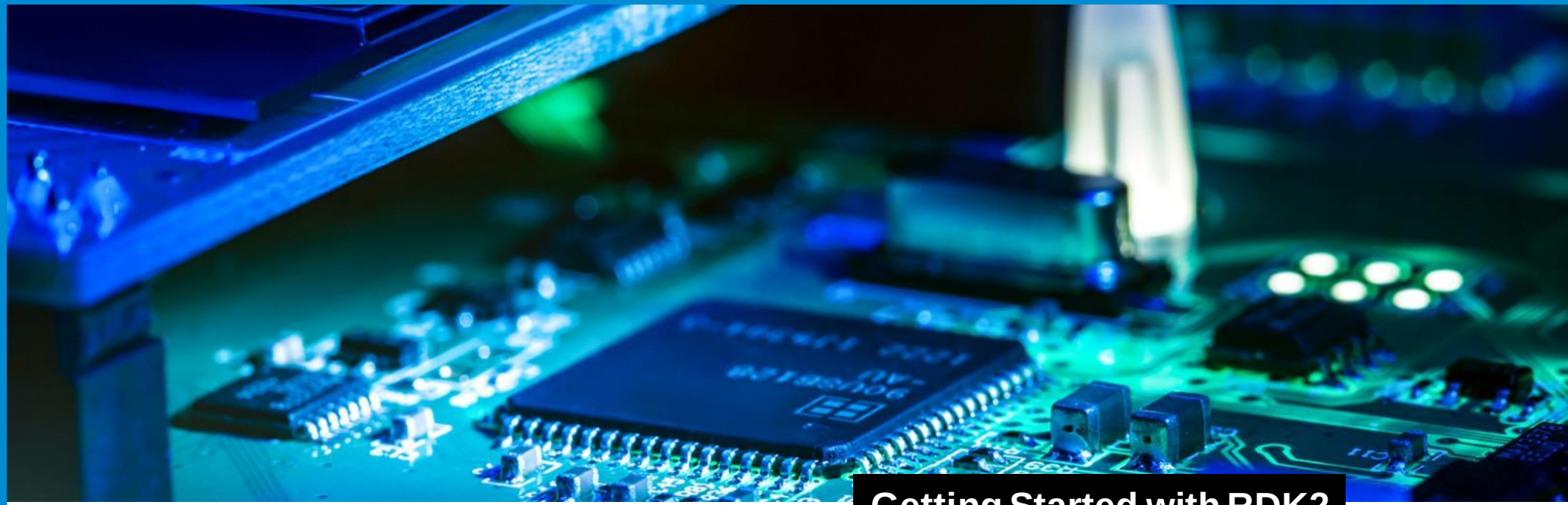
RUTRONIK
ELECTRONICS WORLDWIDE



Getting Started with RDK2

Getting Started with CY8C6245AZI-S3D72 development platform – **RDK2** „*RUTDevKit-PSoC62*“

Registration & Download

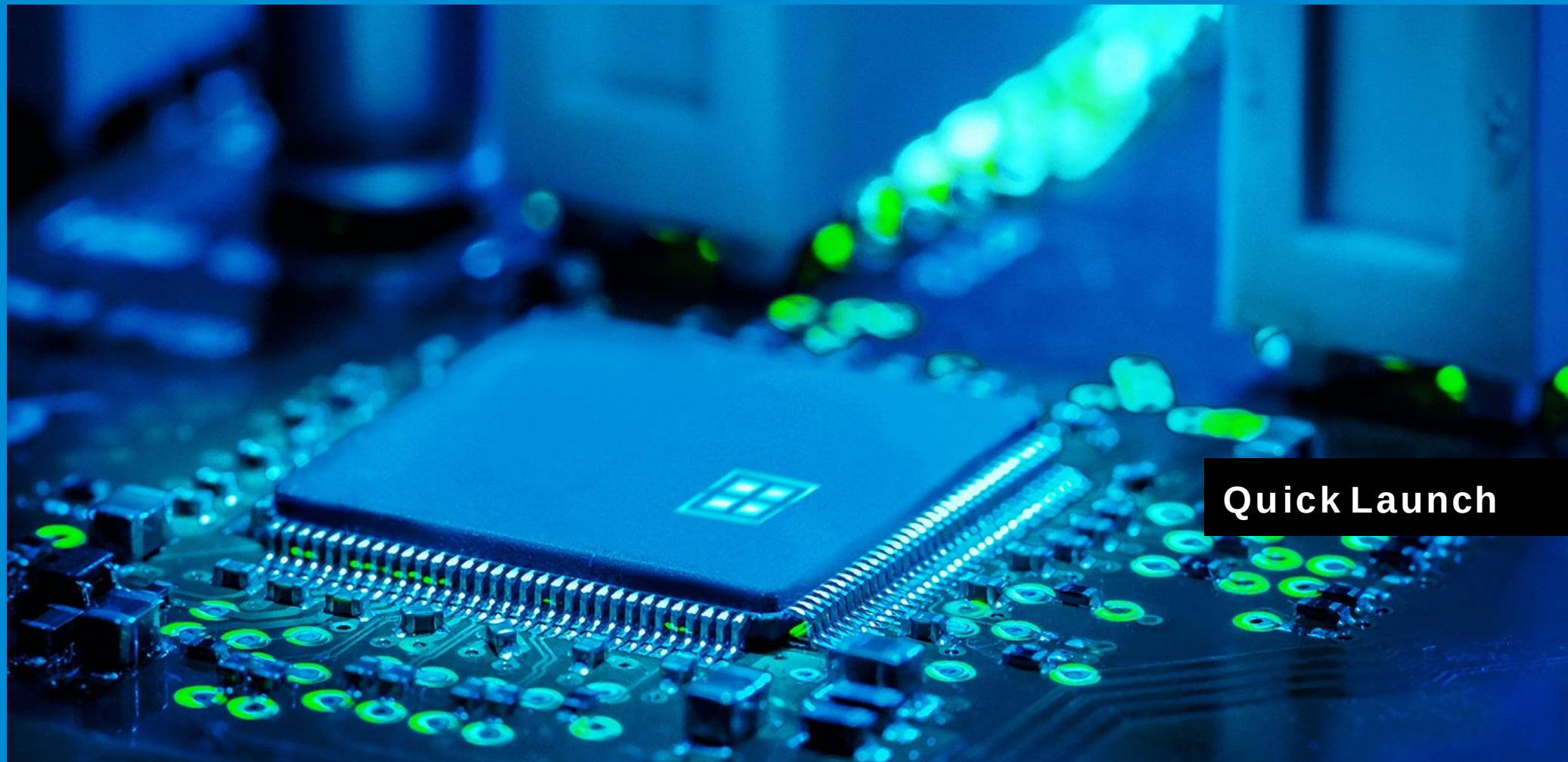
1.) Register or/and login to the Infineon website, press on „myInfineon“ tab.
https://www.infineon.com

2.) Download and install the latest ModusToolbox™ software.

3.) Get all the supported firmware examples including the BSP from the RDK2 homepage. Press the „Download Area“ and login/register to access the data.
https://www.rutronik.com/development-stories/devstories-login/

4.) [Optional] Download and install yours prefered terminal emulator, for example: PuTTY, Tera Term, etc.

RUTRONIK
ELECTRONICS WORLDWIDE

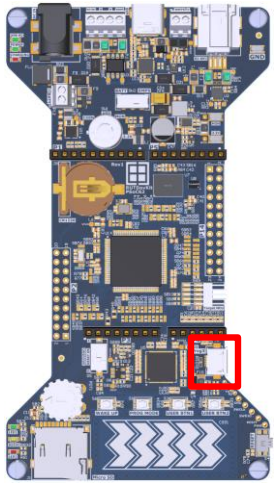


Quick Launch

Connect the RDK2

Connect the RDK2 to yours PC.

