TRAVEO™ T2G Body High Lite Kit
Getting Started Guide

Table of contents

Objective

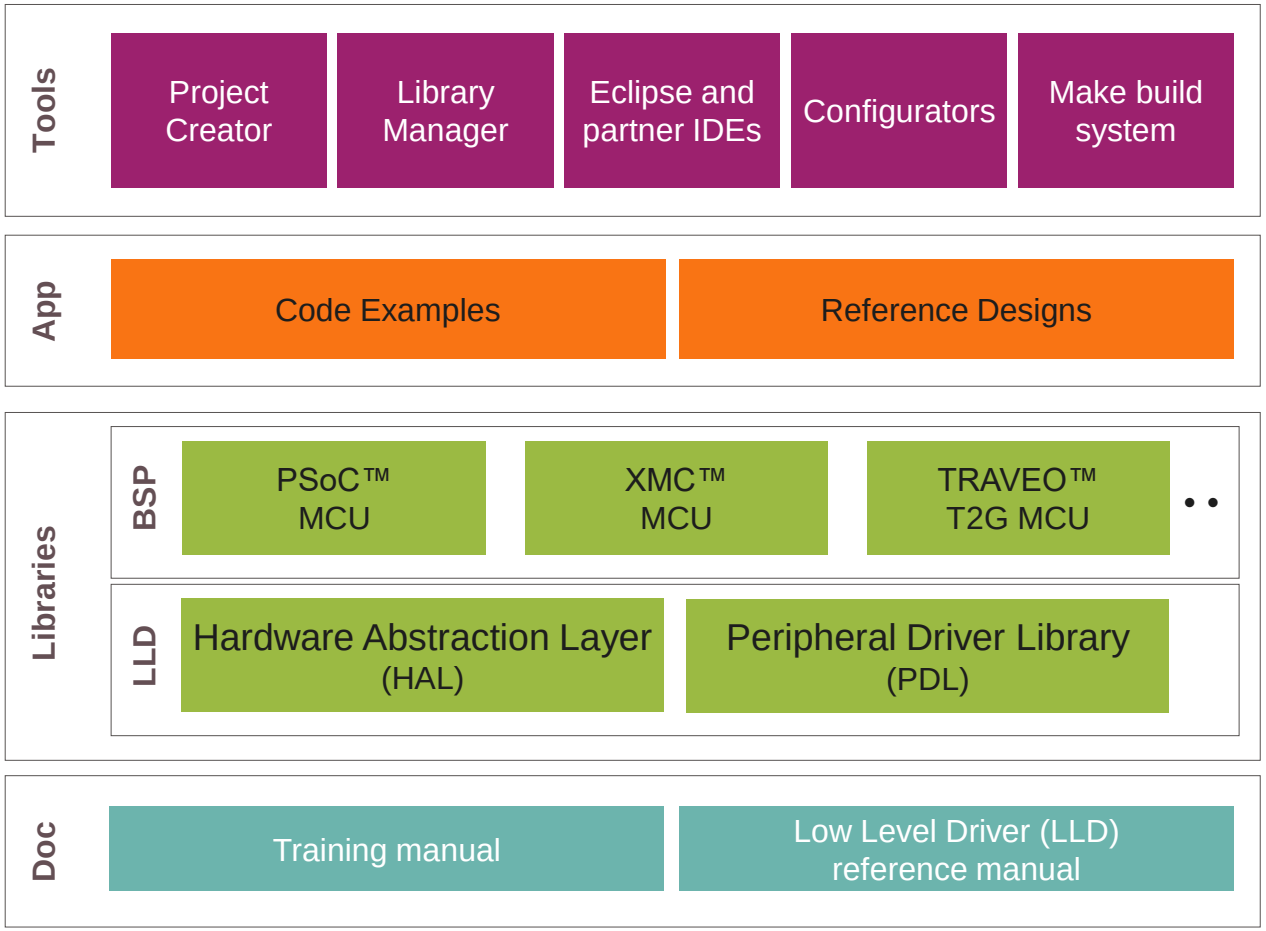
Learning objectives

- How to use Modus Toolbox with KIT_T2G-B-H_LITE
- How to develop a TRAVEO™ application

Overview – Modus Toolbox™

Modus Toolbox™

– Modus Toolbox™ software structure



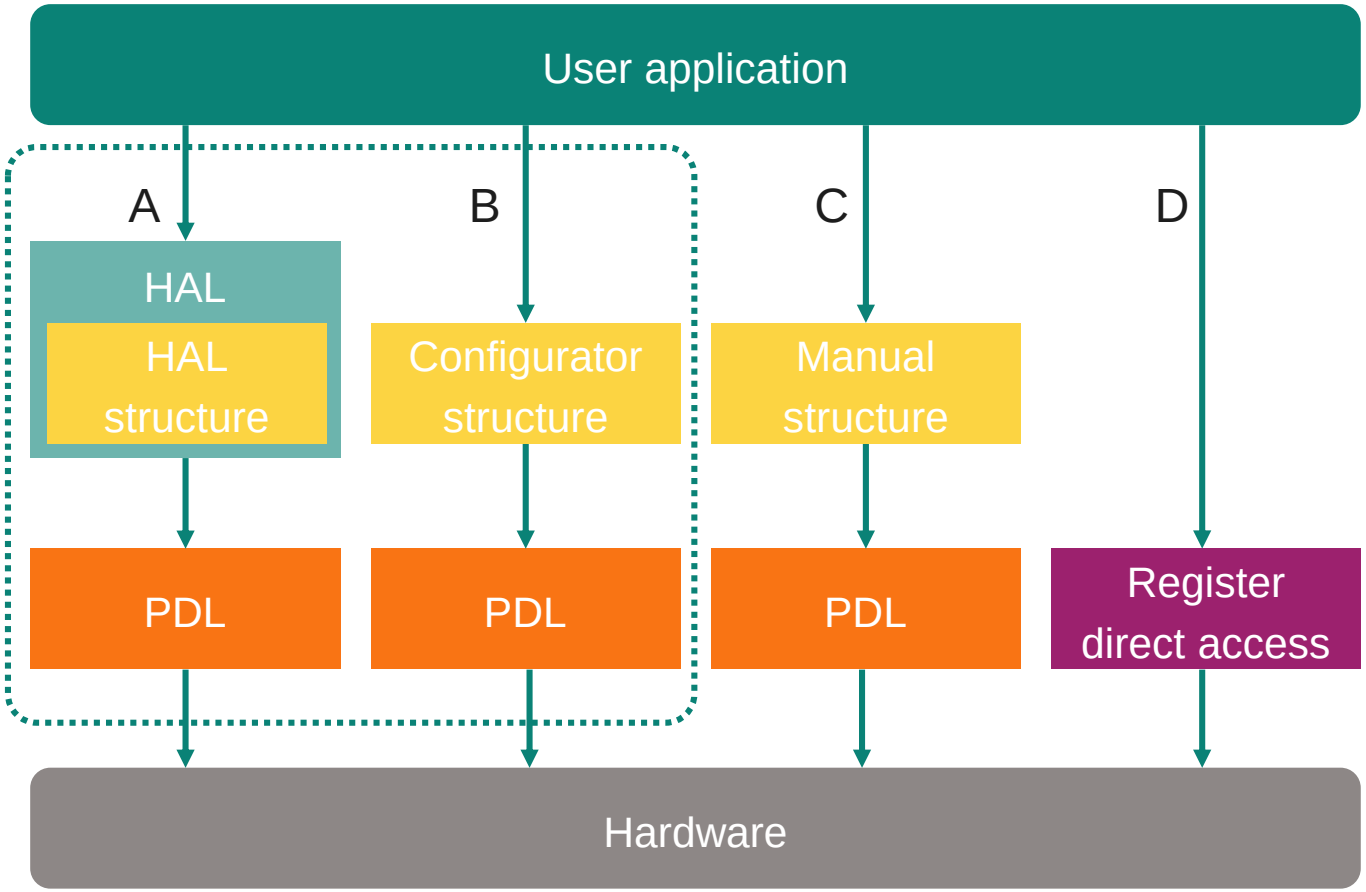
See Appendix A for Additional Information

- Tools:
  - Creation of new embedded applications
  - Managing software components
  - Configuring peripherals
  - Development tools
- Application:
  - Used for starting point of application development
- Libraries:
  - BSP: supports the target device and board
  - HAL: LLD of simple functions and high-level interface
  - PDL: LLD to support all hardware features
- Documentation
  - These are hosted in GitHub

BSP: Board Support Package
HAL: Hardware Abstraction Layer
PDL: Peripheral Driver Library

Modus Toolbox™ (MTB) application interface

– An application has four ways to access the hardware



A. Application code uses the HAL, which interacts with the PDL through structures created by the HAL

B. Application code uses PDL through structures created by a configurator tools

C. Application code uses PDL through structures created manually

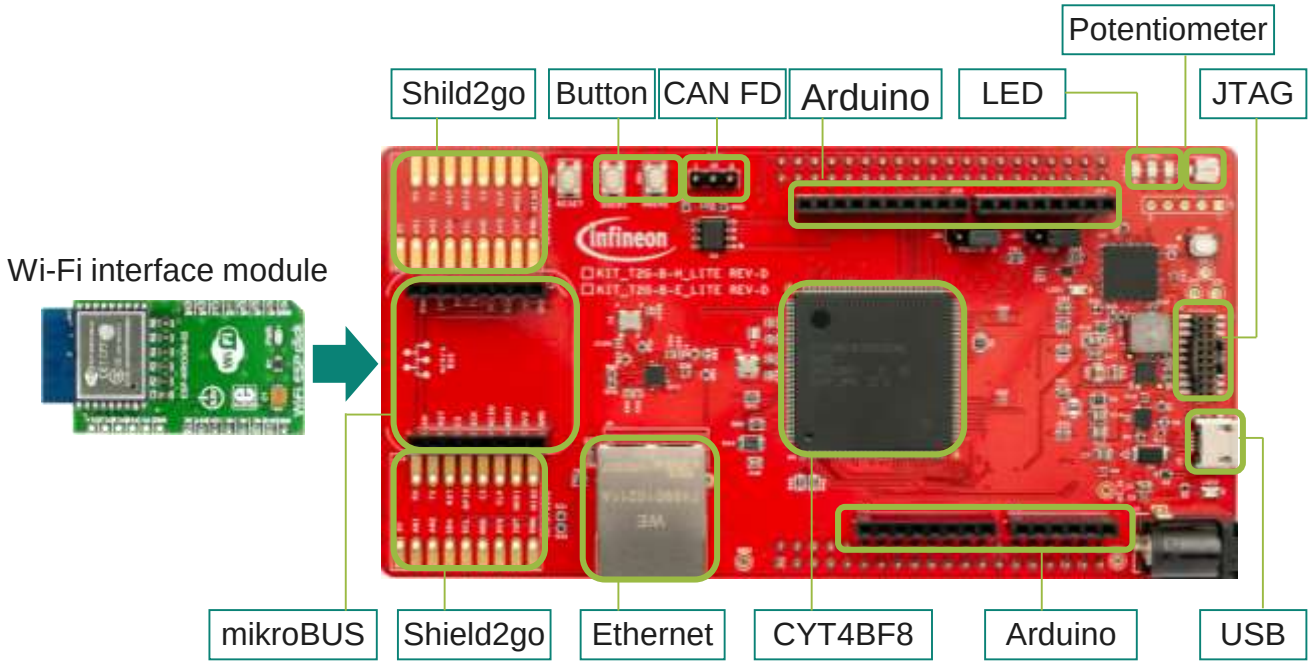
D. Application code uses direct register read and writes

Software & Hardware Setup

TRAVEO™ T2G Evaluation kit (KIT_T2G-B-H_LITE)

– KIT_T2G-B-H_LITE

Features	KIT_T2G-B-H_LITE
MCU	CYT4BF8 (176pin-TEQFP)
Size	130mm x 66mm
MTB connection	Kitprog3 USB connector Kitprog3 programing/Debug
JTAG interface	MIPI-20 connector
User interface	LED x3 Button x2 Potentiometer x1
Communication interface	CAN FD Ethernet
Expansion interface	Arduino x1 mikroBUS x1 Shield2go x2



