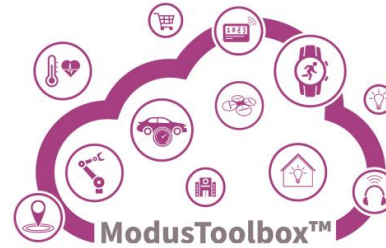




Getting Started with RDK2

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



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.

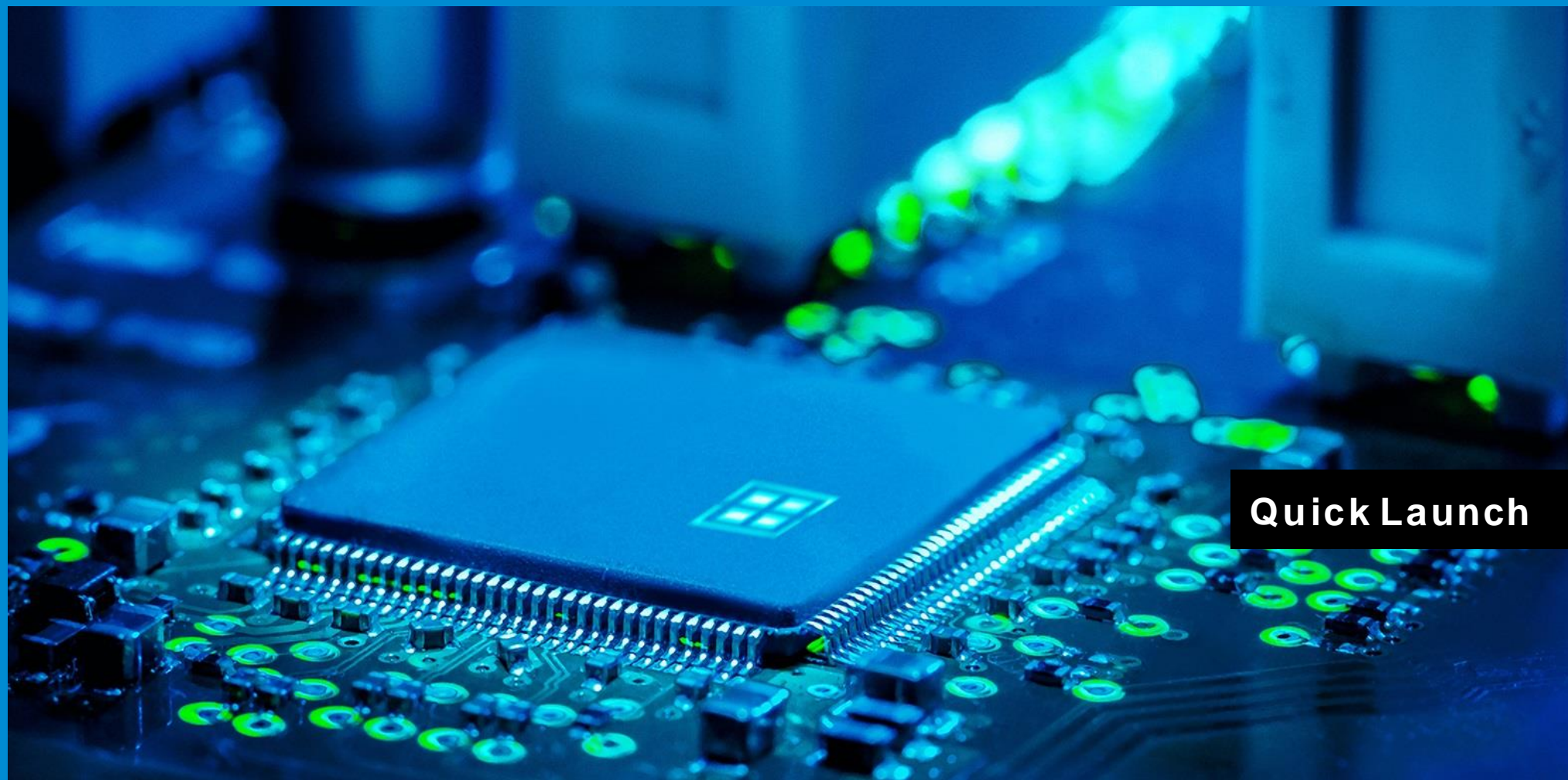
3.) Get all the supported firmware examples including the BSP from the [RDK2](#) homepage. Press the „Download Area“ and login/register to access the data.

<https://www.rutronik.com/development-stories/devstories-login/>

4.) [Optional] Download and install your preferred terminal emulator, for example: [PuTTY](#), [Tera Term](#), etc.



RUTRONIK
ELECTRONICS WORLDWIDE



Quick Launch

Connect the RDK2

Connect the RDK2 to your 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



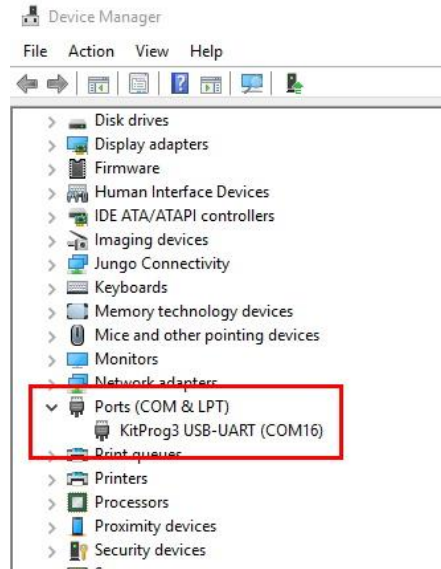
Connect it with your PC

Connect the RDK2

Check if the RDK2 is ready.

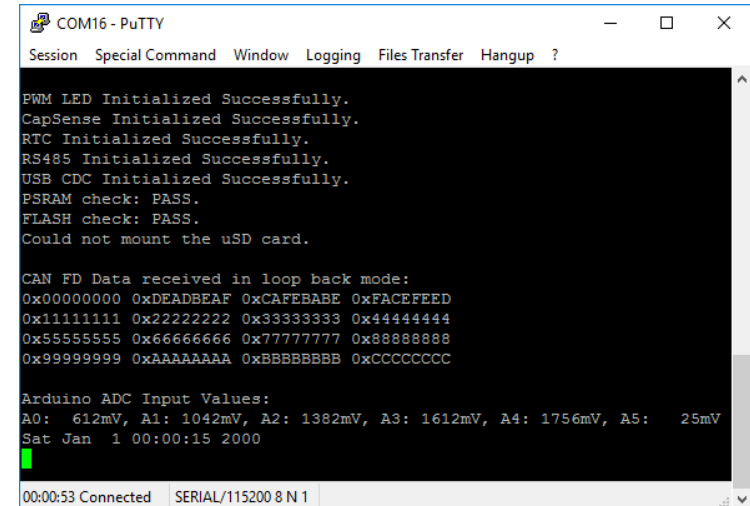


“POWER” and “DEBUG” LEDs should shine constantly. The LED1 reacts to the touch on the slider.