Ensure the switch SW1 is set to “3.3V” position



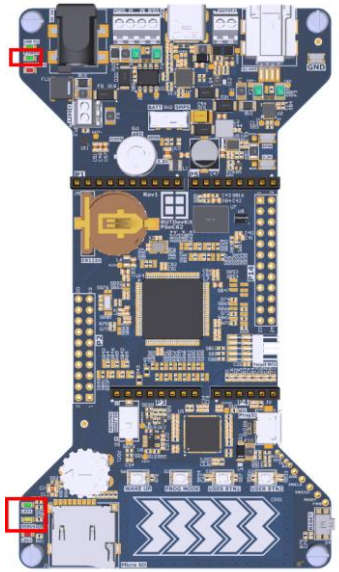
Look for the Micro USB socket with a marking “KitProg3”

Have Micro USB Cable - A to Micro B

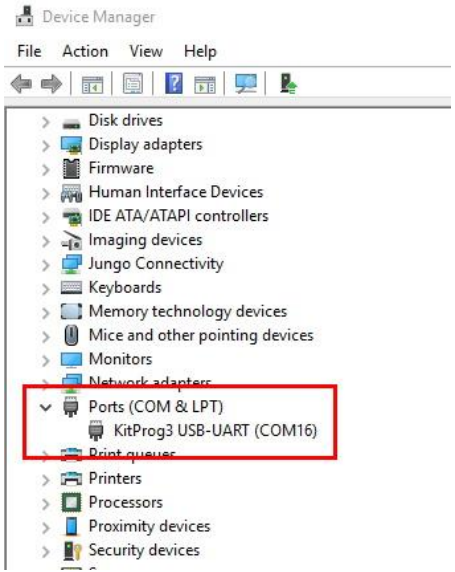
Connect it with yours PC

Connect the RDK2

Check if the RDK2 is ready.



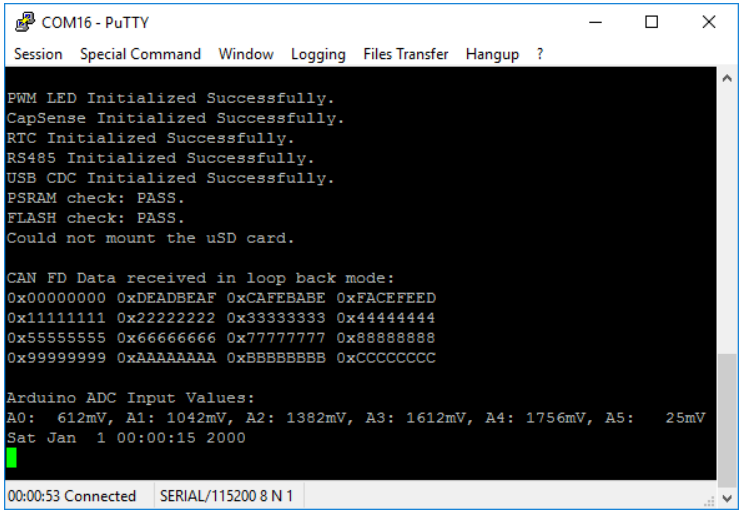
“POWER” and “DEBUG” LEDs should shine constantly. The LED1 reacts to the touch on the slider.



The “KitProg3” must be seen in the “Device Manager” window.

All new RDK2s print out hardware test results to the KitProg3 serial terminal (115200 bit/s).

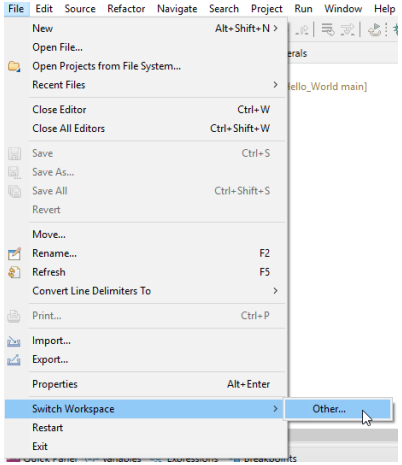
Press the RESET on the RDK2 if necessary.



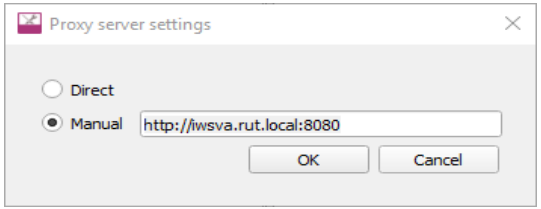
Working with the ModusToolbox and Rutronik PC

If you are working with yours personal PC, (not the Rutronik provided Laptop PC) please skip this setup.

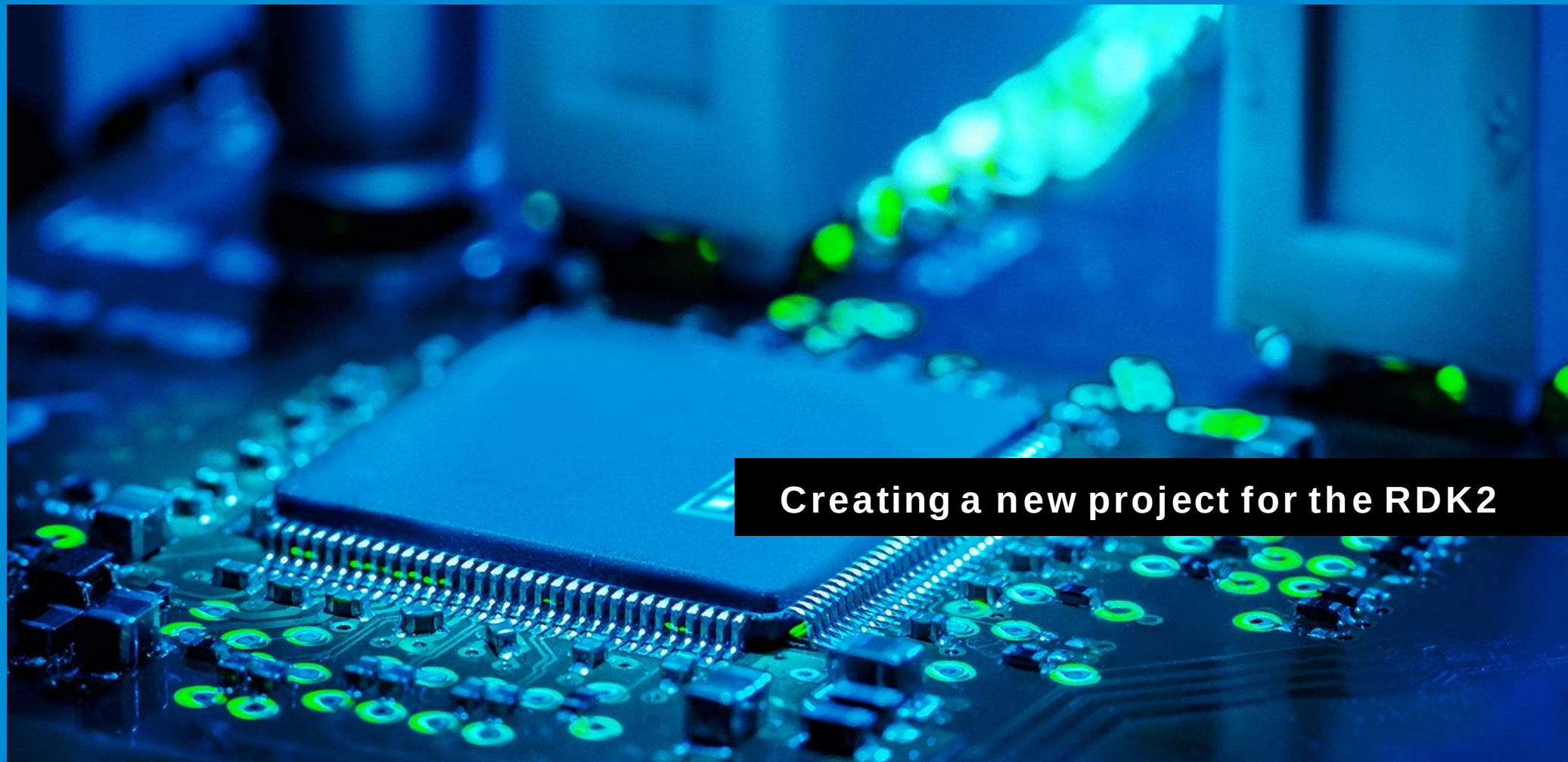
1.) It is recommended to change the default workspace directory to: C:\Users\user_name\modus_workspace.