CYT4BF series

TRAVEO™ T2G body MCU family

Applications

Body control module (BCM), gateway, and infotainment

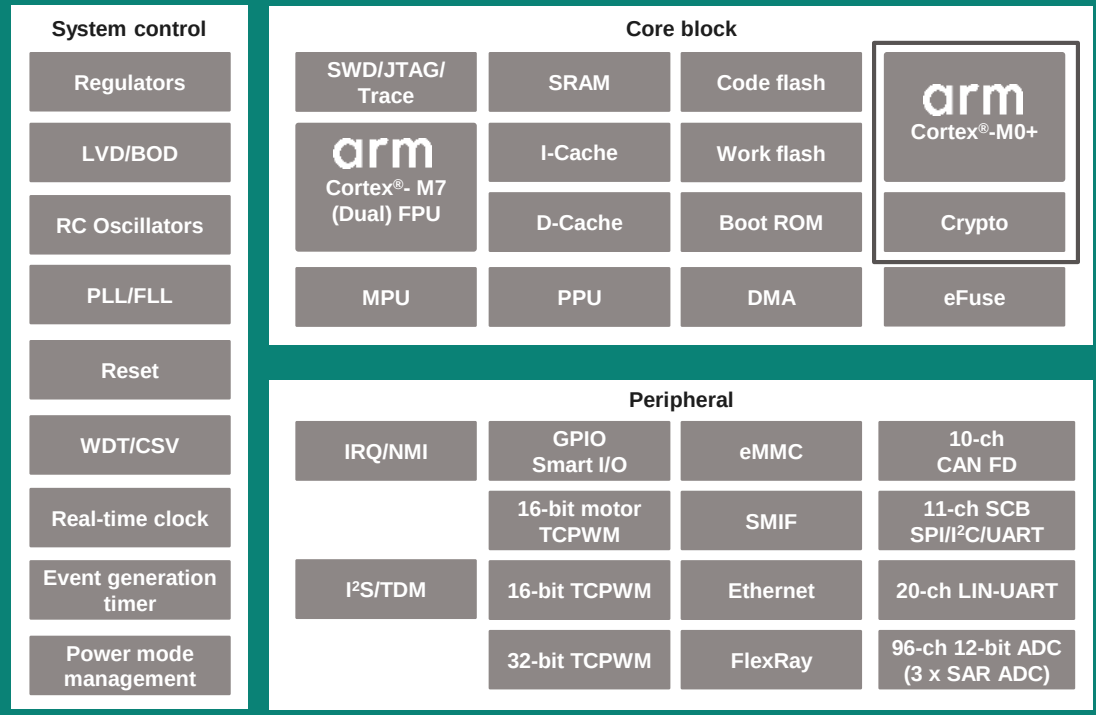
Features

- **32-bit MCU core systems**
  - 350-MHz Arm® Cortex®-M7 Dual, I/D-Cache
  - 8MB flash, 256KB work flash, 1024KB SRAM, and Cortex®-M0+ for crypto
- **2.7-V to 5.5-V supply voltages**
- **Interfaces**
  - Up to 10-ch CAN FD, up to 11-ch SCB, and 20-ch LIN-UART
  - eMMC, SMIF (QSPI/HS-SPI), up to 2-ch 10/100/1000-Mbit ethernet and FlexRay
  - I²S/TDM with TX 3ch and RX 3ch
- **AD converter**
  - Up to 96-ch, 12-bit with 3x successive approximation ADC (SAR ADC) units
- **Timers**
  - Up to 15-ch motor control, 87-ch 16-bit timer/counter/pulse-width modulation (TCPWM), and 16-ch 32-bit TCPWM
  - Event generation timer
- **High-speed I/O at 3.3-V in BGA**
  - Second operation voltage required for 3.3-V I/O
- **Packages**
  - 176-pin TEQFP, 272-ball BGA, 320-ball BGA

Collateral

Datasheet: CYT4BF series

CYT4BF series



Availability

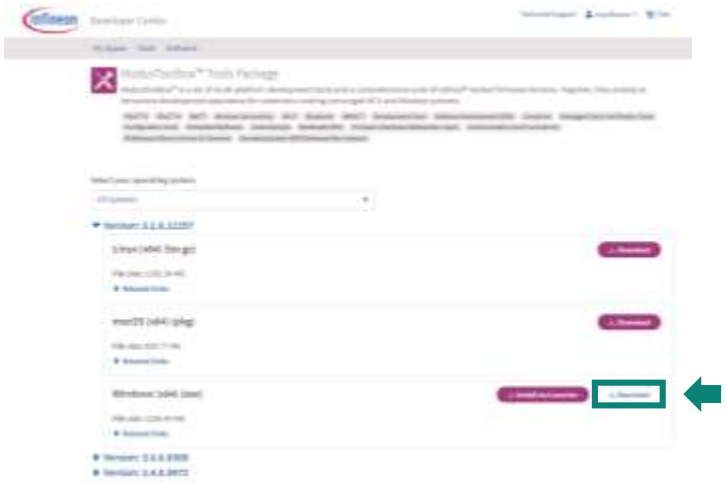
Sampling: Now

Production: Now

Software and Firmware Required for Training

Note: Admin rights are required to install the programs

- **Step 1**: Modus toolbox™ installation - mandatory
  - Install Modus toolbox 3.1
  - URL: https://softwaretools.infineon.com/tools/com.ifx.tb.tool.modustoolbox

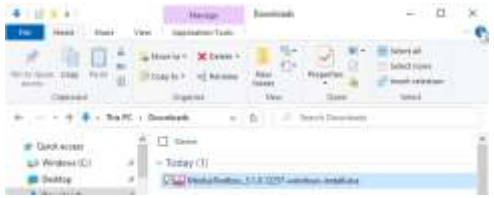


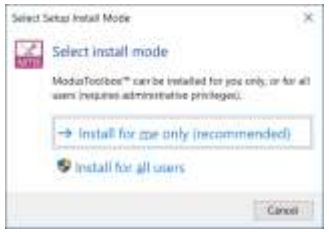
Click here to download Modus Toolbox 3.1

Note: A newer version of Modus Toolbox may be available

Software and Firmware Required for Training

– **Step 2**: Perform the “ModusToolbox_3.1.0.12257-windows-install.exe”



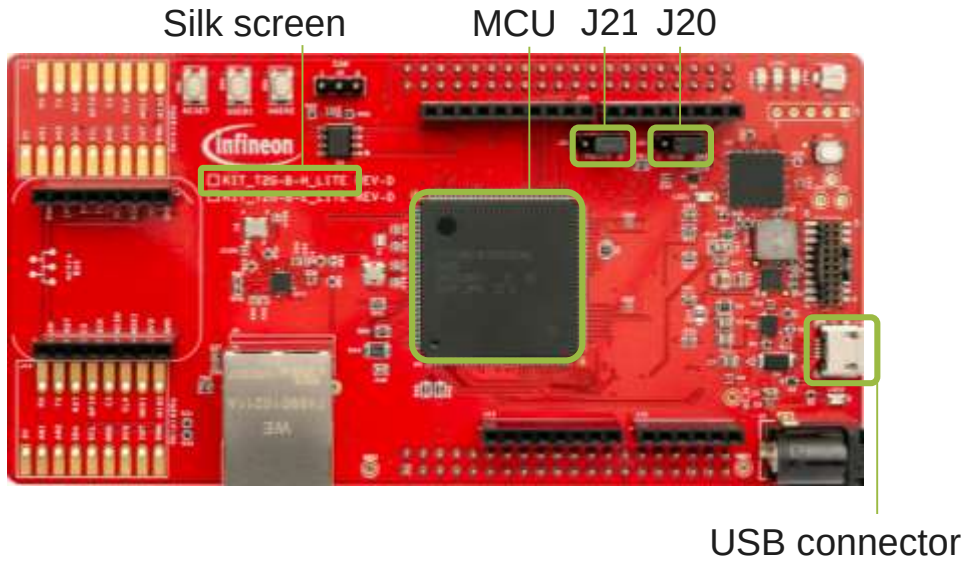


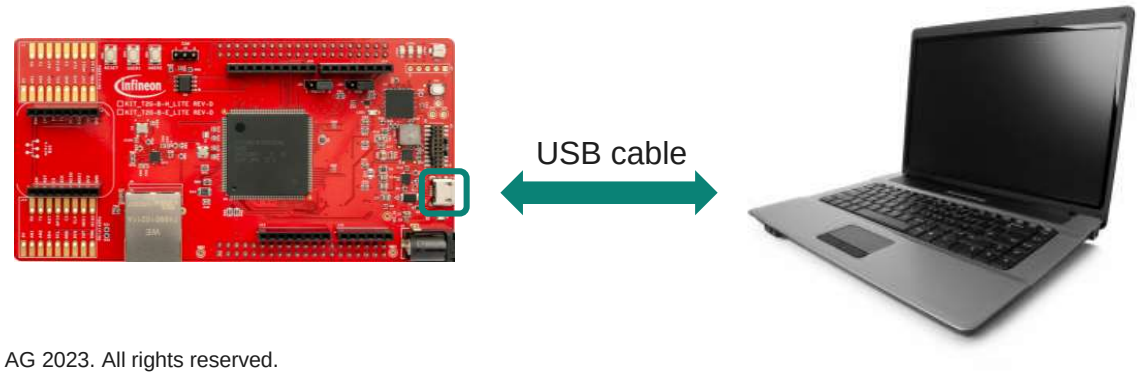
It is recommended the default location.
If you need to install to other location, additional work will be required.

- Select default installation
- Start install when click “**Next**” button

Jumper Connections and Prerequisites for the session

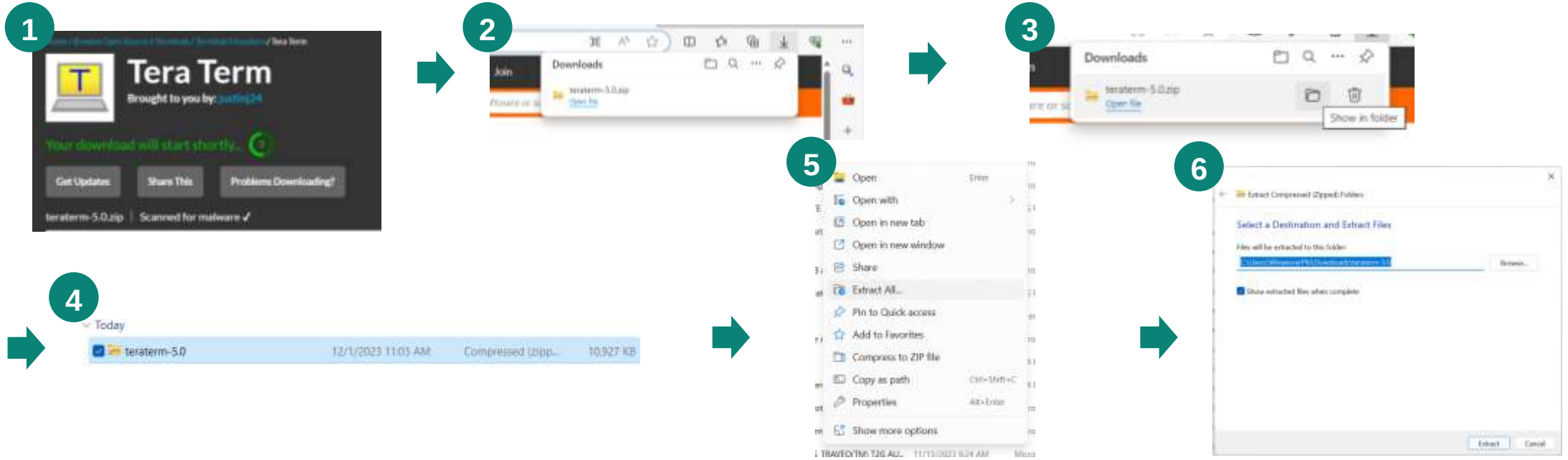
- **Step 3**: Hardware Requirements
  - Make sure your received board has:
    - Silk screen: KIT_T2G-B-H_LITE
    - MCU: CYT4BF8xx
  - Make sure you have a **USB Type-A to Micro-B** cable
  - Make sure default jumpers are inserted on the evaluation Kit
    - J20: VDD power jumper
    - J21: VDDIO power jumper
  - **Now:** Connect your PC and board via USB connector





Software and Firmware Required for Training

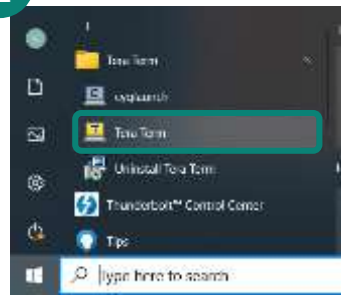
- **Step 4**: Terminal emulator installation
- **Any Terminal emulator can be used** – in case you don't have one please follow the steps below
  1. Tera Term 5.0 (free): **Download here –** the download will start immediately after you clicked on the link
  2. Go to Downloads and follow the steps below to launch the Tera Term
     1. Download will start automatically after clicking on the link
     2. Open to the **Downloads** folder
     3. Click on **"Show in folder"**
     4. Right Click on **teraterm-5.0**
     5. Click on **Extract All...**
     6. Click on **Extract**



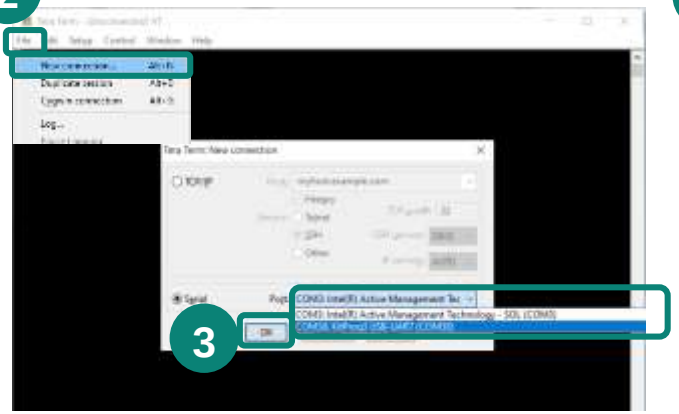
Software and Firmware Required for Training

