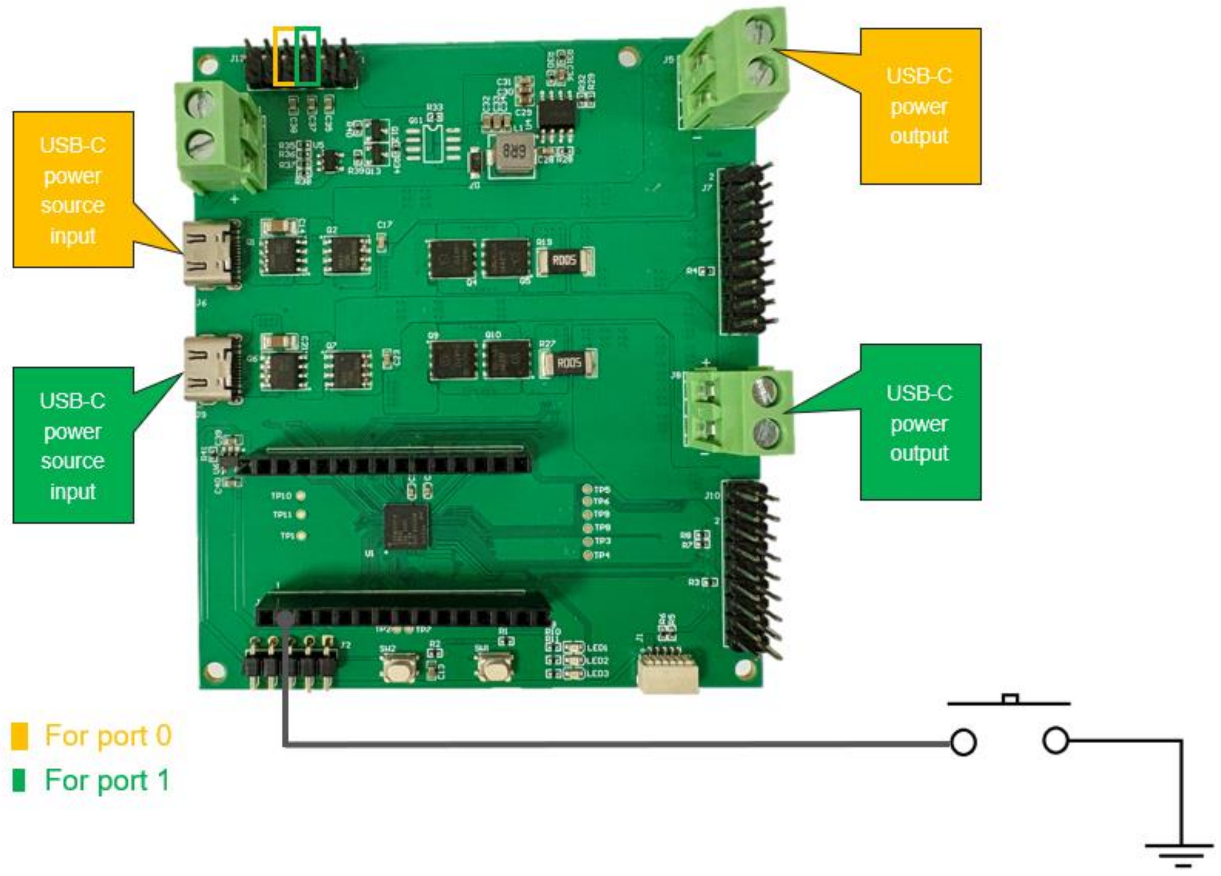
- Example: usb_pd_dynamic_snk_pdo_control_port0 and usb_pd_dynamic_snk_pdo_control_port1

  These examples are demonstrated the capabilities of changing the sink PDO in run-time, and based on the present sink PDO, retrieve the power from the external USB PD power source.

  Example usb_pd_dynamic_snk_pdo_control_port0 is available for both CY7113 and PMGDuino; it is recommended to supply the system power from V_{BUS} of USB-C port 0 (short pin 7-8 of J12 if using PMGDuino kit).

  Example usb_pd_dynamic_snk_pdo_control_port1 is available for PMGDuino only; it is recommended to supply the system power from V_{BUS} of USB-C port 1 (short pin 5-6 of J12 if using PMGDuino kit).

  A external button should be connected to digital pin D15 and another pin of this button should be grounding to make it as a low active trigger input signal; once the button is clicked, the sink PDO will be changed and the voltage change retrieved from the external USB PD power source can be measured from head J5 for example usb_pd_dynamic_snk_pdo_control_port0 and J8 for example usb_pd_dynamic_snk_pdo_control_port1 respectively.



- Example: usb_pd_dynamic_src_pdo_control_port0 and usb_pd_dynamic_src_pdo_control_port1

  These examples demonstrated the capabilities of changing source PDO in run-time and supply the power to the external USB PD device depended on present source PDO and the request from the external USB-PD device.

  These examples are only available for PMGDuino kit.