2.) Open the File→New→ "ModusToolbox Application" → Settings → Proxy server settings and enter the proxy address: http://iwsva.rut.local:8080



RUTRONIK
ELECTRONICS WORLDWIDE



Creating a new project for the RDK2

Creating new projects using “RutDevKit-PSoC62-BSP”

RUTRONIK ELECTRONICS WORLDWIDE

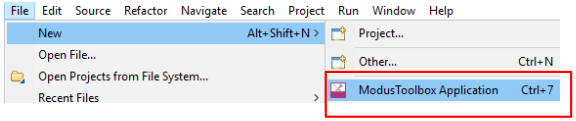
1.) Have the “RutDevKit-PSoC62-BSP” in yours workspace directory.

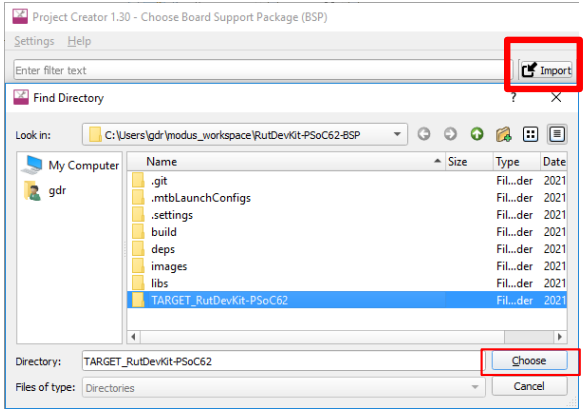
2.) Click: File → New → ModusToolbox Application.

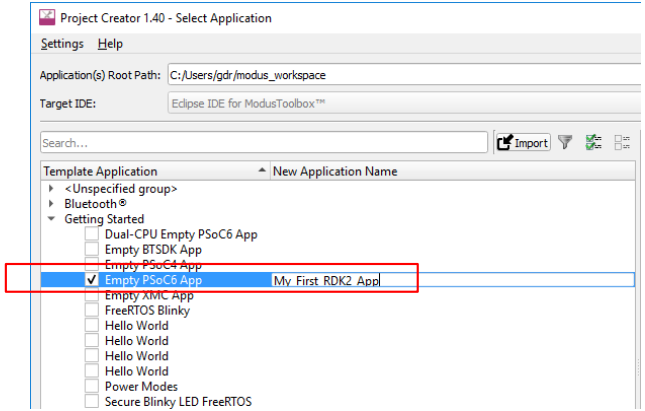
3.) Click: Import and select “TARGET_RutDevKit-PSoC62” folder in the “RutDevKit-PSoC62-BSP” and Click “Choose” and “Next”.

4.) Select “Empty PSoC6 App”, rename it and Click on “Create”.

5.) Wait until project creation is finished.



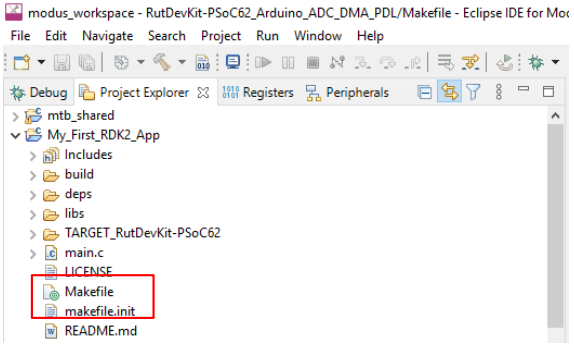




Creating new projects using “RutDevKit-PSoC62-BSP”

RUTRONIK ELECTRONICS WORLDWIDE

6.) Modify the “Makefile” to include essential library components and disable code optimisation*