- **Step 5**: Confirm your **Tera Term** setup **(Used to communicate with the board)**
  - Terminal emulator setting
  1. Open **Tera Term** Software on PC
  2. Select **New connection** in File tab
  3. Select **Serial option** and select the **COMxx: KitProg3 USB-UART (COMxx)** in the list Box, then click the "**OK**" button
  4. Select **Setup** (Tab bar) and then **Serial port** ... and make sure below setup, and click the "**New setting**" button
  5. Select **Setup** (Tab bar) and then **Terminal** ... and make sure below setup , and click the "**OK**" button
  6. *Optional* – you can save your settings under **Setup – Save Setup..**

1

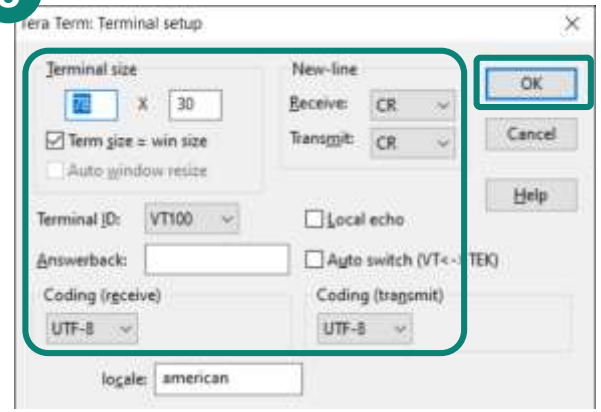


2



4

5



Note: Please do not close Tera Term window

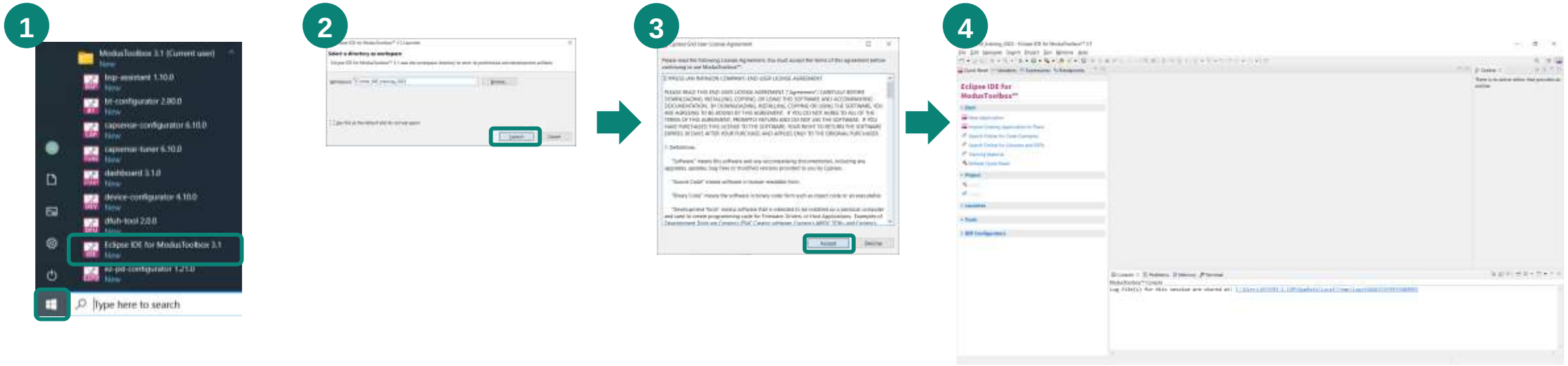
Create „Hello World“ Application (Single Core)

Hello World Application

First Use Case Description:

In this example you will test your Software and Hardware Setup

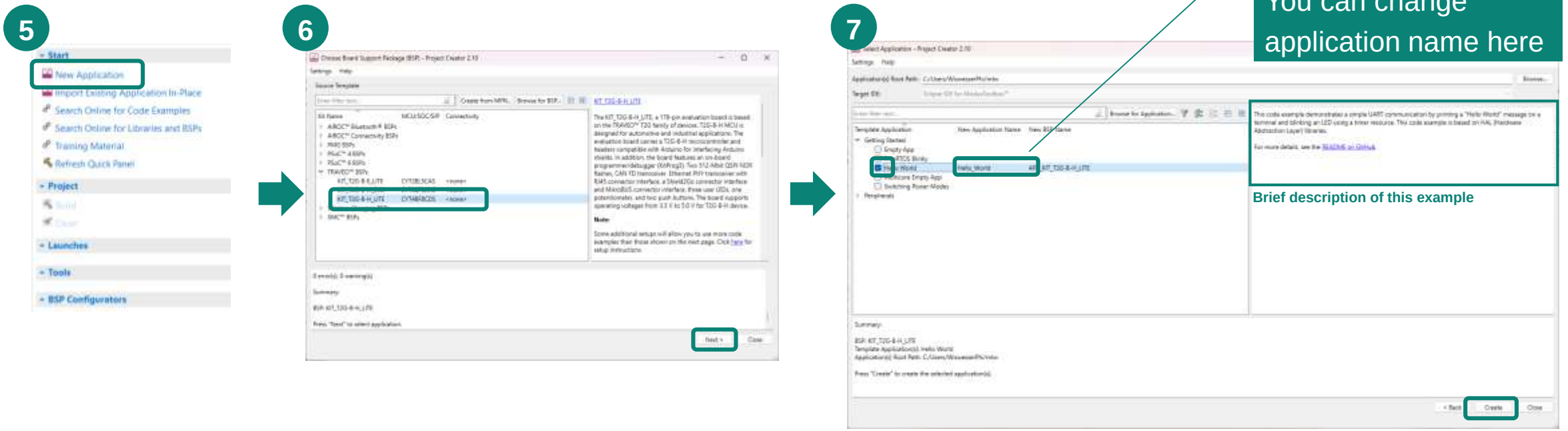
- **Step 1**: Create “**Hello World”** application
- Launch Modus Toolbox™
  1. From the start menu, run the “**Eclipse IDE for ModusToolbox 3.1**” application
  2. Set your workspace location, then click the “**Launch”** button.
  3. License Agreement dialog is displayed at the first launch. Click the “**Accept**“ button.
  4. After that, open the Eclipse IDE window.



Hello World Application

– Create “**Hello World**” application

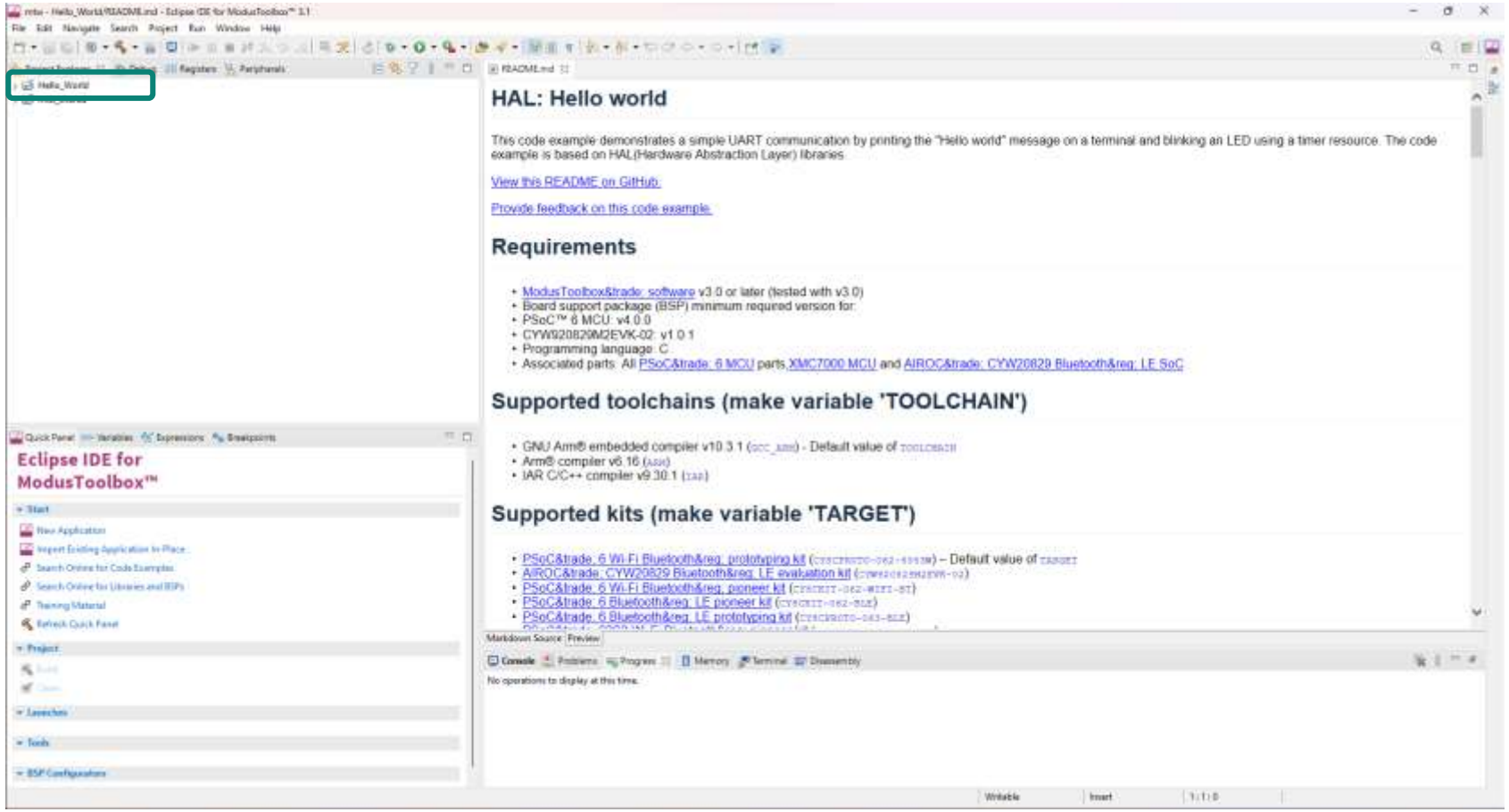
5. Click New Application under Start. After that, open the Project creator
6. Select “**KIT_T2G-B-H_LITE**” in the Choose Bard Support Package (BSP) window, and click “**Next**” button
7. Select “Hello world” in Getting Started, then click the “**Create**” button



Hello World Application

– Create “**Hello World**” application

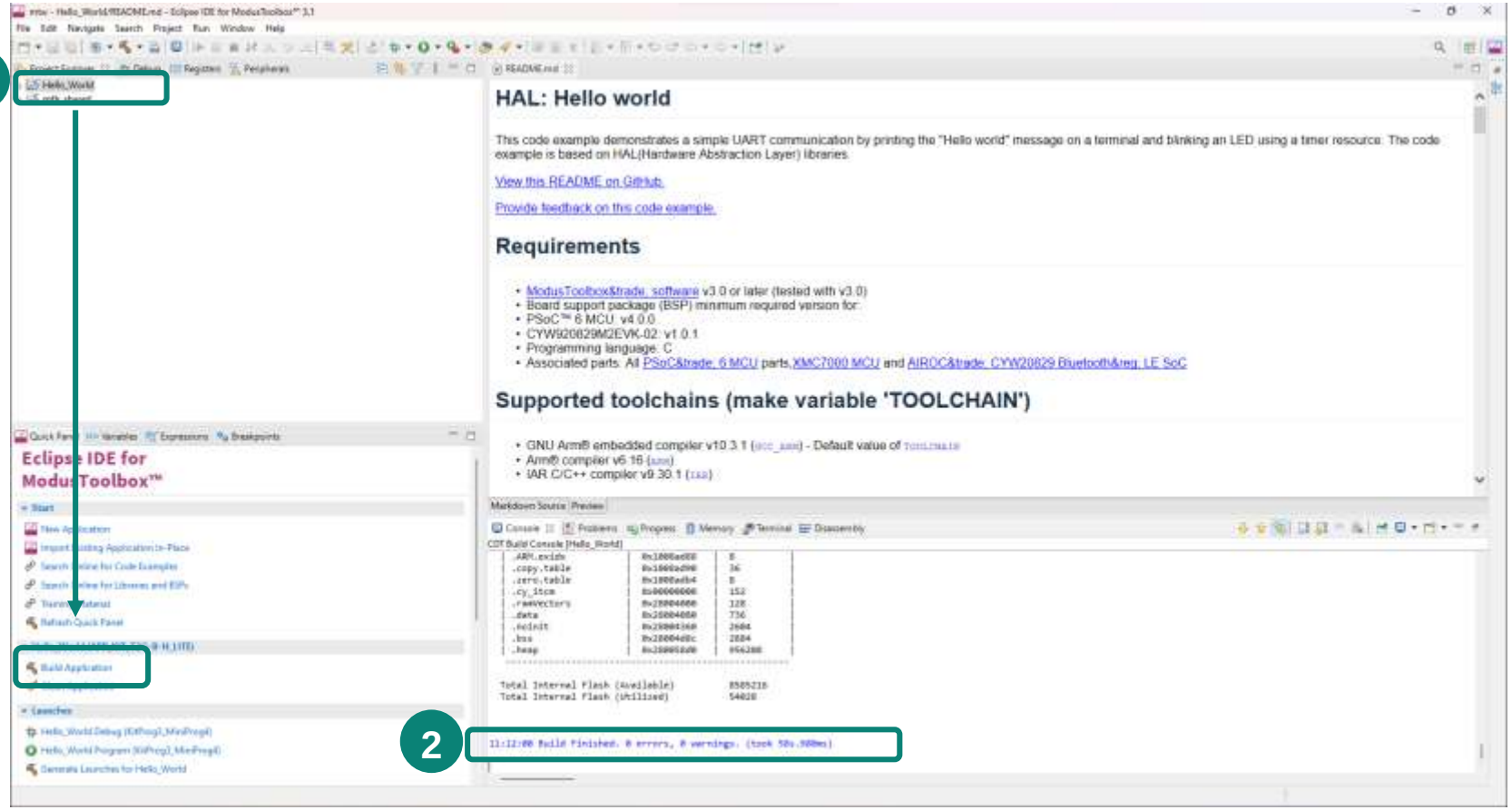
8. When application creation is complete, return to the Eclipse IDE window and the applications will be displayed in the Project Explorer