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

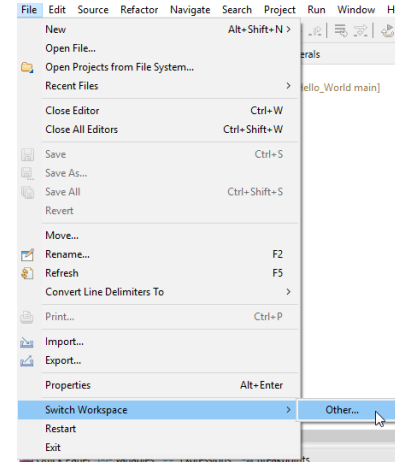
All new RDK2s print out hardware test results to the KitProg3 serial terminal (115200 bit/s). Press the RESET on the RDK2 if necessary.



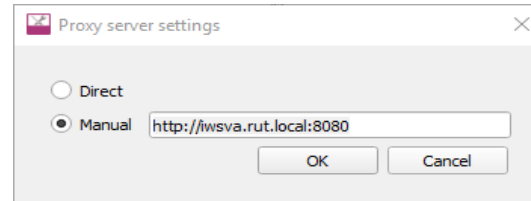
Working with the ModusToolbox and Rutronik PC

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>





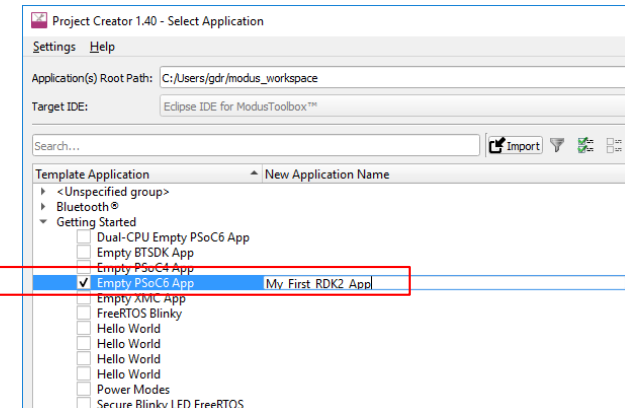
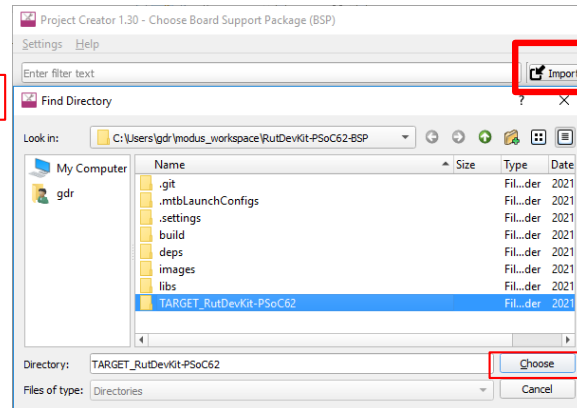
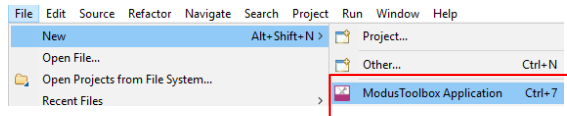
RUTRONIK
ELECTRONICS WORLDWIDE

A close-up photograph of a microchip mounted on a circuit board. The scene is illuminated with a strong blue light, and there are several bright green circular highlights on the board's surface. A black rectangular box is overlaid on the right side of the image, containing white text.

Creating a new project for the RDK2

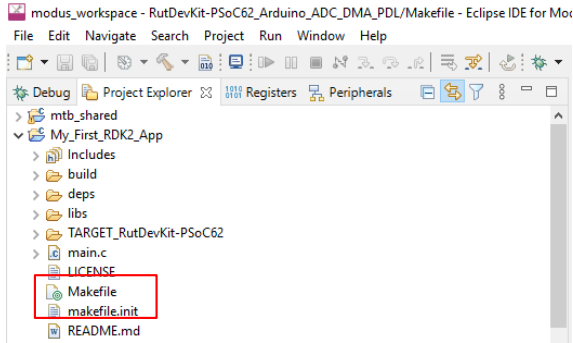
Creating new projects using “RutDevKit-PSoC62-BSP”

- 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.



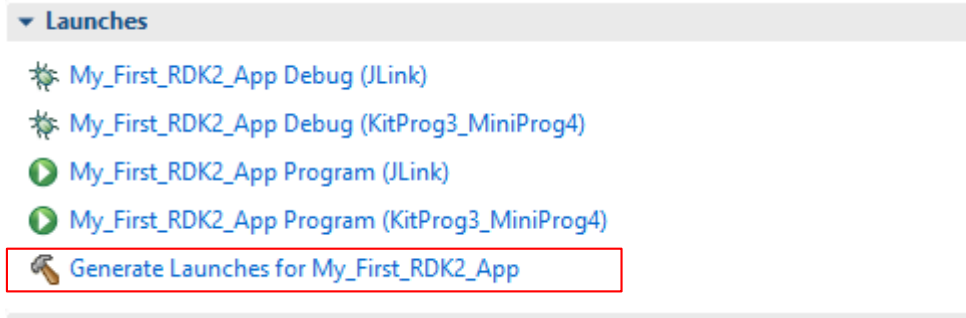
Creating new projects using “RutDevKit-PSoC62-BSP”

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



```
CONFIG=Costum  
DEFINES += COMPONENT_CAT1  
DEFINES += COMPONENT_CAT1A  
CFLAGS =-O0
```