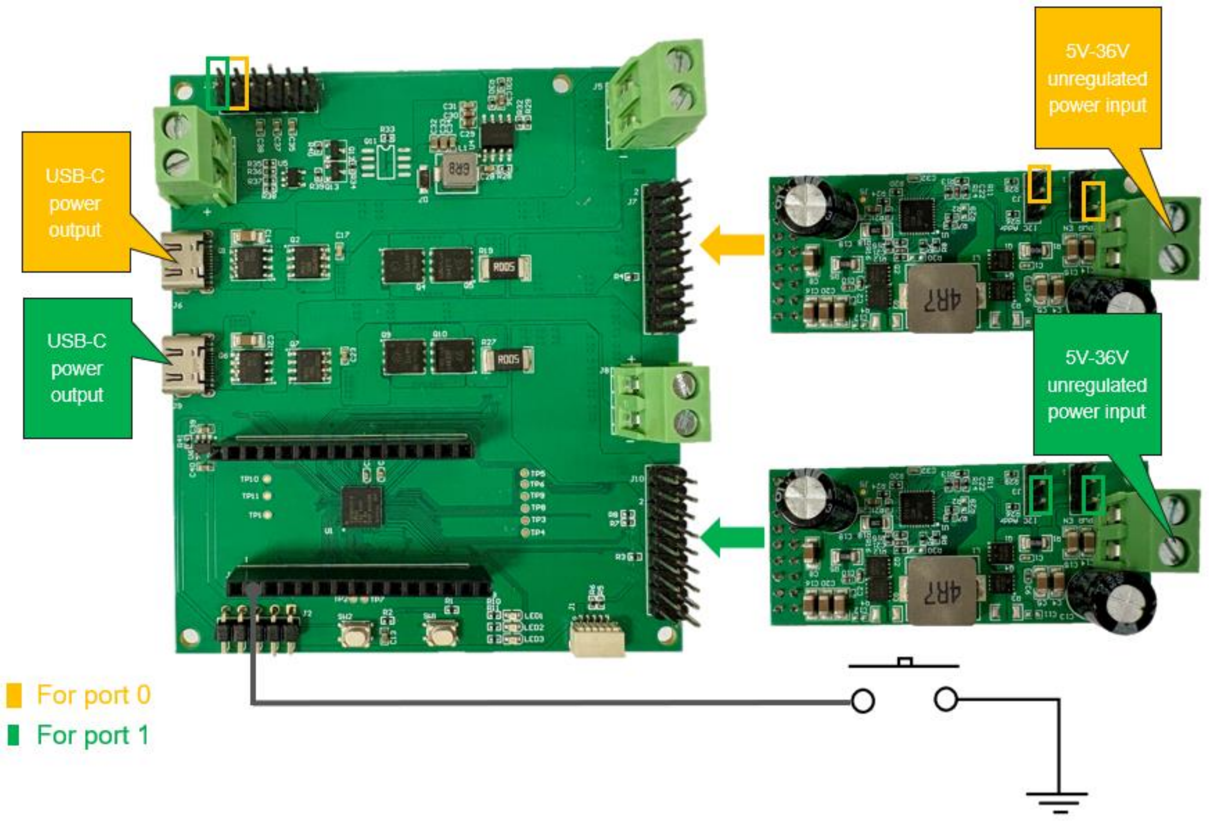
  When working with these examples, RT6190 PMIC board is recommended to be deployed as the power source.

It is recommended that supply the system power by the V_{BUS} of either USB-C port 0 when working with example usb_pd_dynamic_src_pdo_control_port0 (short pin 9-10 of J12) or USB-C port 1 when working with example usb_pd_dynamic_src_pdo_control_port1 (short pin 11-12 of J12).

When working with example usb_pd_dynamic_src_pdo_control_port0, RT6190 PMIC board should be connected to J7 of PMGDuino board, and short pin 1-2 of jumper J3 and pin 2-3 of jumper J1 of RT6190 PMIC board for selecting the I^{2}C device address to 0x2C.

When working with example usb_pd_dynamic_src_pdo_control_port1, RT6190 PMIC board should be connected to J10 of PMGDuino board, and short pin 2-3 of jumper J3 and pin 2-3 of jumper J1 of RT6190 PMIC board for selecting the I^{2}C device address to 0x2D.

An external power source should be connected to header J2 of RT6190 PMIC board and the recommended input voltage range should be 5V-28V, the current capability should be depended on the application (as the maximum current claimed in the source PDOs).


5V-36V unregulated power input
USB-C power output
USB-C power output
5V-36V unregulated power input
For port 0
For port 1

- Example: usb_pd_get_partner_pdo

  This example shows the capabilities of retrieving the source and sink PDOs from the partner port.

  This example is available for both CY7113 and PMGDuino.

  The system power source is recommended being supplied from V_{BUS} of USB-C port 0 (short pin 7-8 if using PMGDuino)

5 Appendix

5.1 References

- EZ-PD™ PMG1 product information
- PMG1-S3 CYPM1311/CYPM1321 data sheet
- PMG1 Duino Wiki page
- PMG1-S3 CY7113 kit schematic
- PMGDuino kit schematic
- RT6190 data sheet
- RT6190 PMIC board schematic
- ModusToolbox™ home page
- PMG1 Duino V0.1.2 release
- PMG1 Duino example codes

5.2 Contributors

- William Chuang
- Asurada Ko
- Karen Hsu
- Carl Liu
- Joshua Chen
- Jason Liu
- Kai Zheng

Revision history

Document version	Date of release	Description of changes
V0.1	Jan. 17, 2025	Version V0.1 based on PMG1 Duino V0.1.2

Published by

Infineon Technologies AG

Do you have a question about this document?

Email: william.chuang@infineon.com