Hello World Application

– **Step 2**: Build “**Hello World**”

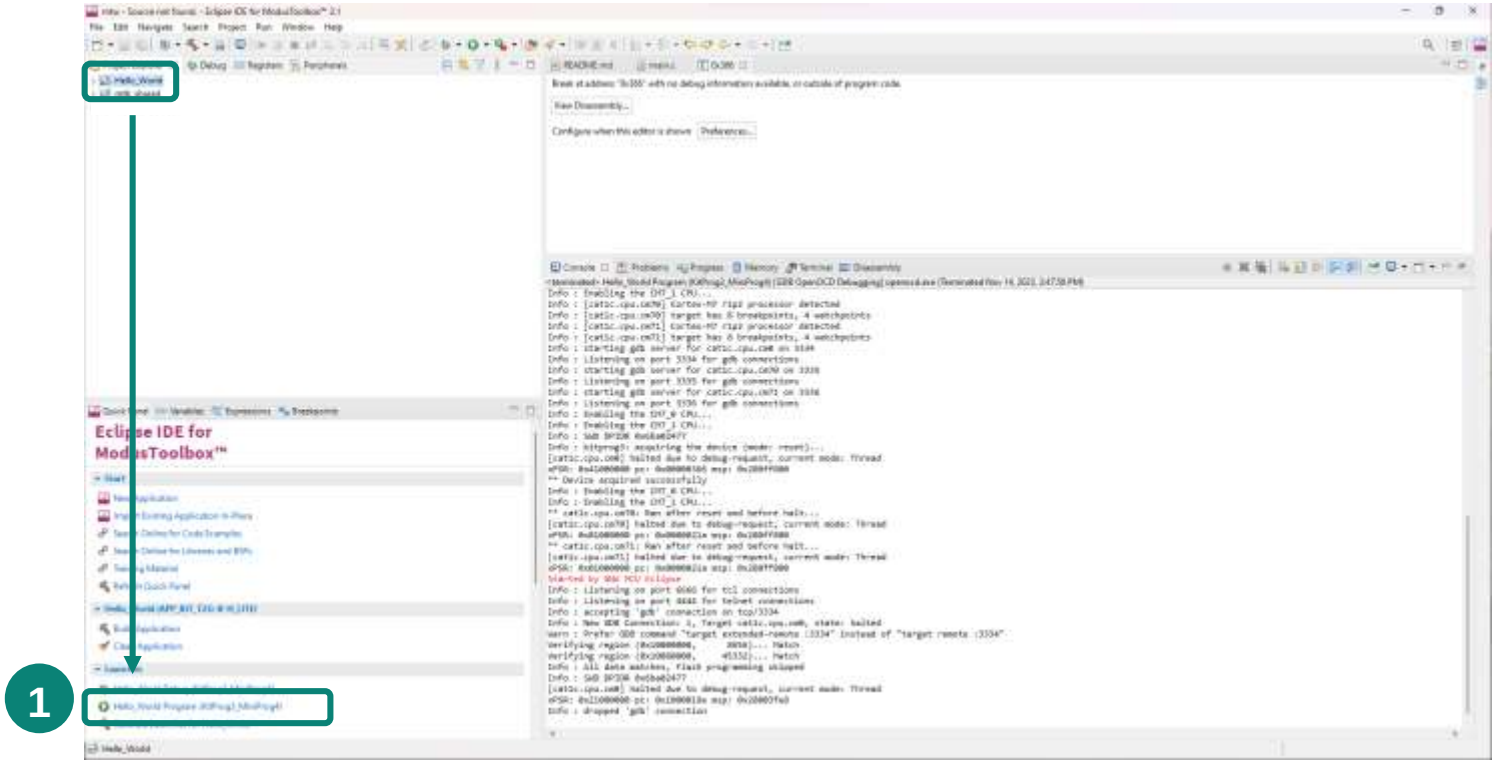
1. Select Hello World application, then click the “**Build Application**” in the Quick panel
2. Confirm build window for successful completion without error

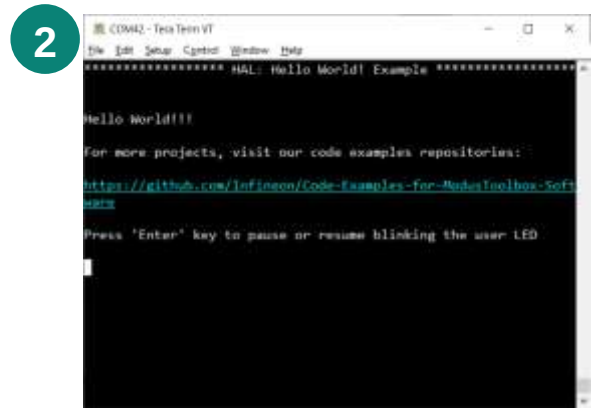


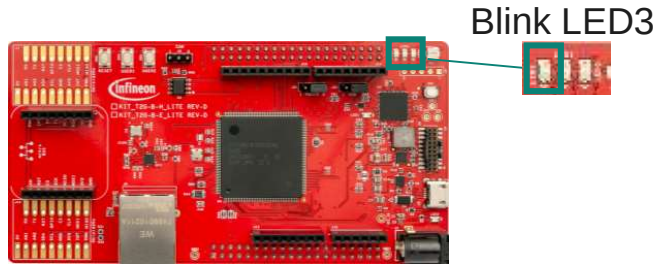
Hello World Application

– **Step 3**: Program and Test “**Hello World”**

1. Select “Hello World” application, then click the “**Hello_World Program (KitProg3_MiniProg4)**” in the Quick panel under **Launches**.
2. You can observe the message on the Tera Term, and blinking LED3 on the board



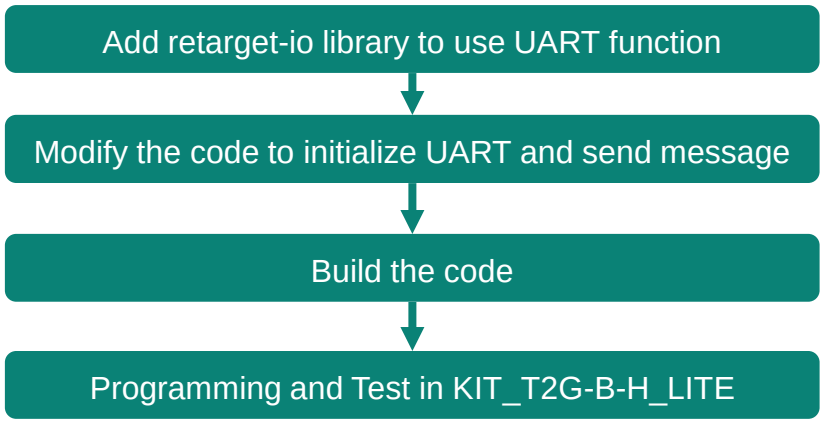


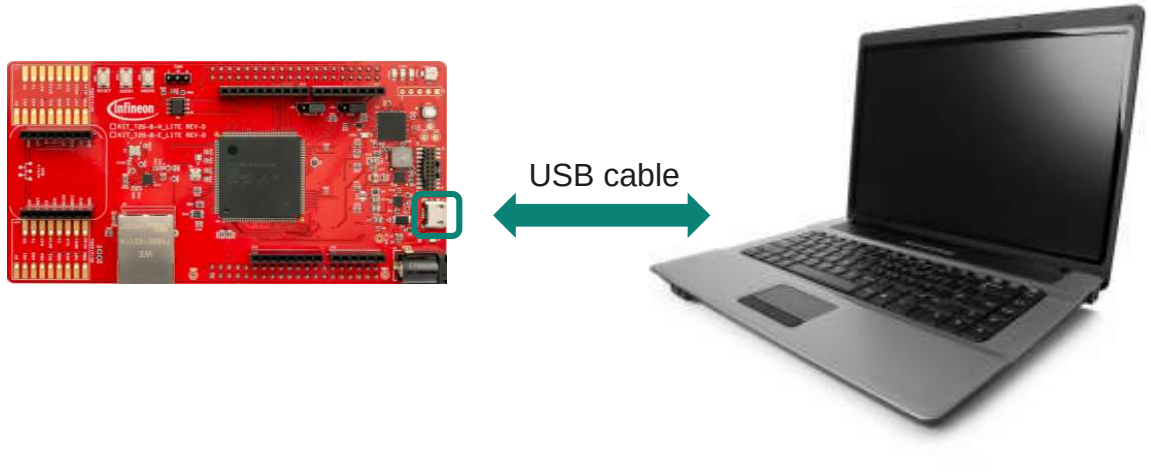


Create Basic Multicore Application

Create Basic Multicore Application- Overview

- **What you learn this exercise**
  - You will learn how to create application to display the message to terminal software
  - You will learn how to add new library using **Library manager**
- Create “MyTRAVEO” application from “Multicore” application
- In this exercise, we will use UART in the application to transmit data from MCU to PC.
- Add “retarget-io” library using Library manager
  - retarget-io: A utility library to retarget the standard input/output (STDIO) messages to a UART port. You can directly print messages on a UART terminal using ***printf()***.
- UART is controlled by **CM0+**