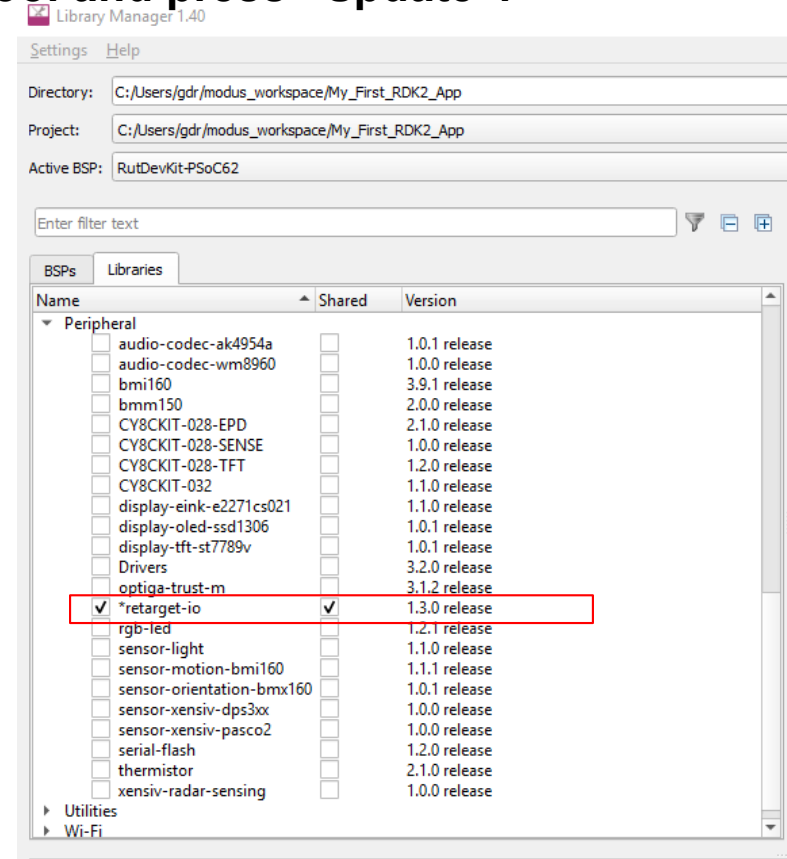
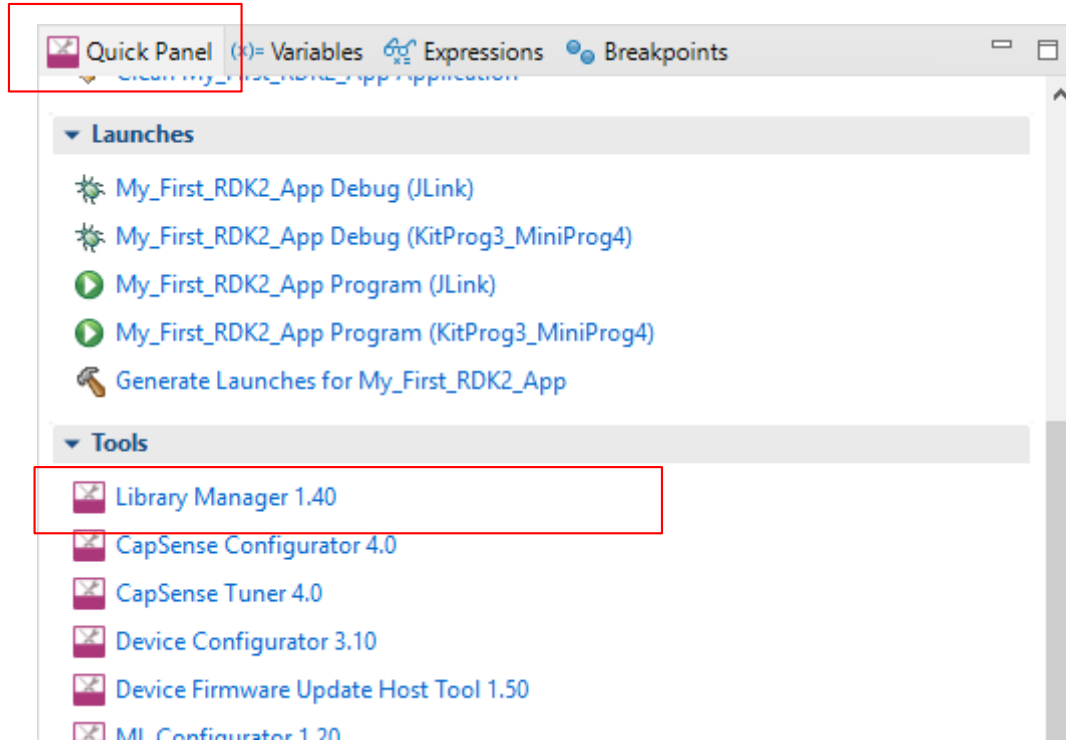
7.) Press “Generate Launches for (project name)” in Quick Panel



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

Creating new projects using “RutDevKit-PSoC62-BSP”

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();