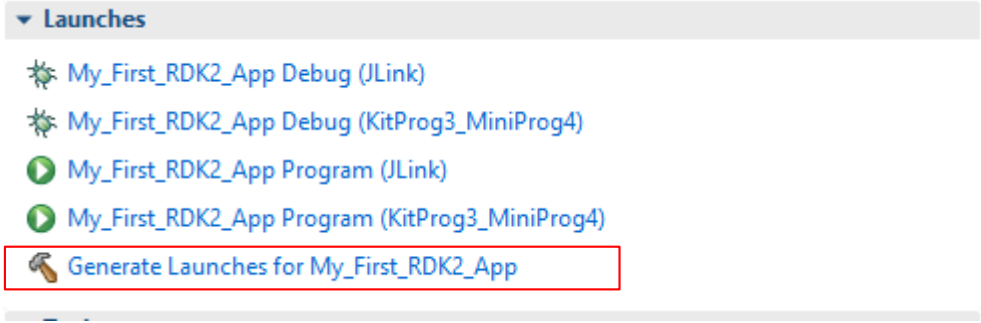
CONFIG=Costum

DEFINES += COMPONENT_CAT1

DEFINES += COMPONENT_CAT1A

CFLAGS =-O0

7.) Press “Generate Launches for (project name”) in Quick Panel

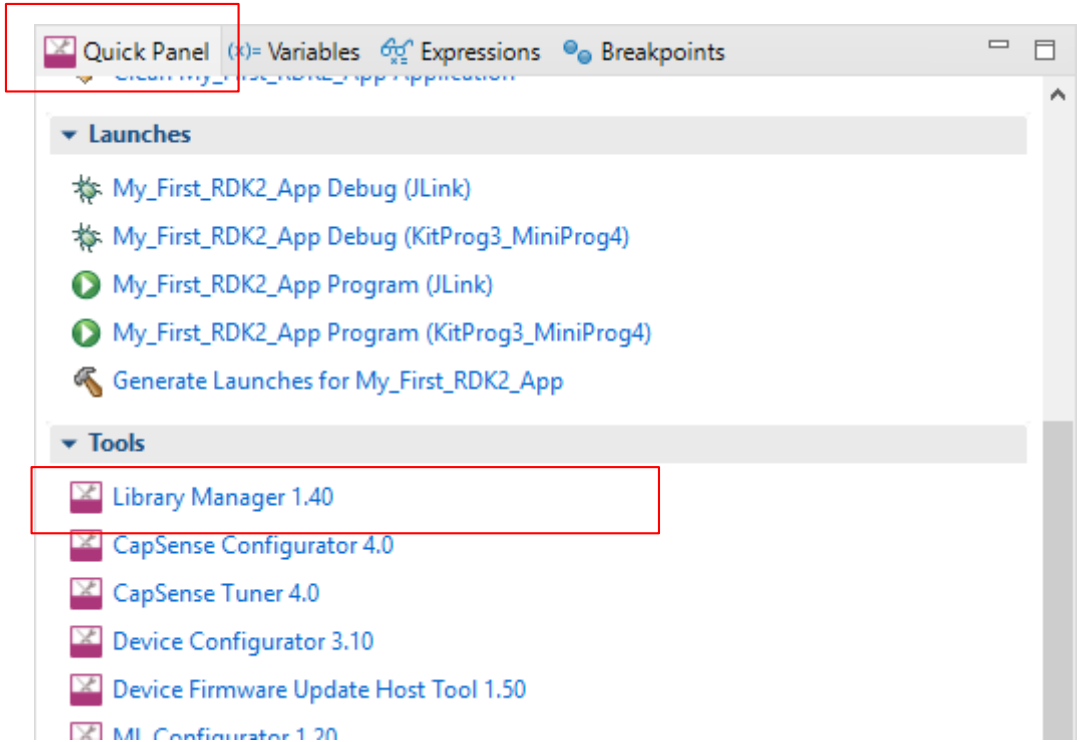


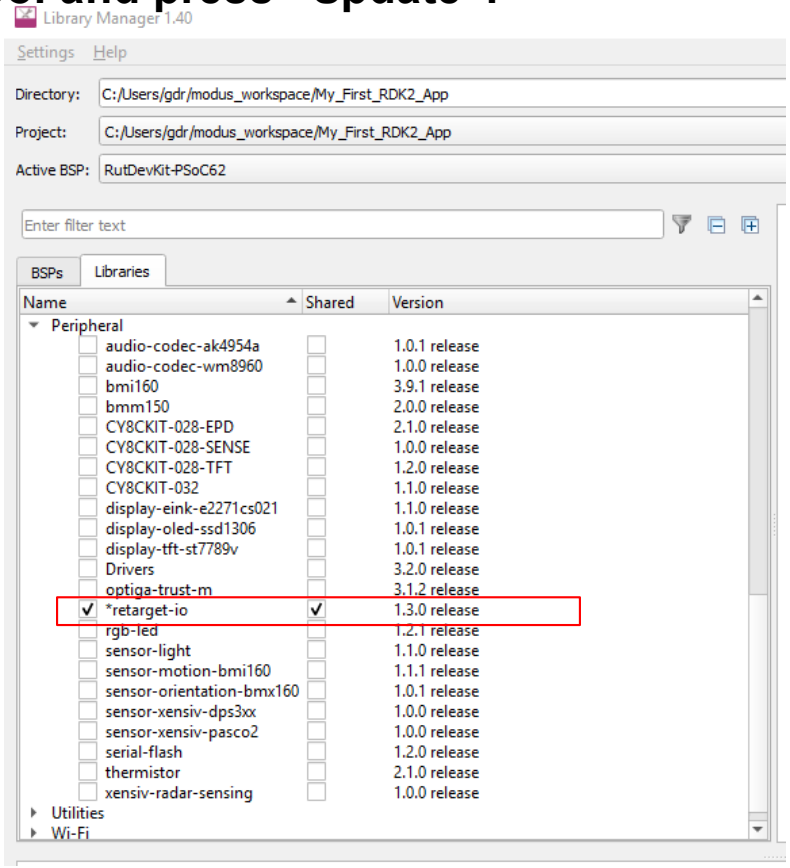
***only for debugging, learning and demo purposes. Normally, code optimisations should never be disabled.**

Creating new projects using “RutDevKit-PSoC62-BSP”

RUTRONIK ELECTRONICS WORLDWIDE

9.) Select “retarget-io” library in a “Library Manager” tool and press “Update”.

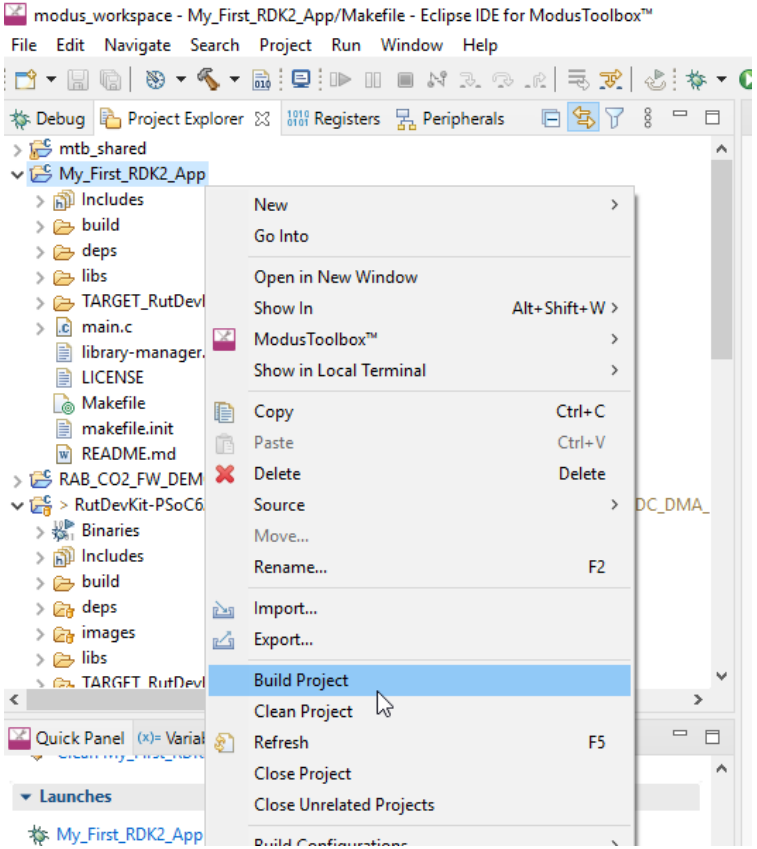


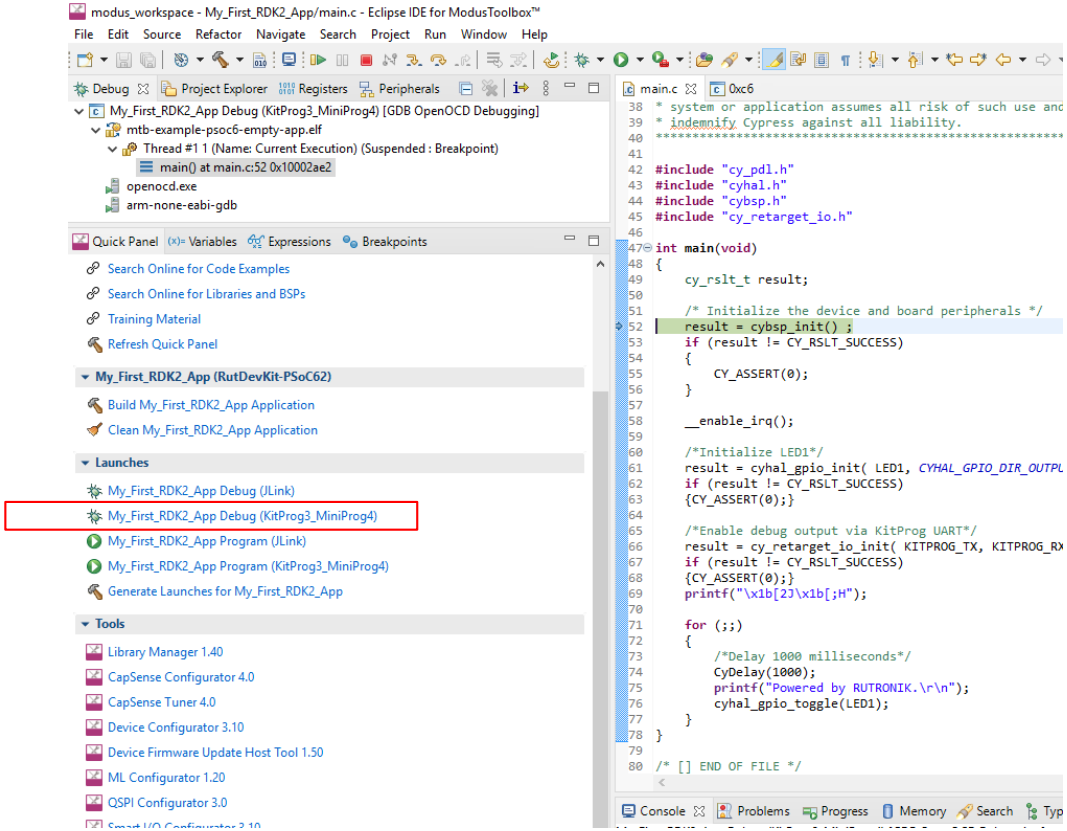


9.) Copy/Paste and save the code example to the “main.c” file.

```
#include "cy_pdl.h"
#include "cyhal.h"
#include "cybsp.h"
#include "cy_retarget_io.h"

int main(void)
{
    cy_rslt_t result;

    /* Initialize the device and board peripherals */
    result = cybsp_init() ;
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }

    __enable_irq();

    /*Initialize LED1*/
    result = cyhal_gpio_init( LED1, CYHAL_GPIO_DIR_OUTPUT, CYHAL_GPIO_DRIVE_STRONG, CYBSP_LED_STATE_OFF);
    if (result != CY_RSLT_SUCCESS)
    {CY_ASSERT(0);}

    /*Enable debug output via KitProg UART*/
    result = cy_retarget_io_init( KITPROG_TX, KITPROG_RX, CY_RETARGET_IO_BAUDRATE);
    if (result != CY_RSLT_SUCCESS)
    {CY_ASSERT(0);}
    printf("\x1b[2J\x1b[;H");

    for (;;)
    {
    /*Delay 1000 milliseconds*/
    CyDelay(1000);
    printf("Powered by RUTRONIK.\r\n");
    cyhal_gpio_toggle(LED1);
    }
}
```

10.) Build and Debug the active project.