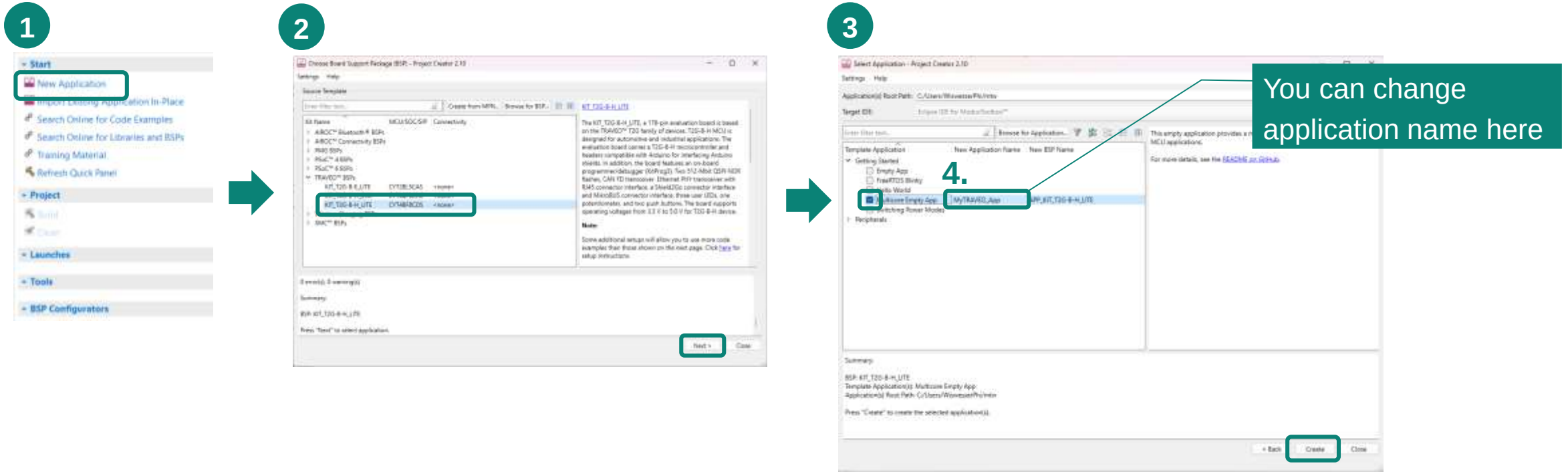




Basic Multicore Application

– Create “**MyTRAVEO**” application

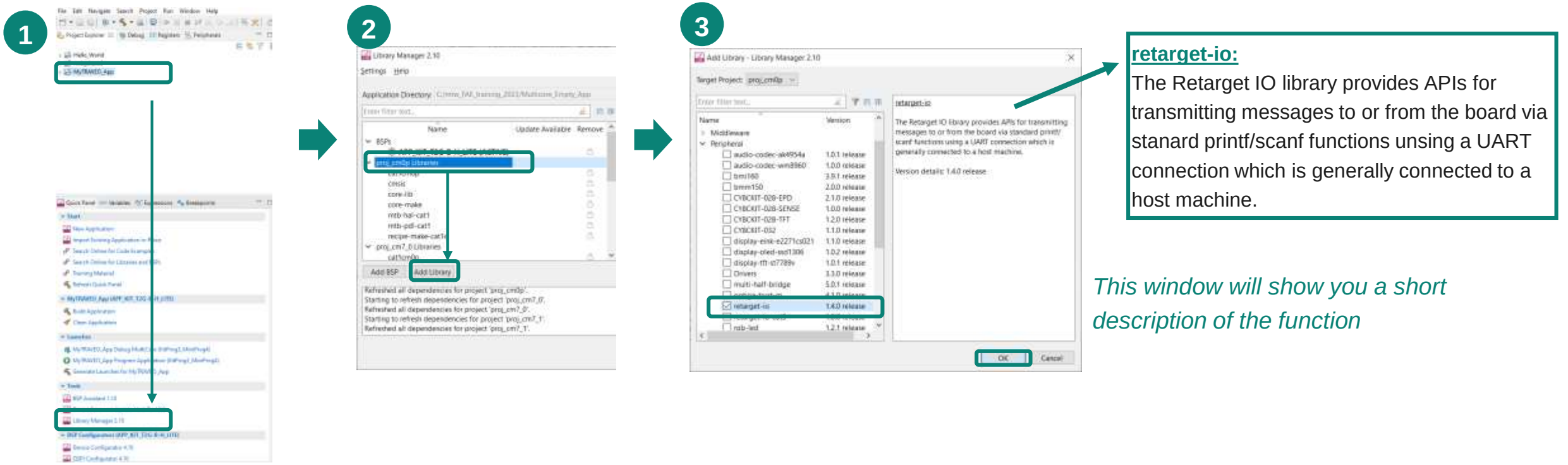
1. Click the New Application in the Start. After that, open the Project creator
2. Select “**KIT_T2G-B-H_LITE**” under **TRAVEO™ BSPs** in the Choose Board Support Package (BSP) window, and click “**Next**” button
3. Select “Multicore Empty App” in Getting Started
4. Change the name to **“MyTRAVEO_App”** then click the “**Create**” button



Basic Multicore Application - Add library

- Add “retarget-io” library to MyTRAVEO_App application
  - You can specify the TX pin, RX pin, and the baud rate through the ***cy_retarget_io_init()*** function.
  - In this session, retaget-io library adds to proj_cm0p Libraries

1. Select your application (Click the **MyTRAVEO_App**), then click the “**Library Manager 2.10**” in the Quick Panel
2. Select “**proj_cm0p Libraries**” and click the “**Add Library**” button
3. Select “**retarget-io**” in **peripheral tab**, then click the “**OK**” button



retarget-io:
The Retarget IO library provides APIs for transmitting messages to or from the board via stanard printf/scanf functions unsing a UART connection which is generally connected to a host machine.

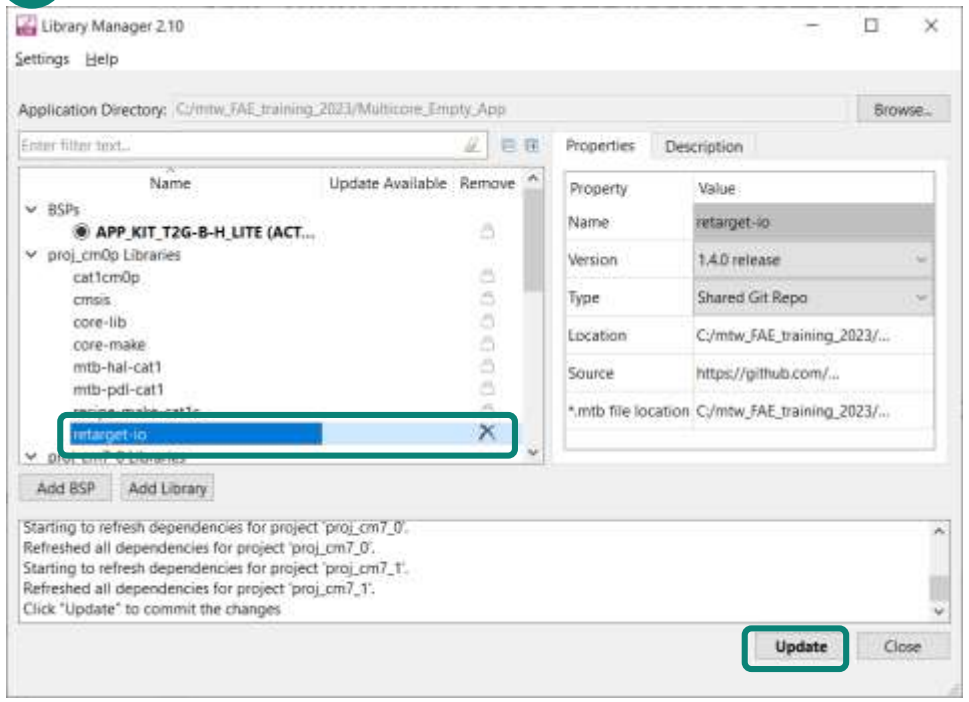
This window will show you a short description of the function

Basic Multicore Application - Add library

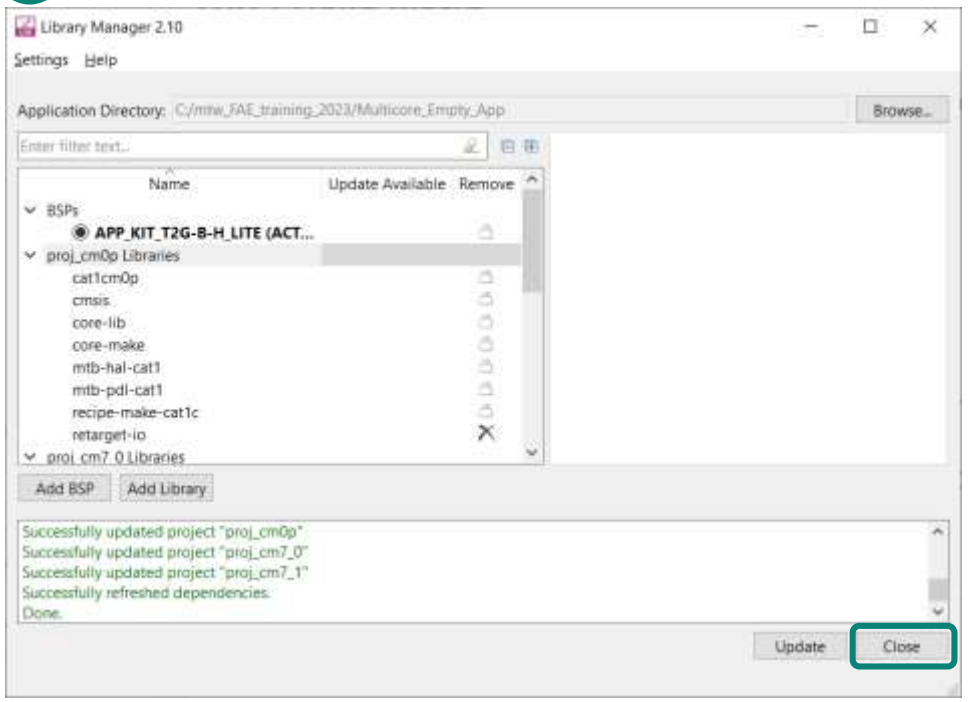
– Add “retarget-io” library to MyTRAVEO_App

4. Make sure “**retarget-io**” has been added, then click the “**Update**” button
5. Click the “**Close**” button **after update completion**

4

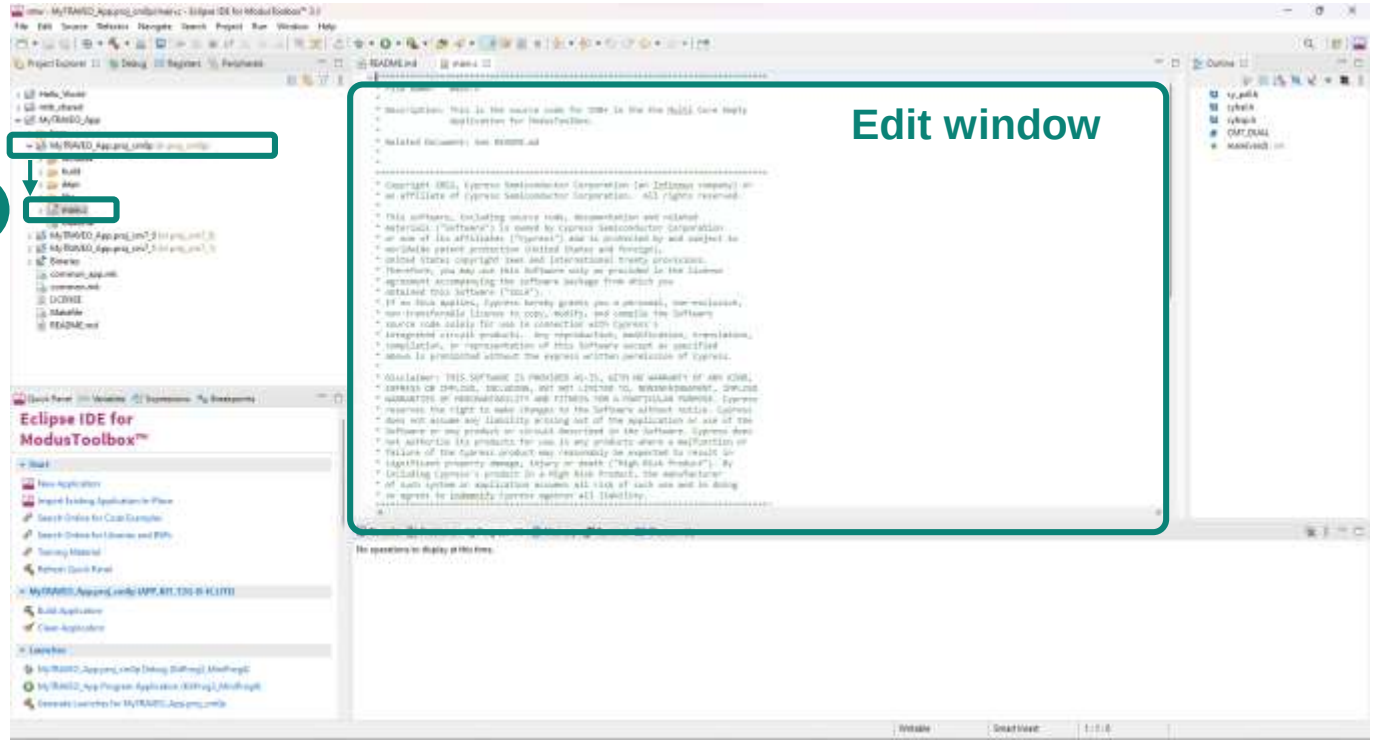


5



Basic Multicore Application - Coding

1. Double-click the "***main.c***" of MyTRAVEO_App.**proj_cm0p** in the MyTRAVEO_App project to open the edit window



2. Include header file for retarget-io

   It provides APIs for transmitting messages to/from the terminal via standard "***printf()***" / "***scanf()***" functions.

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

2

In this document, black characters are already in the main.c

Add the green text yourself to main.c

Following Exercise has same manner

Basic Multicore Application - Coding

3. Add function to initialize retarget-io to use the UART port.
4. Add code of check initialization result
5. Add functions to display on your terminal
6. Comment out to enter DeepSleep power mode

```
__enable_irq();

3  result = cy_retarget_io_init(CYBSP_DEBUG_UART_TX, CYBSP_DEBUG_UART_RX,
                                CY_RETARGET_IO_BAUDRATE);

4  if (result != CY_RSLT_SUCCESS)
   {
      CY_ASSERT(0);
   }

/* Enable CM7_0/1. CY_CORTEX_M7_APPL_ADDR is calculated in linker script, check it
in case of problems. */
Cy_SysEnableCM7(CORE_CM7_0, CY_CORTEX_M7_0_APPL_ADDR);

#if CM7_DUAL
   Cy_SysEnableCM7(CORE_CM7_1, CY_CORTEX_M7_1_APPL_ADDR);
#endif /* CM7_DUAL */

5  printf("\x1b[2J\x1b[;H");
   printf("Hello World!!!\r\n\n");
   printf("Welcome to TRAVEO Training!!!\r\n\n");

for(;;)
{
6     /* Cy_SysPm_CpuEnterDeepSleep(CY_SYSPM_WAIT_FOR_INTERRUPT); */
}
```

- Initialization function for redirecting low level IO commands to allow sending messages over a UART interface.
- ***cy_retarget_io_init*** (tx, rx, baudrate)
  tx: UART TX pin
  rx: UART RX pin
  baudrate: UART baudrate
- You can use a different baud rate if you prefer.