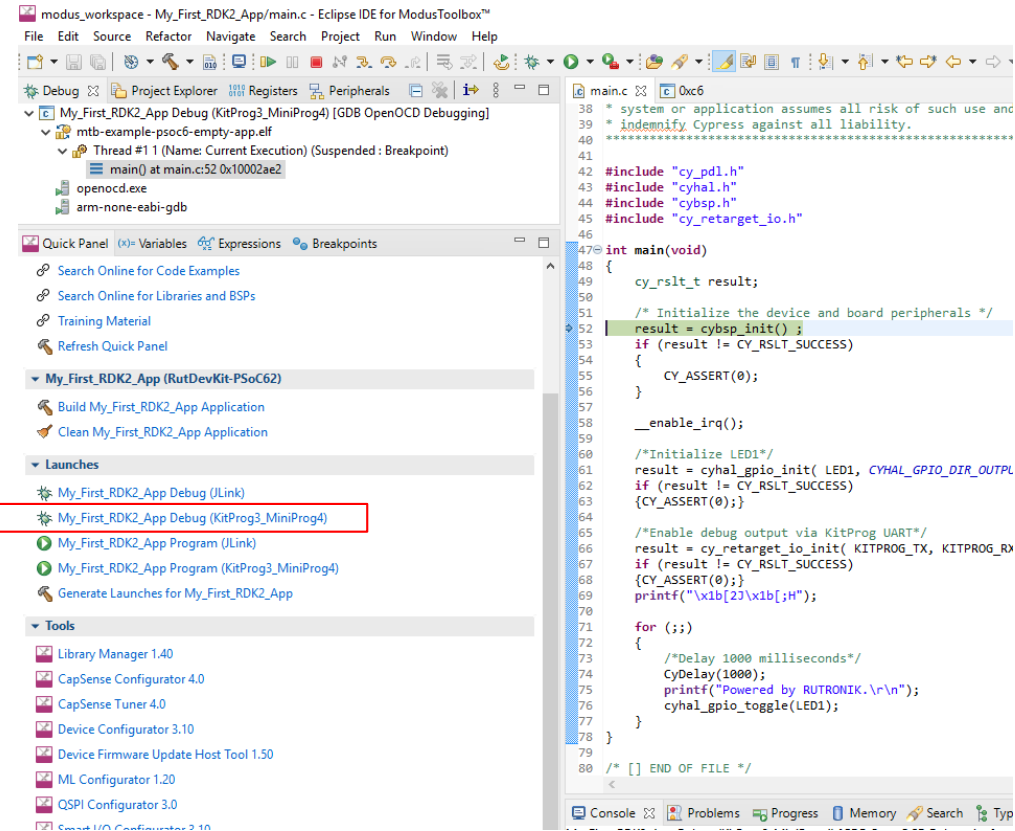
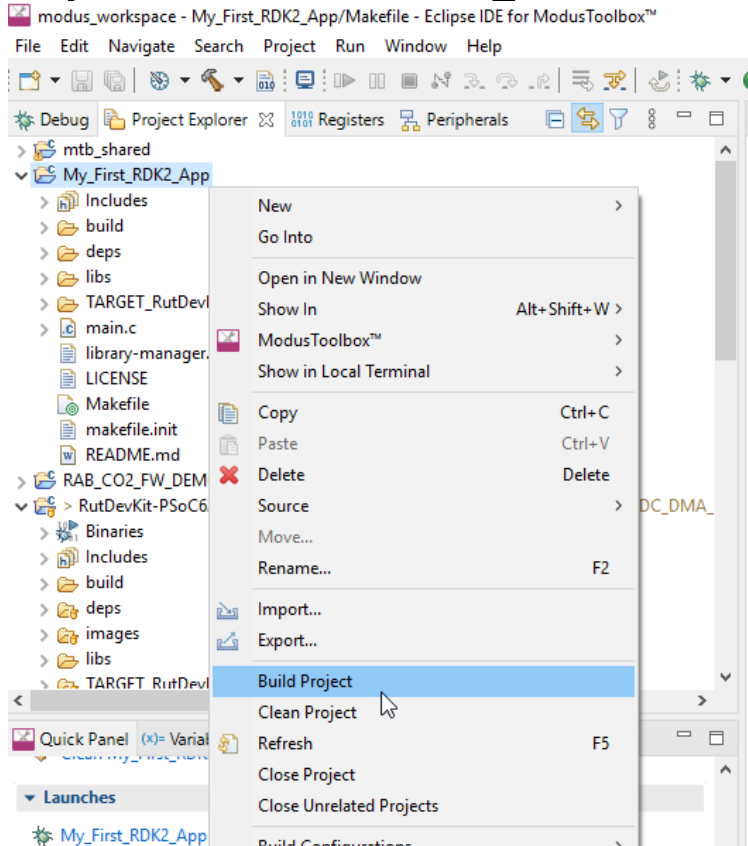
    /*Initialize LED1*/
    result = cyhal_gpio_init( LED1, CYHAL_GPIO_DIR_OUTPUT, CYHAL_GPIO_DRIVE_STRONG, CYBSP_LED_STATE_OFF);
    if (result != CY_RSLT_SUCCESS)
    {CY_ASSERT(0);}