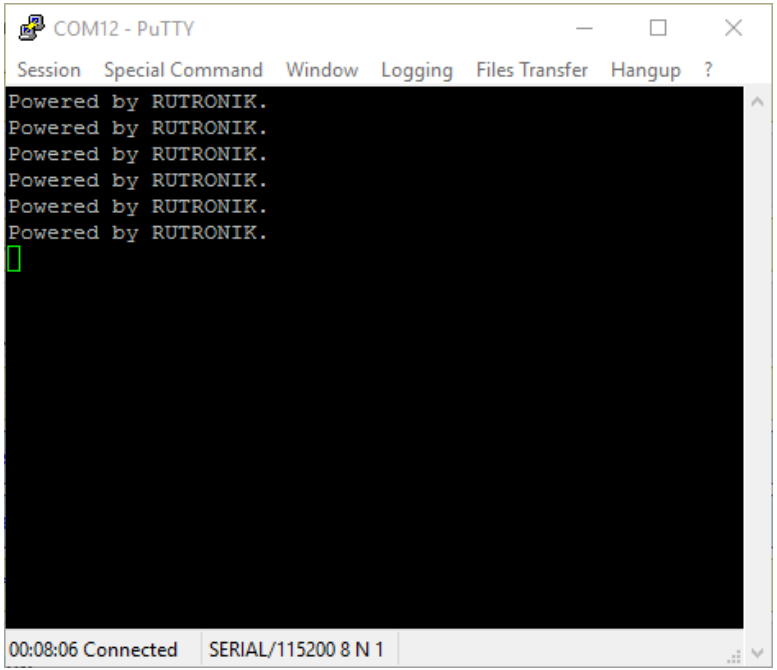




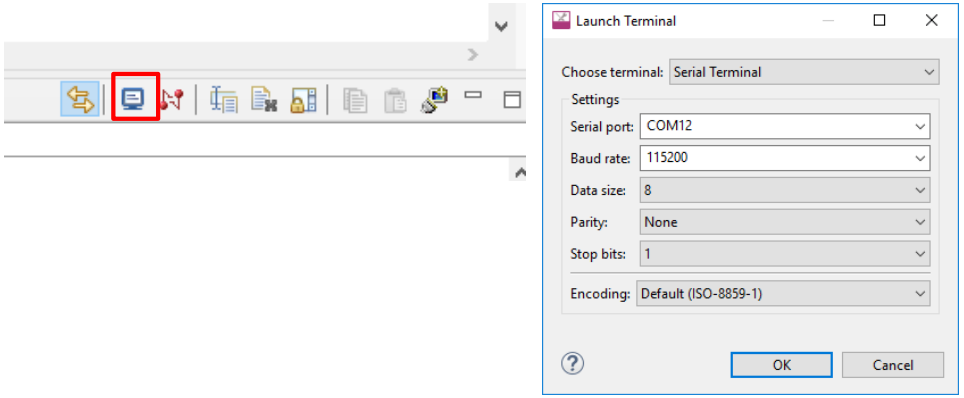
Creating new projects using “RutDevKit-PSoC62-BSP”

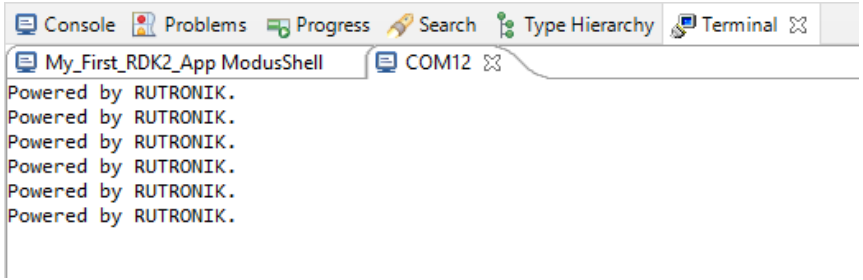
The final result is a blinking LED1 on the RDK2 board and text on the terminal window:

PuTTY Terminal

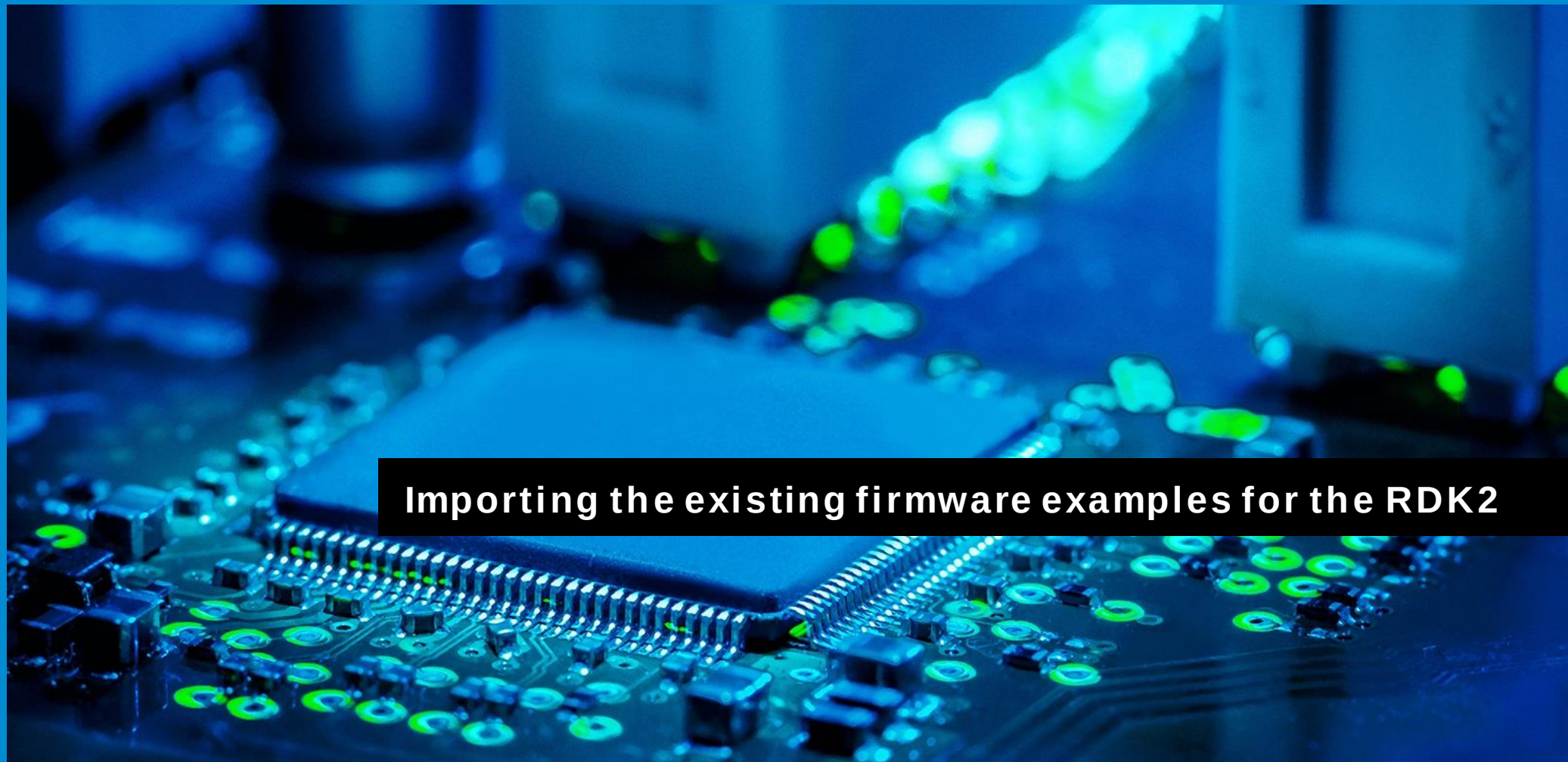


ModusToolbox Terminal





RUTRONIK
ELECTRONICS WORLDWIDE

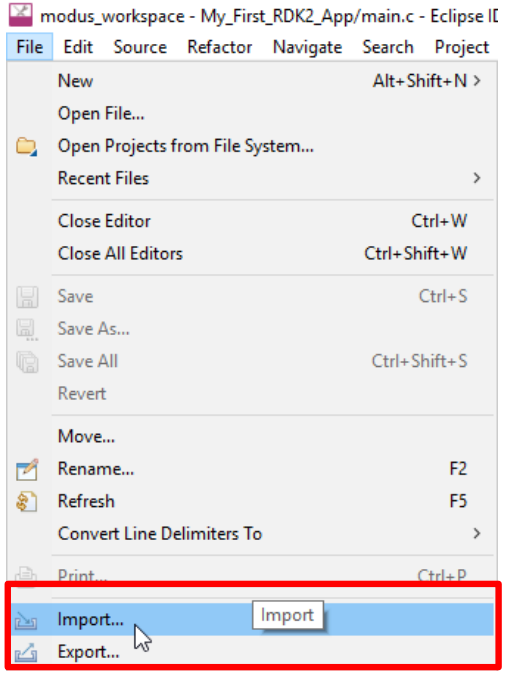


Importing the existing firmware examples for the RDK2

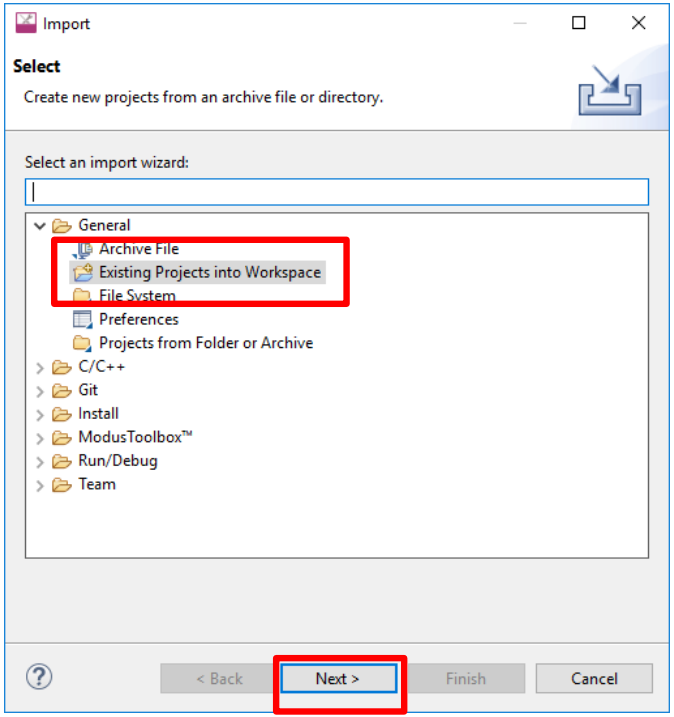
Importing Existing Projects into Workspace

1.) Go: File → Import… → Existing Projects into Workspace → Next.

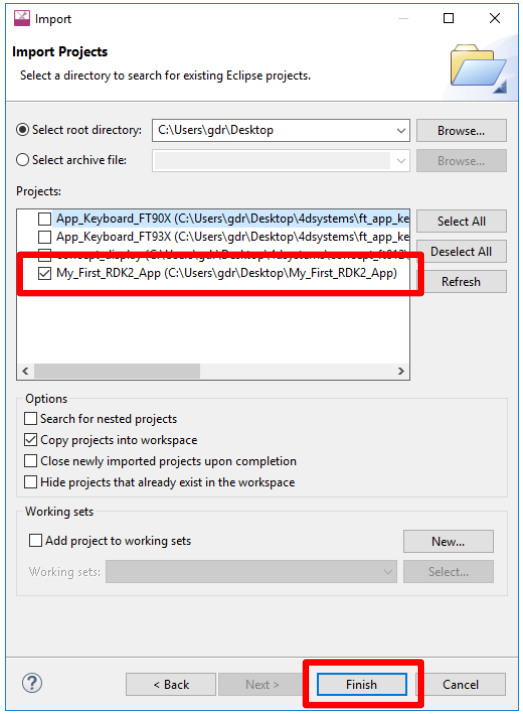
2.) Select a directory and the project to import, select "Copy projects into workspace" then click on "Finish".



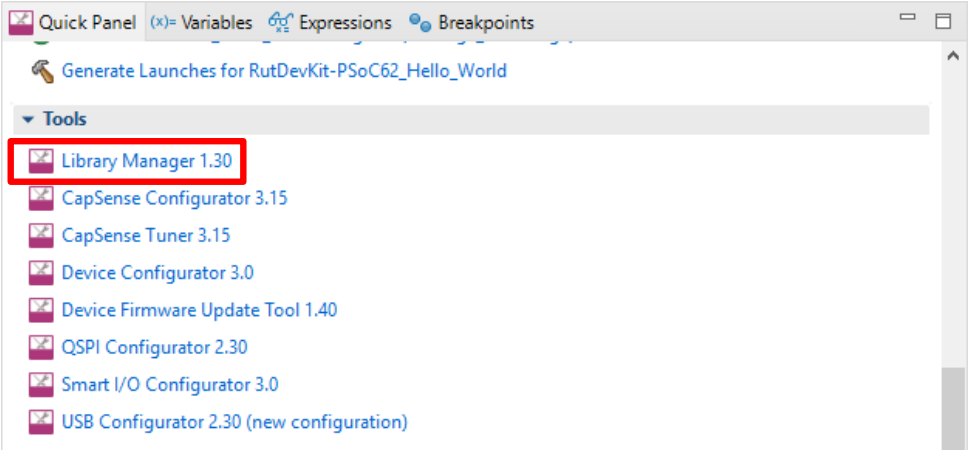
→



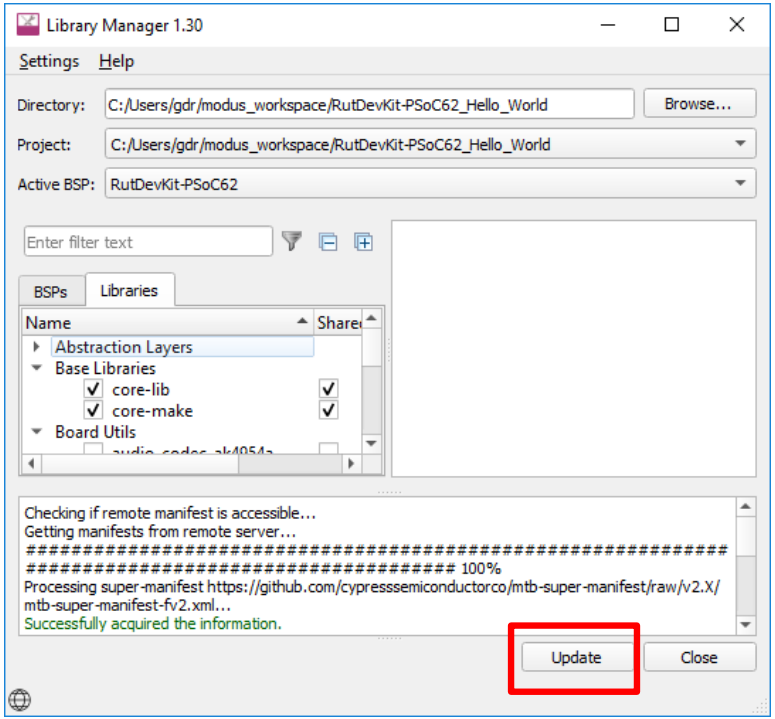
→



3.) Update the libraries using the “Library Manager”.

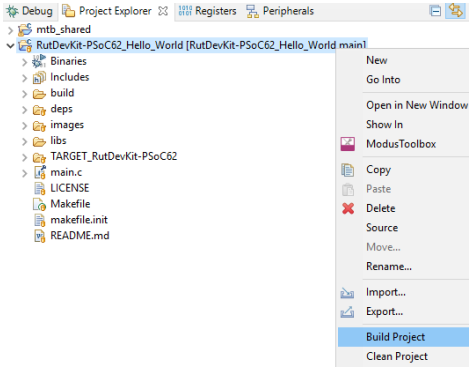


→

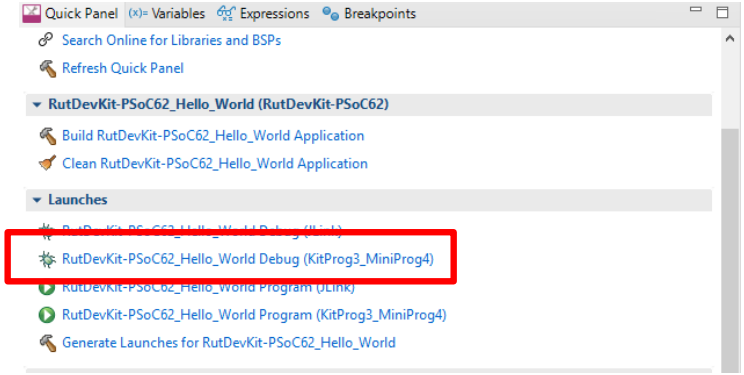


Build and Debug Imported Projects

1.) Build: Right Click on the project and click “Build Project”.