- Check if initialization is success
  It is recommended manner to create application on MTB

- Specify the character to display on the terminal

- Comment out of enter DeepSleep function

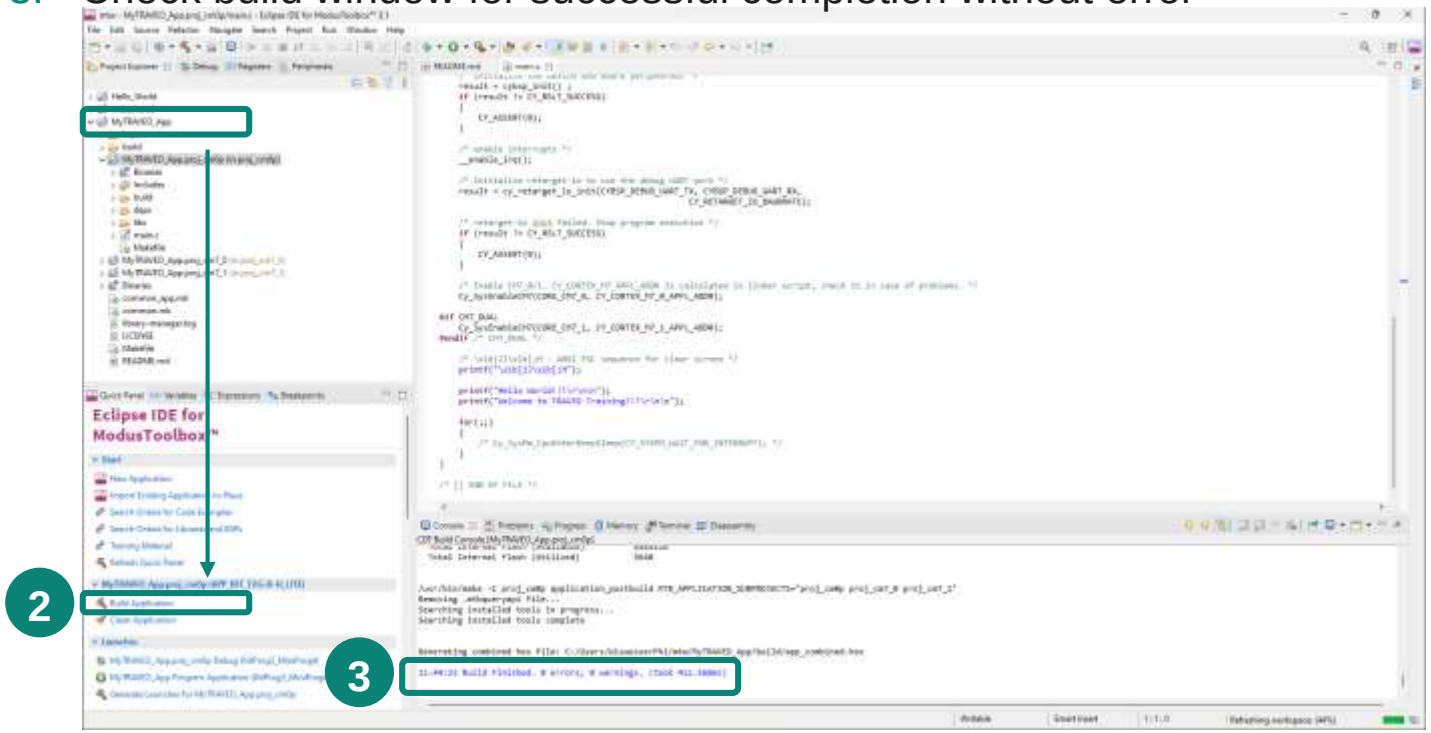
Hint

- **CYBSP_DEBUG_UART_TX** and **CYBSP_DEBUG_UART_RX** pins are already defined by following header file in the BSP (C:/your work space/MyTRAVEO_App/bsps/TARGET_APP_KIT_T2G-B-H_LITE/config/GeneratedSource/cycfg_pins.h)
- **CY_RETARGET_IO_BAUDRATE** is already set to 115200 by following header file. (C:/your work space/mtb_shared/retarget-io/release-v1.4.0/cy_retarget_io.h)
- *For the **final code** please go to the last slide of this exercise (slide 30)*

Basic Multicore Application - Build

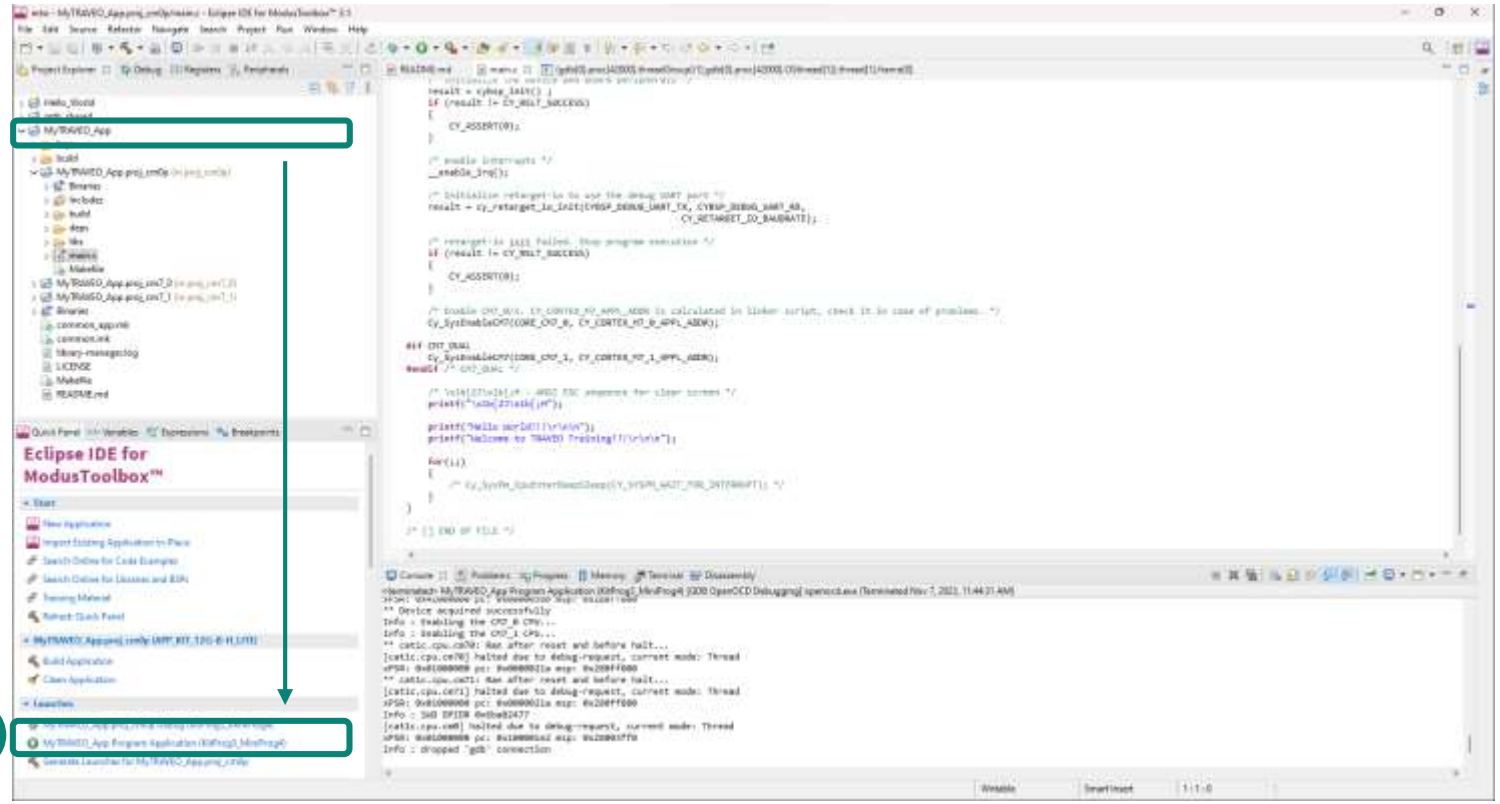
1. After editing, click “**save**” button to save the changes

2. Select your application (MyTRAVEO_App), then click the “**Build Application**” in the Quick panel
3. Check build window for successful completion without error

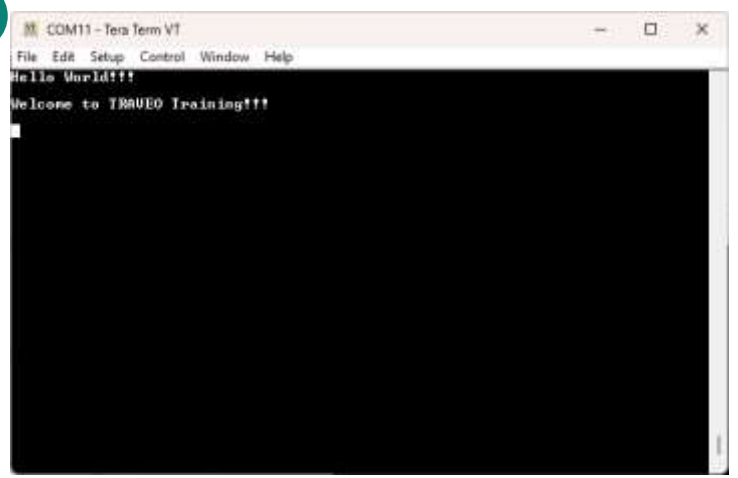


Basic Multicore Application - Flash

1. Select your application (Click the MyTRAVEO App), then click the “**MyTRAVEO_App Program Application (KitProg3_MiniProg4)**” in the Quick panel. After programming, run the program automatically
2. You can observe the message on the Tera Term



2

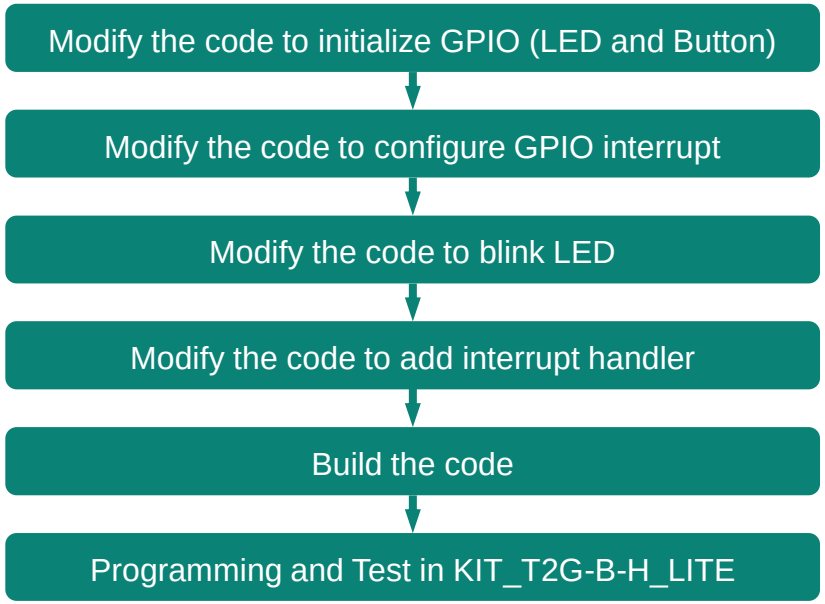


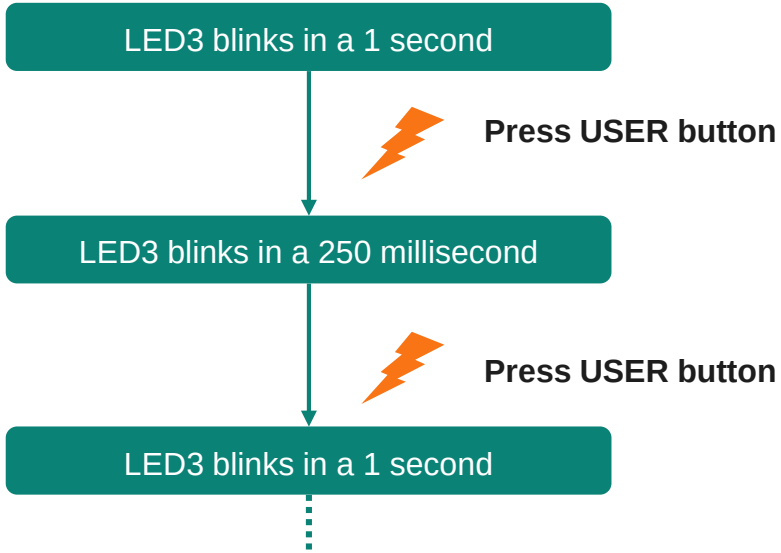
Click here to see the final code for this Exercise