    /*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);
    }
}
```

Creating new projects using “RutDevKit-PSoc62-BSP”

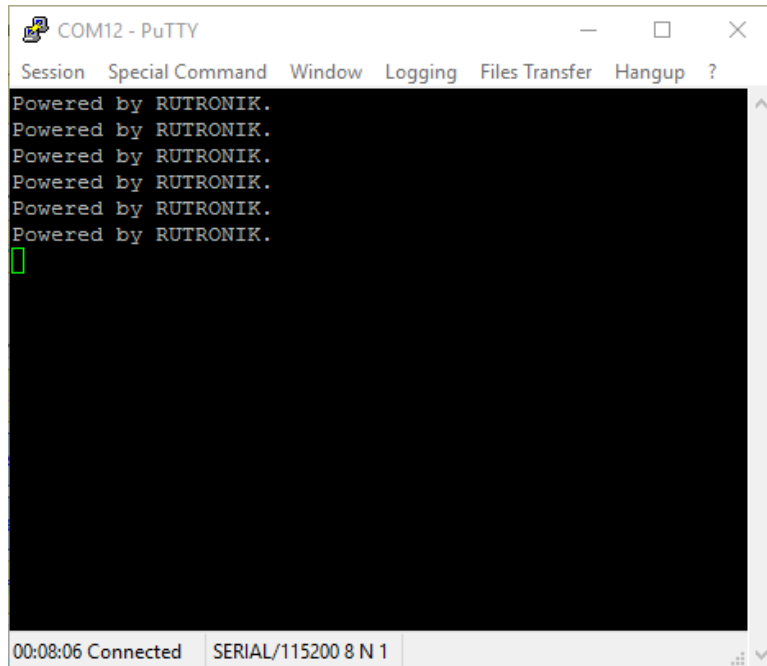
10.) Build and Debug the active project.



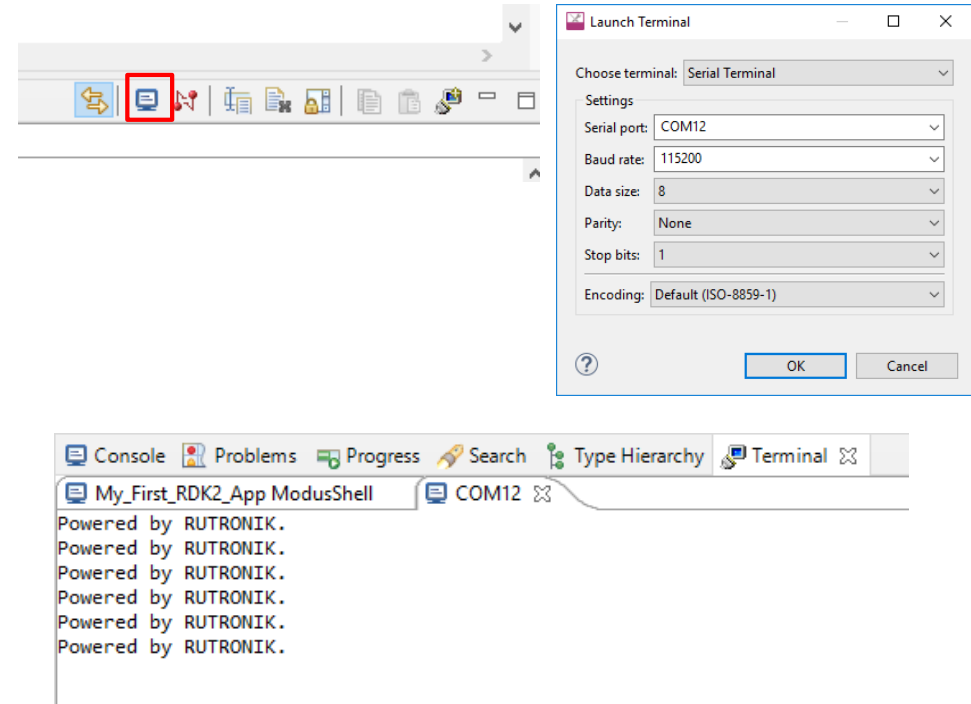
Creating new projects using “RutDevKit-PSoC62-BSP”

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

PuTTY Terminal



ModusToolbox Terminal





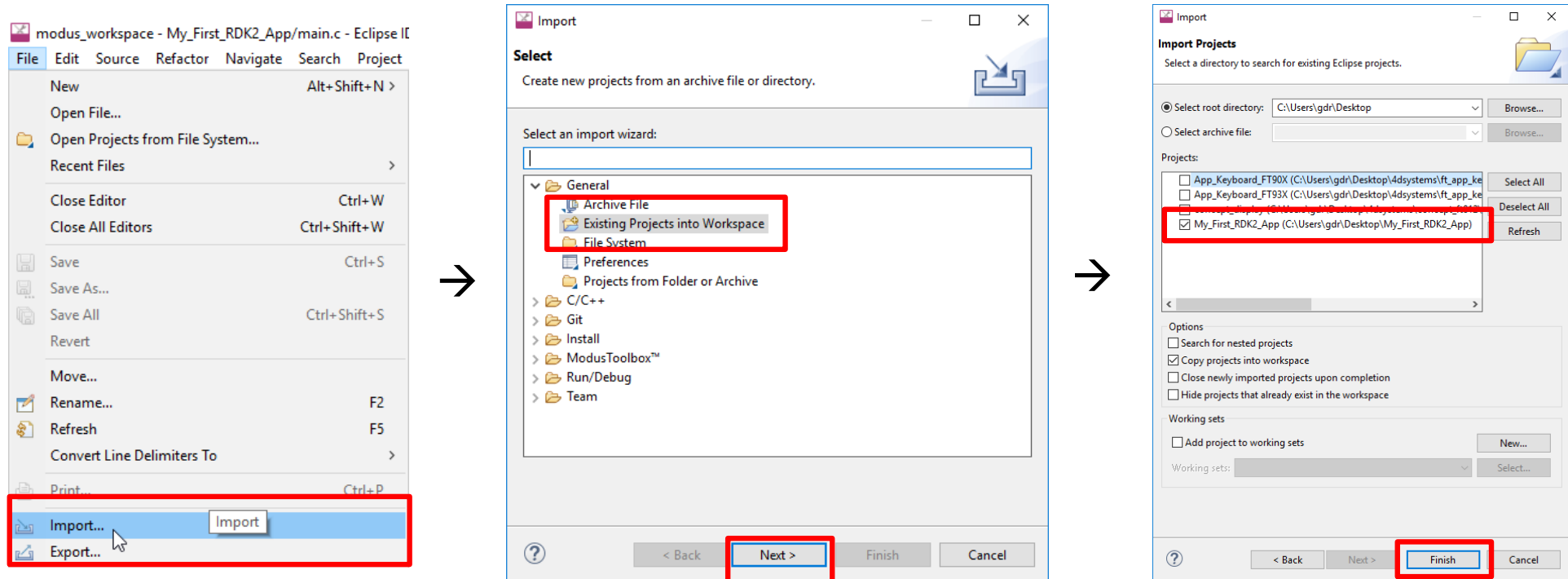
RUTRONIK
ELECTRONICS WORLDWIDE



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”.

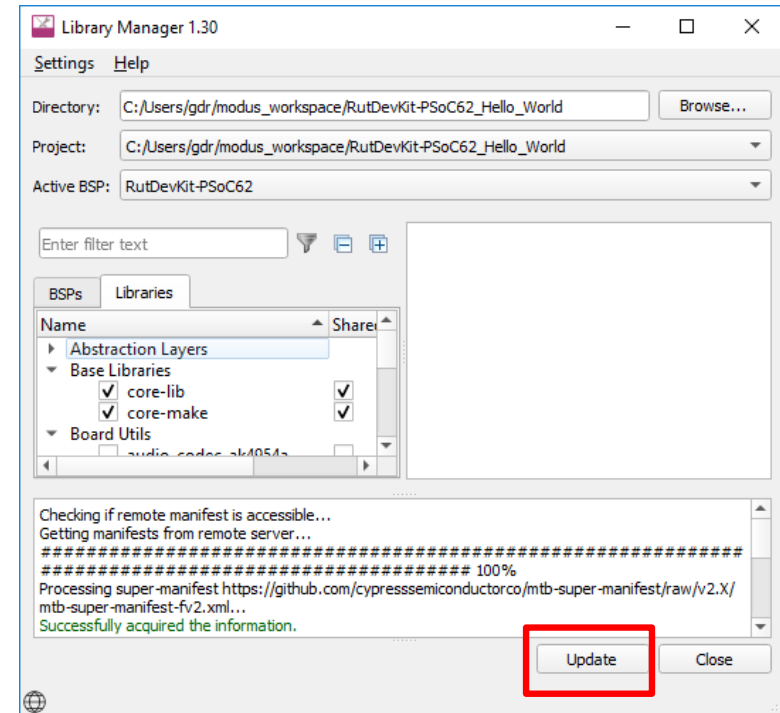
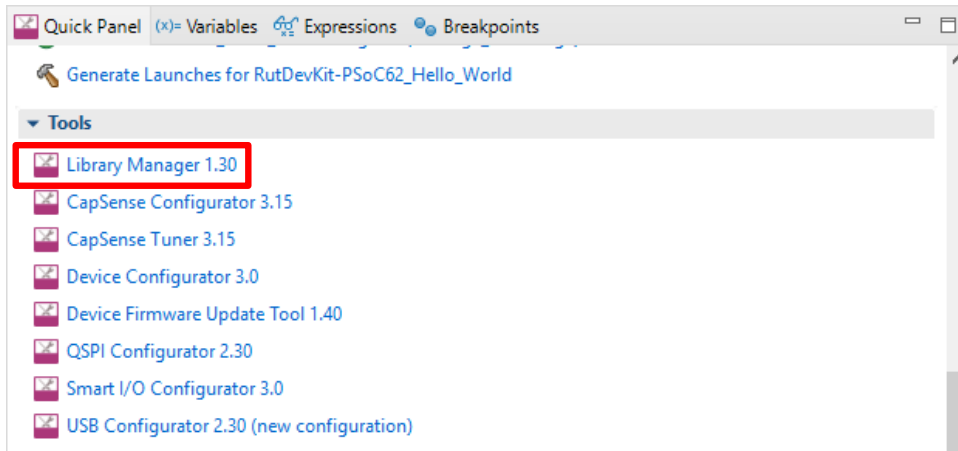


The image illustrates the process of importing an existing project into an Eclipse workspace through three sequential screenshots:

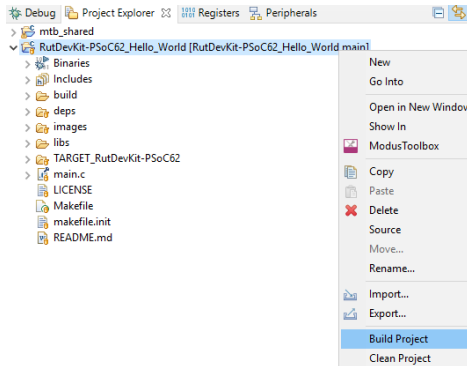
- First Screenshot:** The Eclipse IDE's File menu is open. The 'Import...' option is highlighted with a red box, and a mouse cursor is pointing at it.
- Second Screenshot:** The 'Import' wizard is displayed. Under the 'Select an import wizard' section, 'Existing Projects into Workspace' is selected and highlighted with a red box. The 'Next >' button at the bottom is also highlighted with a red box.
- Third Screenshot:** The 'Import Projects' dialog is shown. The 'Select root directory' is set to 'C:\Users\gdr\Desktop'. In the 'Projects' list, 'My_First_RDK2_App (C:\Users\gdr\Desktop\My_First_RDK2_App)' is selected and highlighted with a red box. In the 'Options' section, 'Copy projects into workspace' is checked. The 'Finish' button at the bottom is highlighted with a red box.

Importing Existing Projects into Workspace

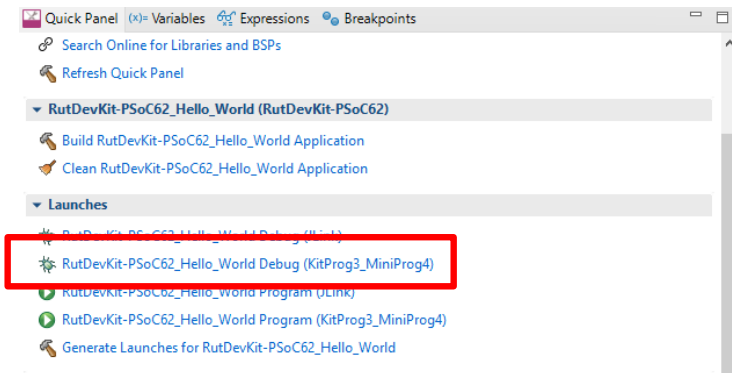
3.) Update the libraries using the “Library Manager”.



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

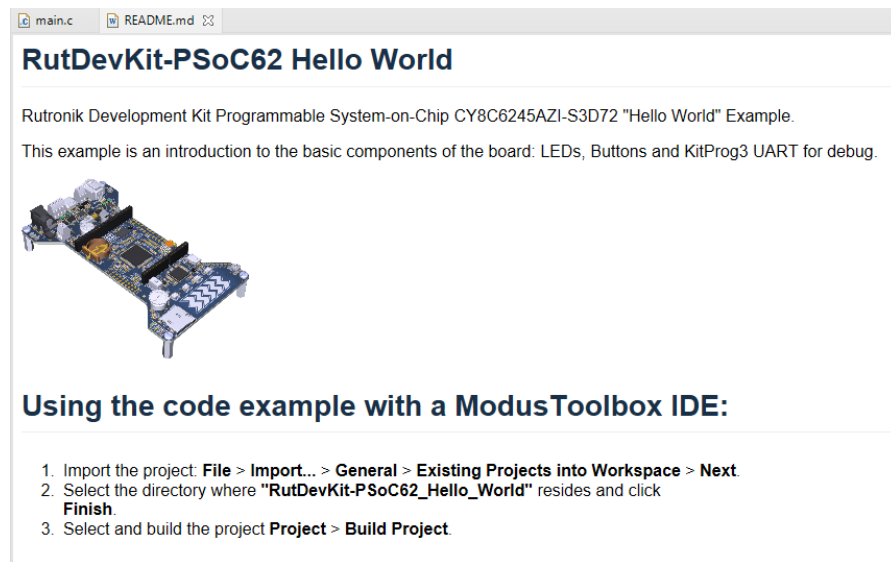
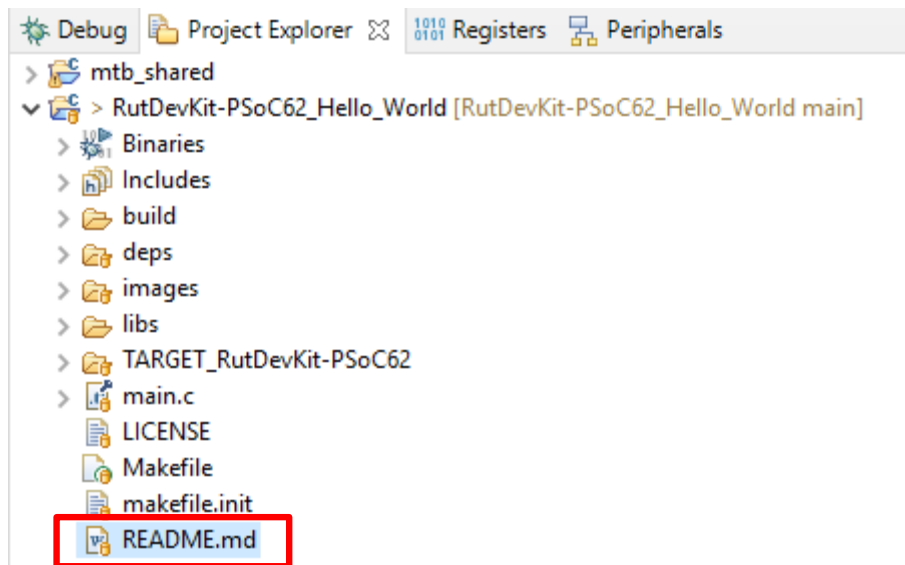


2.) Debug: Click on “KitProg3” debug option in “Quick Panel”.



“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.



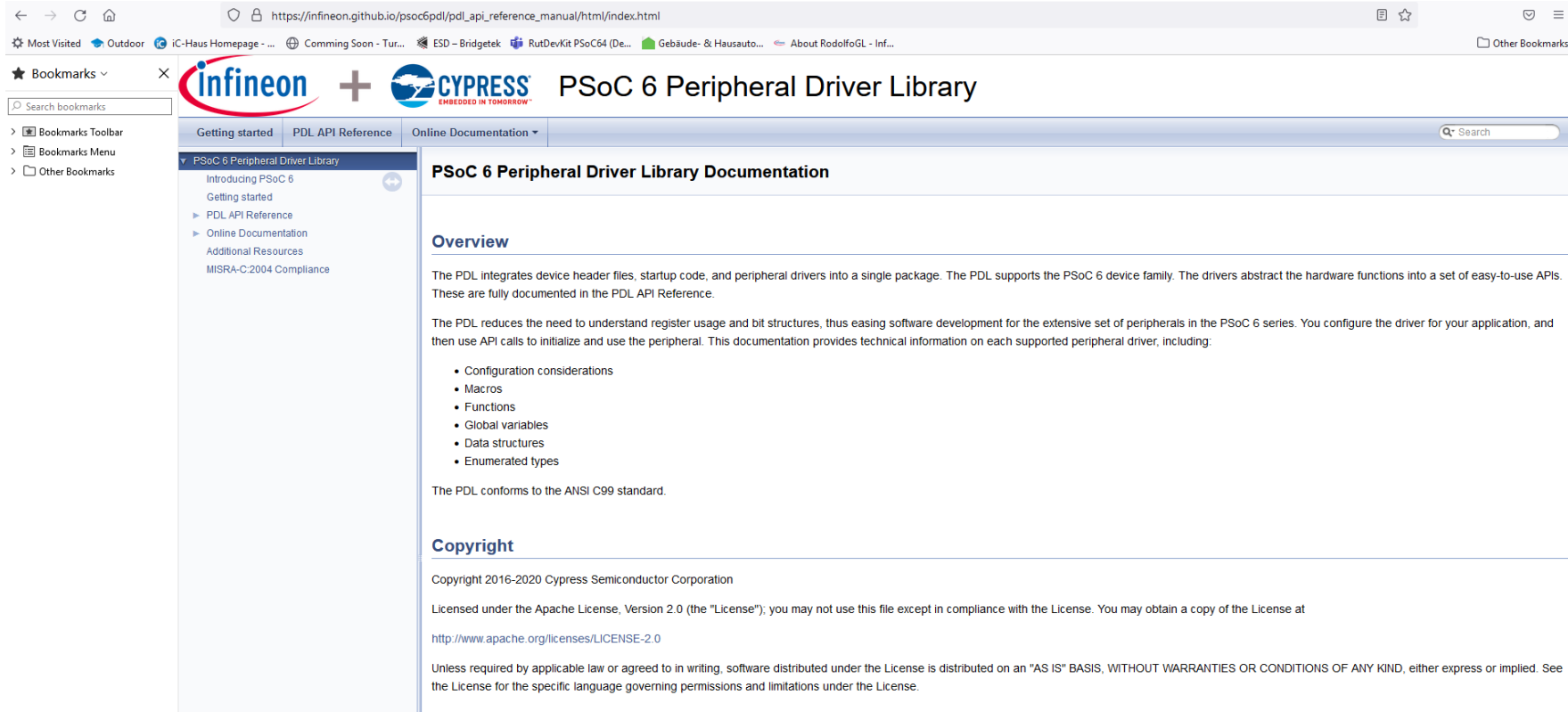


RUTRONIK
ELECTRONICS WORLDWIDE