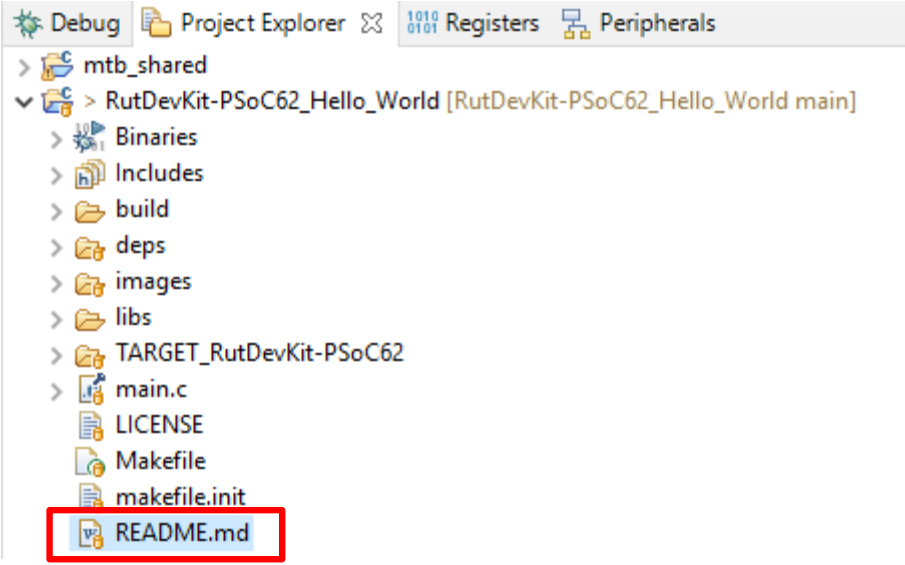


2.) Debug: Click on “KitProg3” debug option in “Quick Panel”.

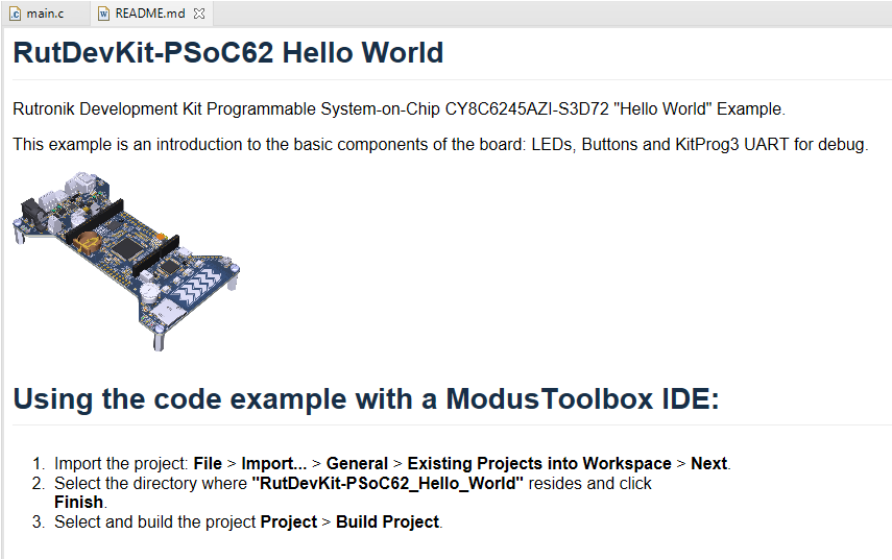


“RutDevKit-PSoC62” README.md

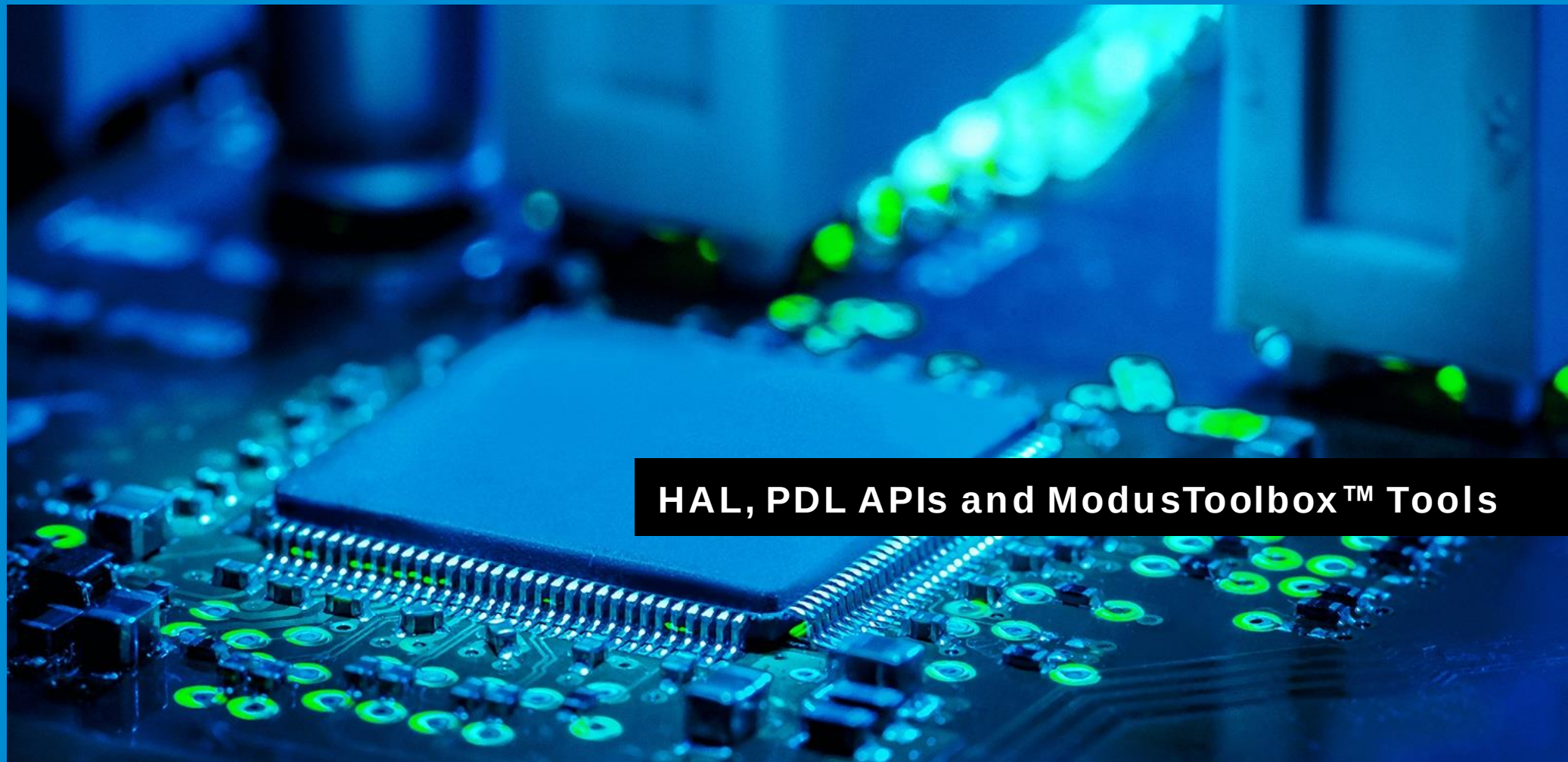
Check the README.md file before starting to explore the code example. You may find important hints or what else is needed to have firmware running properly.