GPIO handling – LED blinking

Create GPIO handling application using HAL - Overview

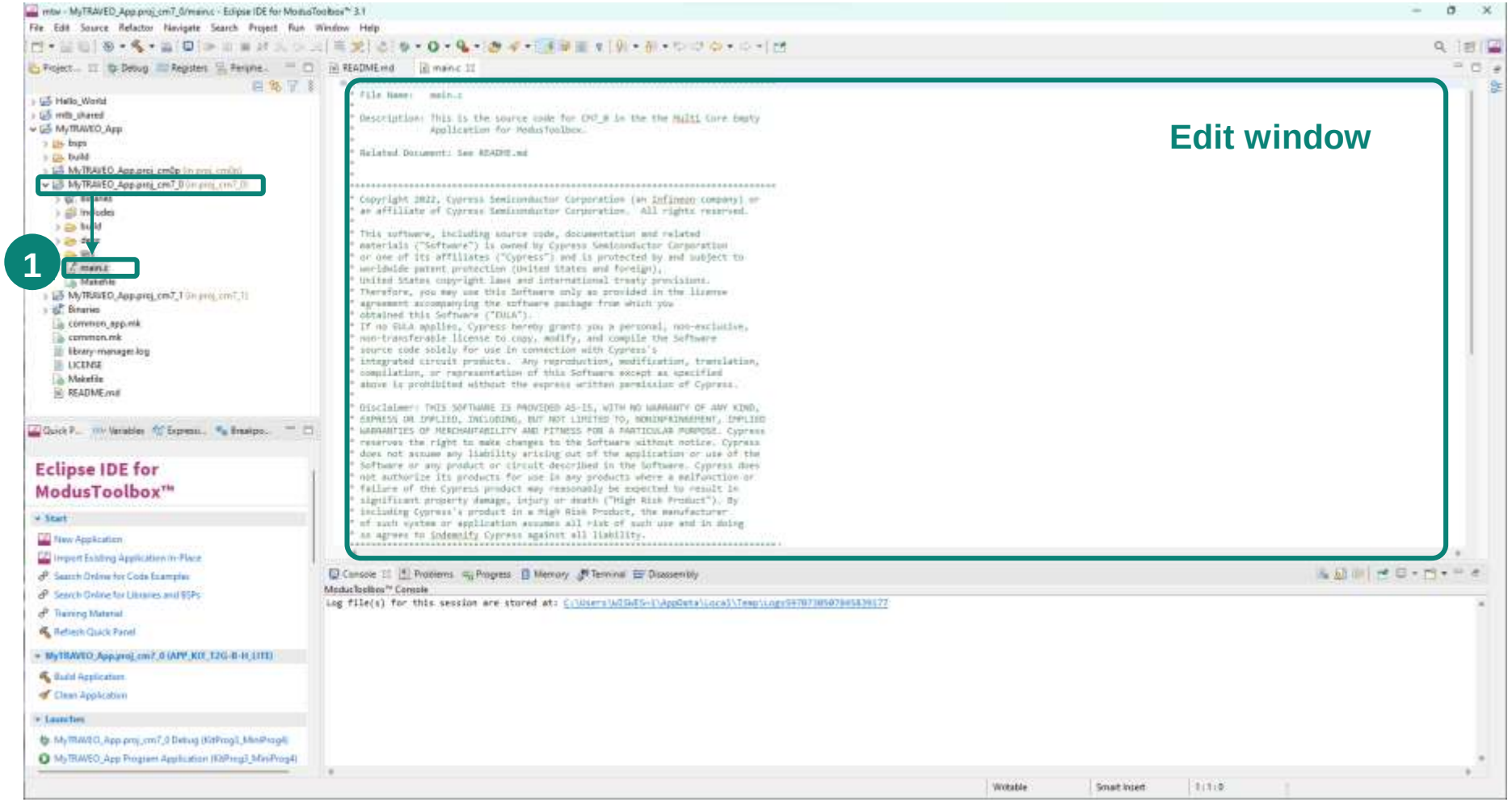
- **What you learn this exercise**
  - You will learn how to create application using **HAL (Hardware Abstraction Layer)**
  - HAL function shows **"*cyhal_xxxx*"**
- In this exercise, we will use HAL function to configure GPIO, and blink LED (LED3)
- First, LED blinks in a 1 second cycle
- When the USER button (USER1) is pressed, a GPIO interrupt is generated and the LED flashes in a cycle of 250 ms
- Each time the USER button is pressed, the flashing cycle of the LED changes.
- In this exercise LED3 is controlled by **CM7_0**





GPIO handling – LED blinking

1. Double-click the “***main.c***” of MyTRAVEO_App.**proj_cm7_0** in the MyTRAVEO App project to open the edit window



GPIO handling – LED blinking

2. Add definitions
3. Add following variables
   "**gpio_intr_flag**" indicates interrupt detection, and "**gpio_btn_callback_data**" stores interrupt handler address
4. Add interrupt handler declarations
5. Add local variable and initialization in ***main()*** function

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

2
```
#define DELAY_MS          (500)
#define DELAY_MS_FAST     (125)
#define GPIO_INTERRUPT_PRIORITY (7u)
```

- **DELAY_MS**: defines wait time for normal blinking
- **DELAY_MS_FAST**: defines wait time for fast blinking
  The number specifies the wait time in ms
- **GPIO_INTERRUPT_PRIORITY:** defines USER button interrupt priority

3
```
volatile bool gpio_intr_flag = false;
cyhal_gpio_callback_data_t gpio_btn_callback_data;
```

- **gpio_intr_flag**: indicate interrupt detection
- **gpio_btn_callback_dat**: store interrupt handler address

4
```
static void gpio_interrupt_handler(void *handler_arg, cyhal_gpio_event_t event);
```

- **Interrupt handler declarations**

```
int main(void)
{
    cy_rslt_t result;
```

5
```
    uint32_t delay_led_blink = DELAY_MS;
```

- **Set local variable and initialize to DELAY_MS**

Hint: For the final code please go to the last page of the exercise.

GPIO handling – LED blinking

6. Add function to initialize GPIO port to use LED output and button input, and add code of check initialization result
7. Add function to register interrupt handler
8. Add function to enable USER button interrupt

```
__enable_irq();

6
result = cyhal_gpio_init(CYBSP_USER_LED, CYHAL_GPIO_DIR_OUTPUT,
            CYHAL_GPIO_DRIVE_STRONG, CYBSP_LED_STATE_OFF);

if (result != CY_RSLT_SUCCESS)
{
   CY_ASSERT(0);
}

result = cyhal_gpio_init(CYBSP_USER_BTN, CYHAL_GPIO_DIR_INPUT,
            CYBSP_USER_BTN_DRIVE, CYBSP_BTN_OFF);
if (result != CY_RSLT_SUCCESS)
 {
    CY_ASSERT(0);
 }

7
gpio_btn_callback_data.callback = gpio_interrupt_handler;
cyhal_gpio_register_callback(CYBSP_USER_BTN,
                  &gpio_btn_callback_data);

8
cyhal_gpio_enable_event(CYBSP_USER_BTN, CYHAL_GPIO_IRQ_FALL,
                  GPIO_INTERRUPT_PRIORITY, true);

for (;;)
```

- Initialize the GPIO pin
- ***cyhal_gpio_init (pin, direction, drive_mode, init_val)***
  pin: GPIO pin to initialize
  direction: pin direction
  drive_mode: pin drive mode
  init_val: Initial value on the pin
- Check if initialization is success

- Register/clear an interrupt handler for pin events
- ***cyhal_gpio_register_callback (pin, callback_data)***
  pin: GPIO pin
  callback_data: The callback data to register. Use NULL to unregister.

- Enable or Disable the specified GPIO event.
- ***cyhal_gpio_enable_event (pin, event, intr_priority, enable)***
  pin: GPIO pin
  event: GPIO event
  intr_priority: Priority for NVIC interrupt events.
  enable: true (Enable interrupts) / false (Disable interrupt)

Hint

- **CYHAL_GPIO_xxx** is already defined in the HAL header file.
- **CYBSP_xxx** is already defined in the BSP header file.

GPIO handling – LED blinking

9. Add LED control code
   Change the wait time, USER button interrupt is detected each time
10. Add function to write the GPIO port and wait LED blink cycle
11. Add interrupt handler under *main()* function

```
for (;;)
{
    if (true == gpio_intr_flag)
    {
        gpio_intr_flag = false;
        /* Update LED toggle delay */
        if (DELAY_MS == delay_led_blink)
        {
            delay_led_blink = DELAY_MS_FAST;
        }
        else
        {
            delay_led_blink = DELAY_MS;
        }
    }
```

9

- Check if interrupt detected
- Interrupt flag set to false when interrupt detected
- Switch the wait time

```
    cyhal_gpio_write(CYBSP_USER_LED, CYBSP_LED_STATE_ON);
    cyhal_system_delay_ms(delay_led_blink);
    cyhal_gpio_write(CYBSP_USER_LED, CYBSP_LED_STATE_OFF);
    cyhal_system_delay_ms(delay_led_blink);
}
}
```

10

- Set the output value for the pin
- ***cyhal_gpio_write (pin, value)***
  pin: GPIO pin
  value: The value to be set (high = true, low = false)

- Requests that the current operation delays for at least the specified length of time.
- ***cyhal_system_delay_ms (milliseconds)***
  milliseconds: The number of milliseconds to delay for

```
static void gpio_interrupt_handler(void *handler_arg, cyhal_gpio_event_t event)
{
    gpio_intr_flag = true;
}
```