The background image is a close-up, low-angle shot of a printed circuit board (PCB) under a blue light. A large, square integrated circuit (chip) is the central focus, with its pins visible. Surrounding the chip are various other components, including smaller chips and capacitors, some of which are glowing with a bright green light. The overall scene is dark and technical, emphasizing electronic engineering.

HAL, PDL APIs and ModusToolbox™ Tools

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



The screenshot shows a web browser displaying the PSoc 6 Peripheral Driver Library Documentation. The browser's address bar shows the URL: https://infineon.github.io/psoc6pdl/pdl_api_reference_manual/html/index.html. The page features the Infineon and Cypress logos at the top, followed by the title "PSoc 6 Peripheral Driver Library". Below the title, there are tabs for "Getting started", "PDL API Reference", and "Online Documentation". The "PDL API Reference" tab is selected. On the left side, there is a sidebar with a search bar and a list of navigation links: "PSoc 6 Peripheral Driver Library", "Introducing PSoc 6", "Getting started", "PDL API Reference", "Online Documentation", "Additional Resources", and "MISRA-C:2004 Compliance". The main content area is titled "PSoc 6 Peripheral Driver Library Documentation" and contains an "Overview" section. The overview text states: "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." It also mentions that the PDL reduces the need to understand register usage and bit structures, easing software development for the extensive set of peripherals in the PSoc 6 series. A bulleted list of topics covered in the documentation is provided: Configuration considerations, Macros, Functions, Global variables, Data structures, and Enumerated types. The page also notes that the PDL conforms to the ANSI C99 standard. At the bottom, there is a "Copyright" section stating that the documentation is copyrighted by Cypress Semiconductor Corporation from 2016 to 2020 and is licensed under the Apache License, Version 2.0. A link to the Apache License is provided: <http://www.apache.org/licenses/LICENSE-2.0>. The final paragraph states that the software is distributed under the License on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, and refers to the License for specific language governing permissions and limitations.