→



RUTRONIK
ELECTRONICS WORLDWIDE



HAL, PDL APIs and ModusToolbox™ Tools

Peripheral Driver Library API

RUTRONIK ELECTRONICS WORLDWIDE

https://infineon.github.io/psoc6pdl/pdl_api_reference_manual/html/index.html

https://infineon.github.io/psoc6pdl/pdl_api_reference_manual/html/index.html

Most Visited | Outdoor | iC-Haus Homepage - ... | Comming Soon - Tur... | ESD – Bridgetek | RutDevKit PSoC64 (De... | Gebäude- & Hausauto... | About RodolfoGL - Inf... | Other Bookmarks

Bookmarks

Search bookmarks

- Bookmarks Toolbar
- Bookmarks Menu
- Other Bookmarks

Infineon + CYPRESS EMBEDDED IN TOMORROW™ PSoC 6 Peripheral Driver Library

Getting started | PDL API Reference | Online Documentation | Search

- PSoC 6 Peripheral Driver Library
  - Introducing PSoC 6
  - Getting started
  - PDL API Reference
  - Online Documentation
  - Additional Resources
  - MISRA-C:2004 Compliance

PSoC 6 Peripheral Driver Library Documentation

Overview

The PDL integrates device header files, startup code, and peripheral drivers into a single package. The PDL supports the PSoC 6 device family. The drivers abstract the hardware functions into a set of easy-to-use APIs. These are fully documented in the PDL API Reference.

The PDL reduces the need to understand register usage and bit structures, thus easing software development for the extensive set of peripherals in the PSoC 6 series. You configure the driver for your application, and then use API calls to initialize and use the peripheral. This documentation provides technical information on each supported peripheral driver, including:

- Configuration considerations
- Macros
- Functions
- Global variables
- Data structures
- Enumerated types

The PDL conforms to the ANSI C99 standard.

Copyright

Hardware Abstraction Layer API

RUTRONIK ELECTRONICS WORLDWIDE

https://infineon.github.io/psoc6hal/html/index.html

https://infineon.github.io/psoc6hal/html/index.html

Most Visited | Outdoor | iC-Haus Homepage - ... | Comming Soon - Tur... | ESD – Bridgetek | RutDevKit PSoC64 (De... | Gebäude- & Hausauto... | About RodolfoGL - Inf... | Other Bookmarks

Bookmarks

Search bookmarks

- Bookmarks Toolbar
- Bookmarks Menu
- Other Bookmarks

infineon + CYPRESS EMBEDDED IN TOMORROW™ Hardware Abstraction Layer (HAL)

Home | HAL API Reference | Search

- Hardware Abstraction Layer (HAL)
  - Hardware Abstraction Layer
  - HAL API Reference

Hardware Abstraction Layer

Overview

The Hardware Abstraction Layer (HAL) provides a high-level interface to configure and use hardware blocks on PSoC MCUs. It is a generic interface that can be used across multiple product families. The focus on ease-of-use and portability means the HAL does not expose all of the low-level peripheral functionality. The HAL can be combined with platform-specific libraries (such as the PSoC 4/6 Peripheral Driver Library (PDL)) within a single application. You can leverage the HAL's simpler and more generic interface for most of an application, even if one portion requires finer-grained control.

To use code from the HAL, the specific driver header file can be included or the top level `cyhal.h` header can be include to allow access to any driver.

API Structure

The API functions for each HAL driver can be divided into the following categories:

- A `_init` function allocates a block (along with any dependent resources), configures it, and enables it.
- A `_free` function disables a block, releases any dependent resources, and marks it as unused.
- Additional functions provide access to block-specific functionality and, in some cases, manipulate a block's configuration after it has been initialized.

The `_init` API functions require a pointer to an instance of a driver-specific type (for example, `cyhal_uart_t`). This instance must be allocated by the application code and passed via pointer into the initialization function, which will populate its contents (this structure enables the HAL to avoid performing any dynamic memory allocation). From an application point of view, these structs function as an opaque handle. The same object must be passed to all subsequent API calls that operate on the same hardware instance. The struct definitions are defined within the platform-specific HAL implementation. Application code should not rely on the specific contents, which is an implementation detail and is subject to change between platforms and/or HAL releases.

Many `_init` functions also have an argument for a pointer to a `cyhal_clock_t` instance. This is an optional argument to enable sharing of clock dividers in large designs (via using `cyhal_clock_init` or `cyhal_clock_allocate` to allocate a shared divider). If a `NULL` value is passed, the init function will allocate a clock divider that is exclusive to that block instance.

Note: Some APIs that manipulate block timing may not be able to support as wide of a range of values when using a shared divider. When a divider is shared the driver cannot unilaterally change the divider's value because that would affect other divider clients. This is not an issue with a dedicated divider.

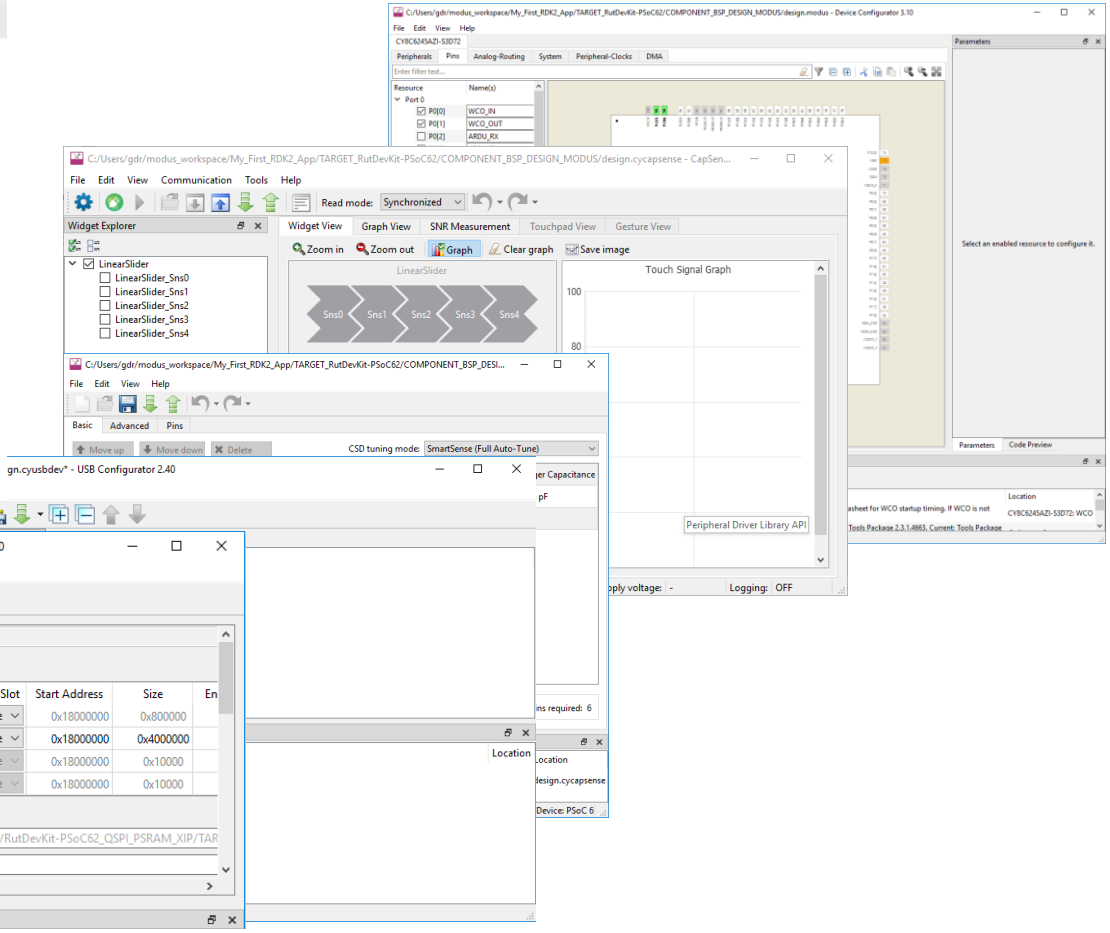
Resource Identification

For peripherals, the HAL generally does not identify block instances directly (for example, by index). Instead, block instances are identified indirectly by specifying the desired pin for each function. The `_init` function selects the block instance that connects to the specified pins. If multiple block instances can connect to the specified pins, the HAL may select any available instance.

ModusToolbox Tools

RUTRONIK ELECTRONICS WORLDWIDE

- Tools
  - Library Manager 1.40
  - CapSense Configurator 4.0
  - CapSense Tuner 4.0
  - Device Configurator 3.10
  - Device Firmware Update Host Tool 1.50
  - ML Configurator 1.20
  - QSPI Configurator 3.0
  - Smart I/O Configurator 3.10
  - USB Configurator 2.40 (new configuration)



Gintaras Drukteinis
Technical Support
Phone : +370 372 45568
eMail : gdr@rutronik.com