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



The screenshot shows a web browser displaying the Infineon Hardware Abstraction Layer (HAL) API Reference page. The browser's address bar shows the URL <https://infineon.github.io/psoc6hal/html/index.html>. The page features the Infineon and Cypress logos at the top, followed by the title "Hardware Abstraction Layer (HAL)". The left sidebar contains a navigation menu with "Home" and "HAL API Reference" (selected). The main content area is titled "Hardware Abstraction Layer" and includes an "Overview" section. The overview text states: "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." It also mentions that to use code from the HAL, the specific driver header file can be included or the top level `cyhal.h` header can be included to allow access to any driver. Below the overview is the "API Structure" section, which states: "The API functions for each HAL driver can be divided into the following categories:" and lists three categories:

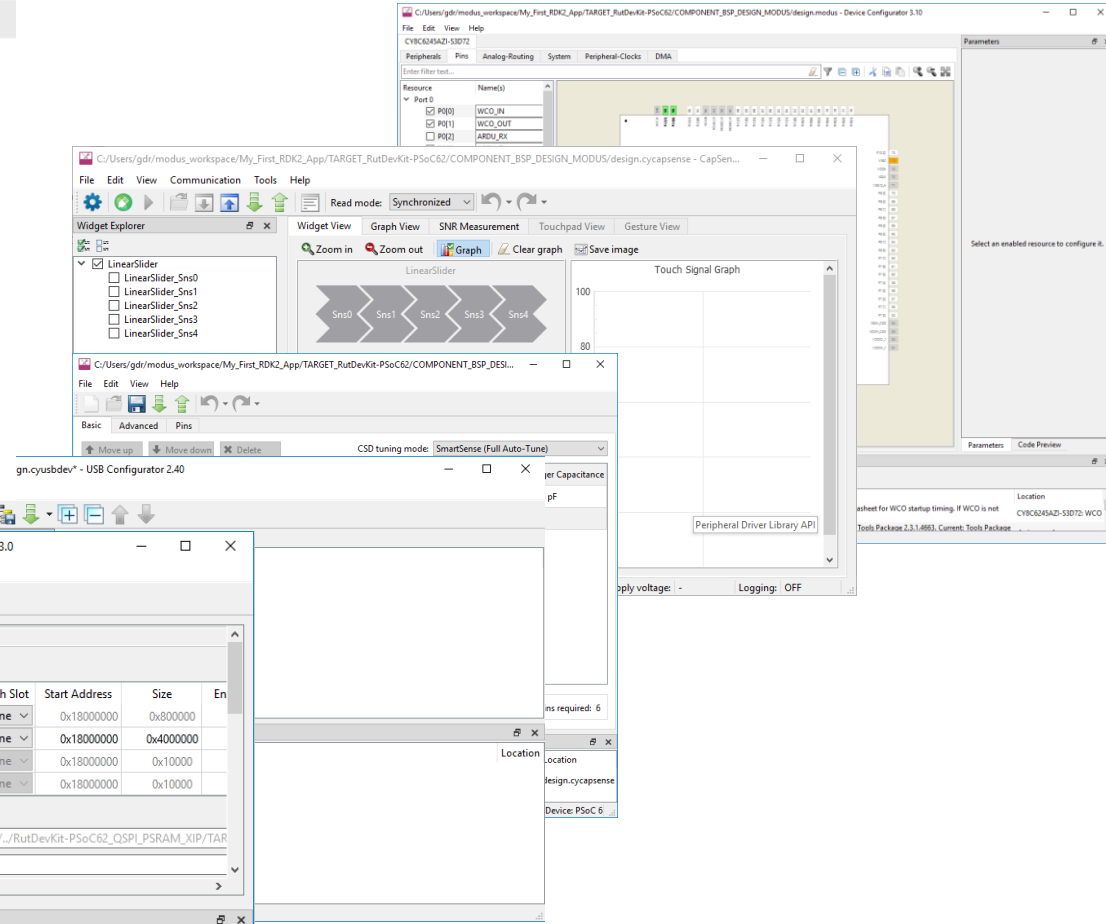
- A `_init` function allocates a block (along with any dependent resources), configures it, and enables it.
- A `_free` function disables a block, releases any dependent resources, and marks it as unused.
- Additional functions provide access to block-specific functionality and, in some cases, manipulate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

It then explains that `_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. It also mentions that 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. A note states: "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." The "Resource Identification" section states: "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

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)



The screenshot displays the ModusToolbox environment with several tool windows open:

- Device Configurator 3.10:** Shows the 'Peripherals' tab for the CY8C640A2I-S3072 device. The 'Port 0' section lists resources: P0[0], P0[1], P0[2], P0[3], P0[4], P0[5], P0[6], P0[7], P0[8], P0[9], P0[10], P0[11], P0[12], P0[13], P0[14], P0[15], P0[16], P0[17], P0[18], P0[19], P0[20], P0[21], P0[22], P0[23], P0[24], P0[25], P0[26], P0[27], P0[28], P0[29], P0[30], P0[31], P0[32], P0[33], P0[34], P0[35], P0[36], P0[37], P0[38], P0[39], P0[40], P0[41], P0[42], P0[43], P0[44], P0[45], P0[46], P0[47], P0[48], P0[49], P0[50], P0[51], P0[52], P0[53], P0[54], P0[55], P0[56], P0[57], P0[58], P0[59], P0[60], P0[61], P0[62], P0[63], P0[64], P0[65], P0[66], P0[67], P0[68], P0[69], P0[70], P0[71], P0[72], P0[73], P0[74], P0[75], P0[76], P0[77], P0[78], P0[79], P0[80], P0[81], P0[82], P0[83], P0[84], P0[85], P0[86], P0[87], P0[88], P0[89], P0[90], P0[91], P0[92], P0[93], P0[94], P0[95], P0[96], P0[97], P0[98], P0[99].
- CapSense Configurator 4.0:** Shows the 'Widget Explorer' with a 'LinearSlider' widget selected. The 'Widget View' shows a 'LinearSlider' widget with 'Sns0' through 'Sns4' sensors. The 'Graph View' shows a 'Touch Signal Graph' with a 'LinearSlider' widget.
- QSPI Configurator 3.0:** Shows the 'Target Device Family' as CAT1A. The 'Program Chunk Size' is 4 KB and the 'Erase Chunk Size' is 256 KB. The 'Data Select' table is as follows:

Slave Slot	Memory Part Number	Configuration	Data Select	Memory Mapped	Pair with Slot	Start Address	Size	En
0	AP56404L-3SQR-ZR	Uniform	Quad SPI-Data[0:3]	<input type="checkbox"/>	None	0x18000000	0x800000	
1	S25HL512T	Hybrid at Top	Quad SPI-Data[0:3]	<input checked="" type="checkbox"/>	None	0x18000000	0x4000000	
2	Not Used		SPI-MOSI-MISO Data[0:1]	<input type="checkbox"/>	None	0x18000000	0x10000	
3	Not Used		SPI-MOSI-MISO Data[0:1]	<input type="checkbox"/>	None	0x18000000	0x10000	

The 'Location' field is set to: `z:/gdr/modus_workspace/RutDevKit-PSoC62_SensorFusion/TARGET_RutDevKit-PSoC62/COMPONENT_BSP_DESIGN_MODUS/design.cyqspi`. The 'Memory Part Number' is 'AP56404L-3SQR-ZR' and the 'Max Erase Time' is '1'.



Gintaras Drukteinis

Technical Support

Phone : +370 372 45568

eMail : gdr@rutronik.com