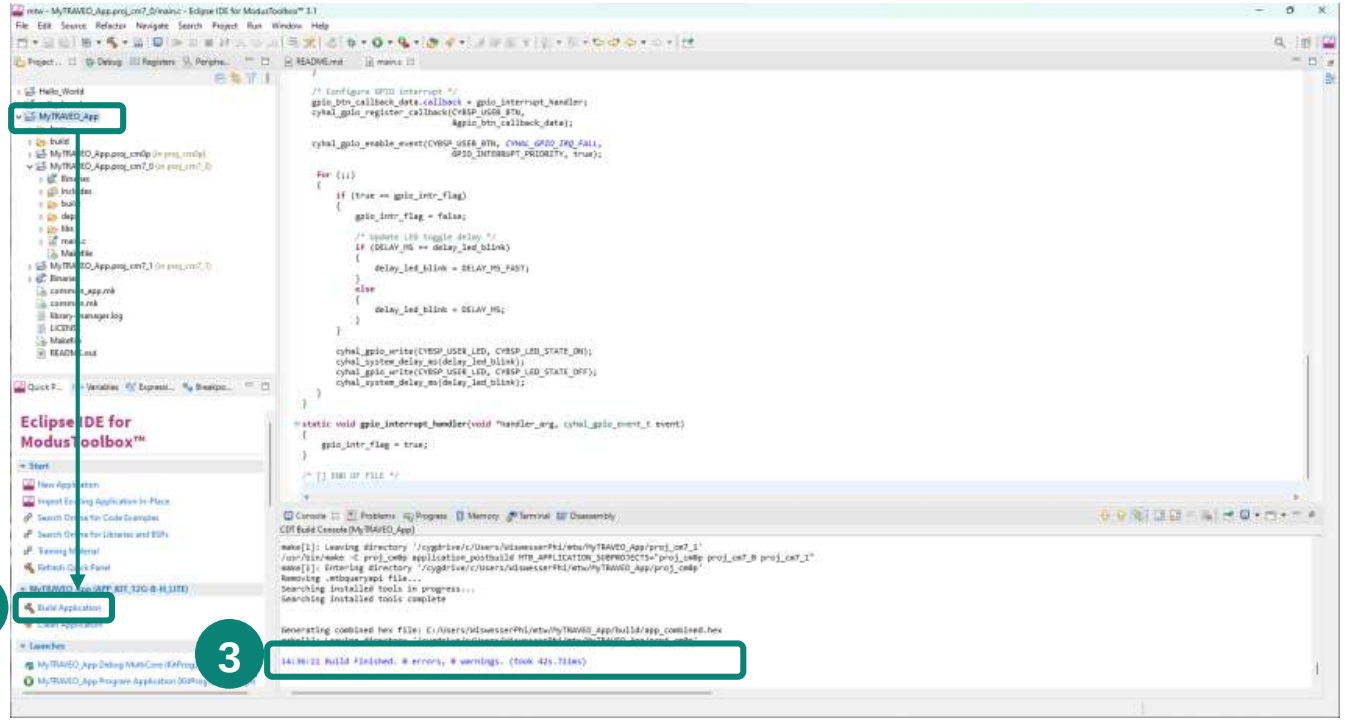
11

- Set the “gpio_intr_flag” to true, when USER button interrupt occur

GPIO handling – LED blinking

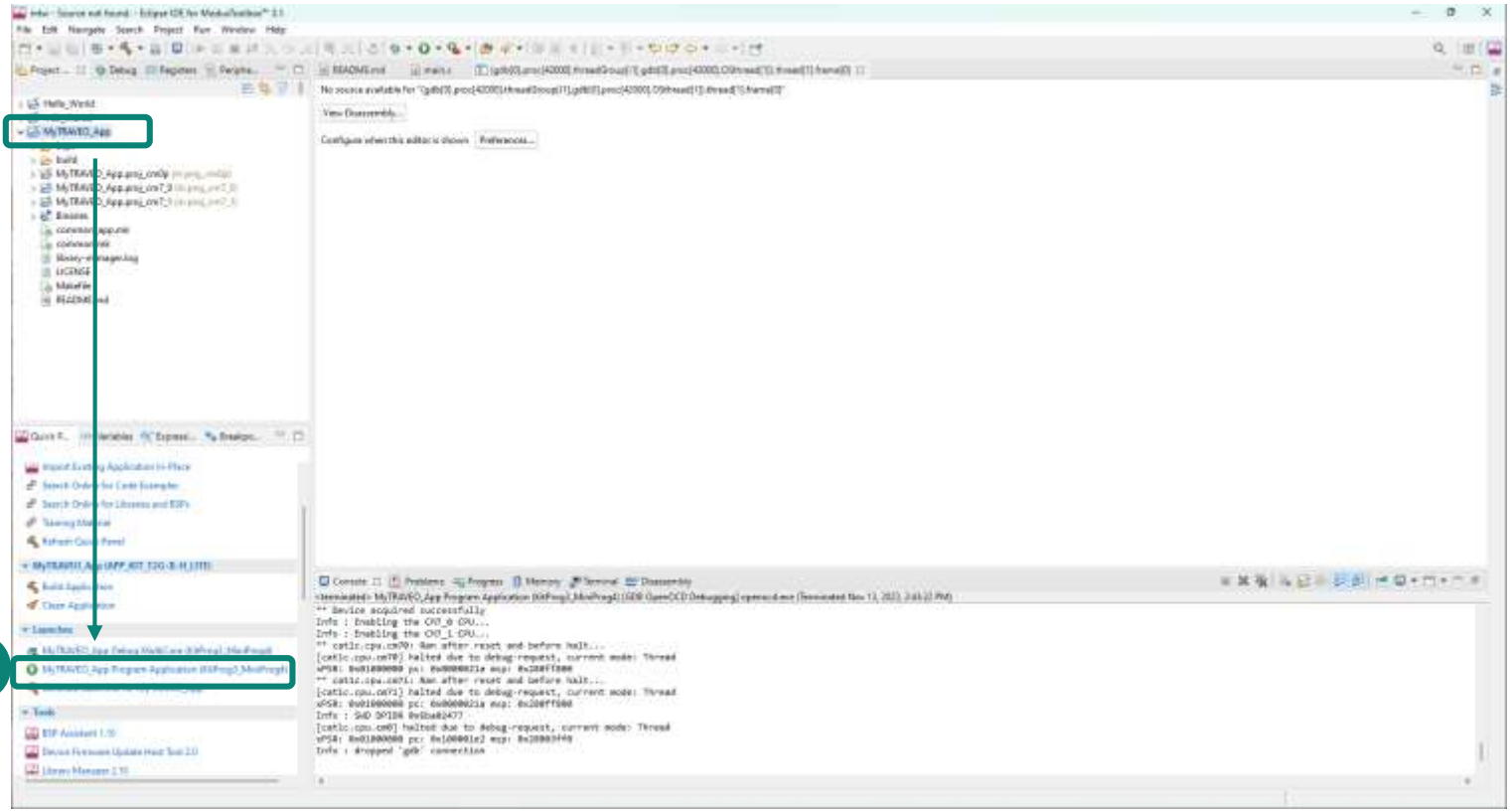
1. After editing, click “**save**” button to save the changes

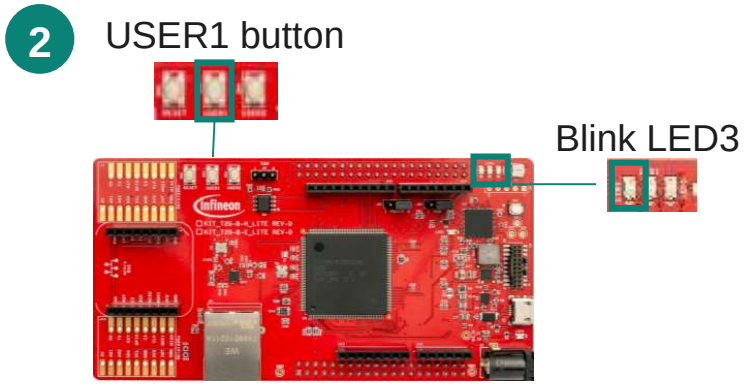
2. Select your application (MyTRAVEO_App), then click the “**Build Application**” in the Quick panel
3. Check build window for successful completion without error



GPIO handling – LED blinking

1. Select your application (Click the MyTRAVEO_App), then click the “**MyTRAVEO_App Program Application (KitProg3_MiniProg4)**” in the Quick panel. After programming, run the program automatically
2. You can observe blinking LED (LED3)
   Also, you can verify that the LED blinking speed changes by pressing the USER1 button each time.





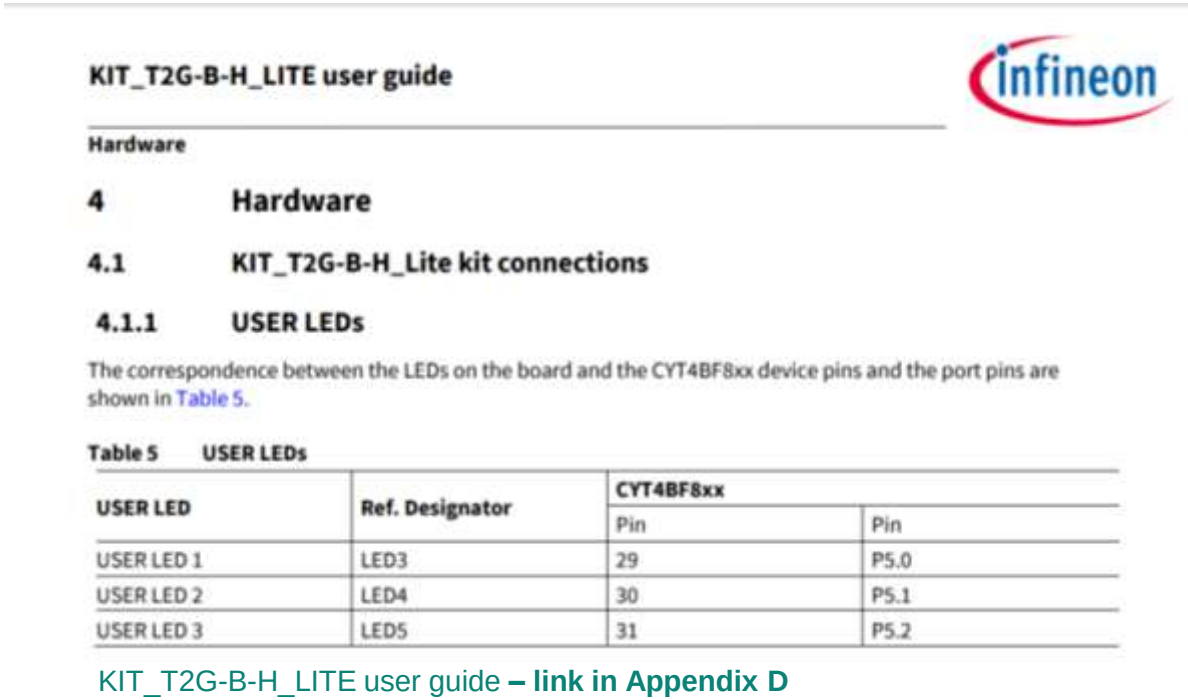
Click here to see the final code for this Exercise

LED / User Button Configuration

- HAL structure is used by HAL function, and defined by BSP (Board Support Package)
- In BSP, some functions are defined by device or board. e.g. LEDs, buttons, etc.
- These definitions can be used in HAL functions
- If you use a different board, just assign the defined name to the appropriate port, therefore your application will work without any source code changes.

- Default setup: The correspondence between the LEDs on the board and the CYT4BF8xx device pins and the port pins are pre defined and shown in Table 5

- **In the following slides you will learn how to change this using the device configurator**



KIT_T2G-B-H_LITE user guide

Hardware

4 Hardware

4.1 KIT_T2G-B-H_Lite kit connections

4.1.1 USER LEDs

The correspondence between the LEDs on the board and the CYT4BF8xx device pins and the port pins are shown in Table 5.

Table 5 USER LEDs

USER LED	Ref. Designator	CYT4BF8xx	
		Pin	Pin
USER LED 1	LED3	29	P5.0
USER LED 2	LED4	30	P5.1
USER LED 3	LED5	31	P5.2

KIT_T2G-B-H_LITE user guide **– link in Appendix D**

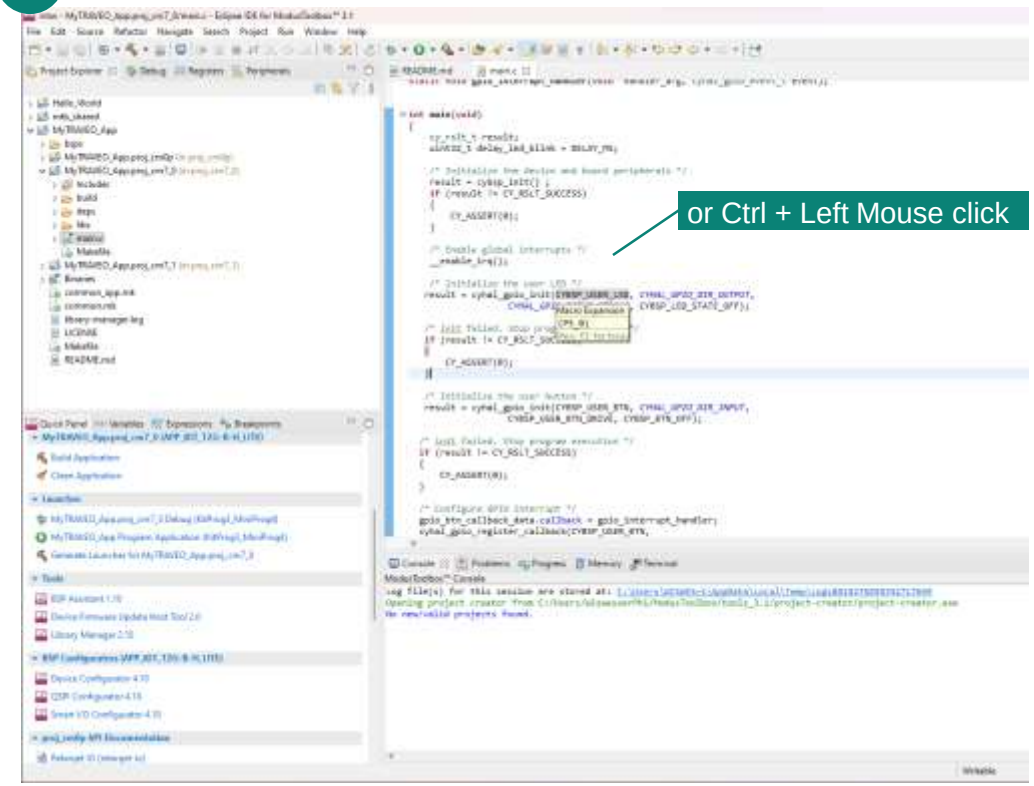
LED / User Button Configuration

You can see the **assigned pin cofiguration of each function** by:

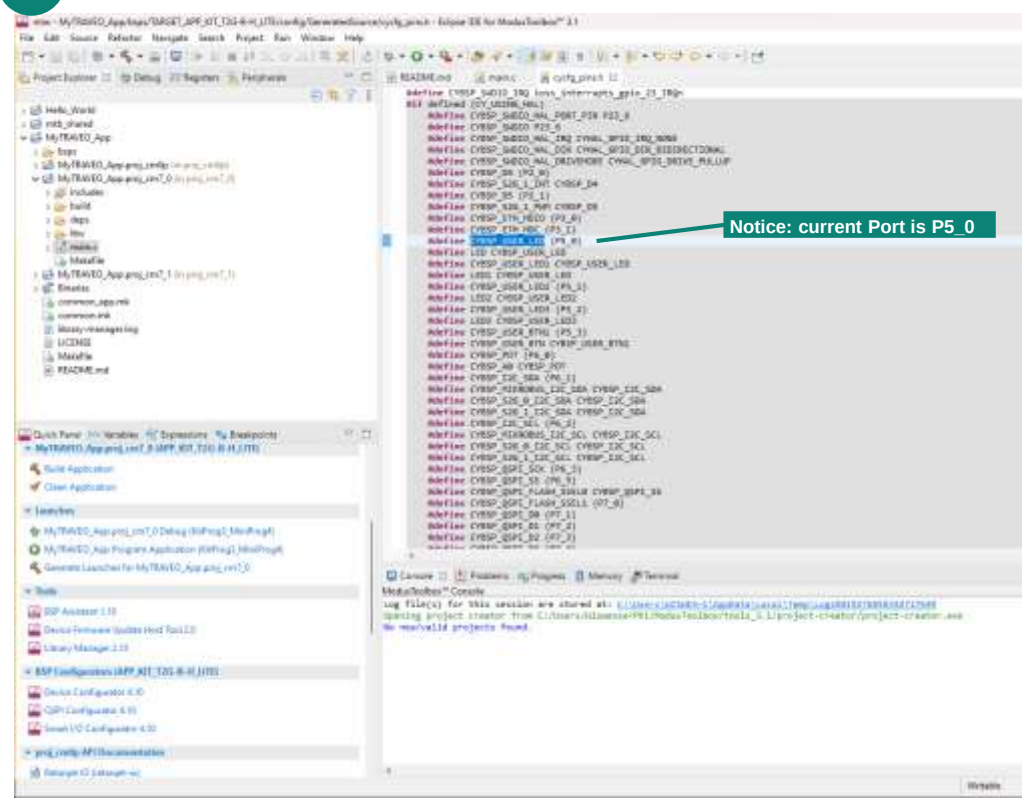
1. **hovering** over the CYBSP_USER_LED command in the edit window
2. or by **clicking Ctrl + Left Mouse Click** on the command which will get you to the cycfg_pins.h window

In the main.c

1



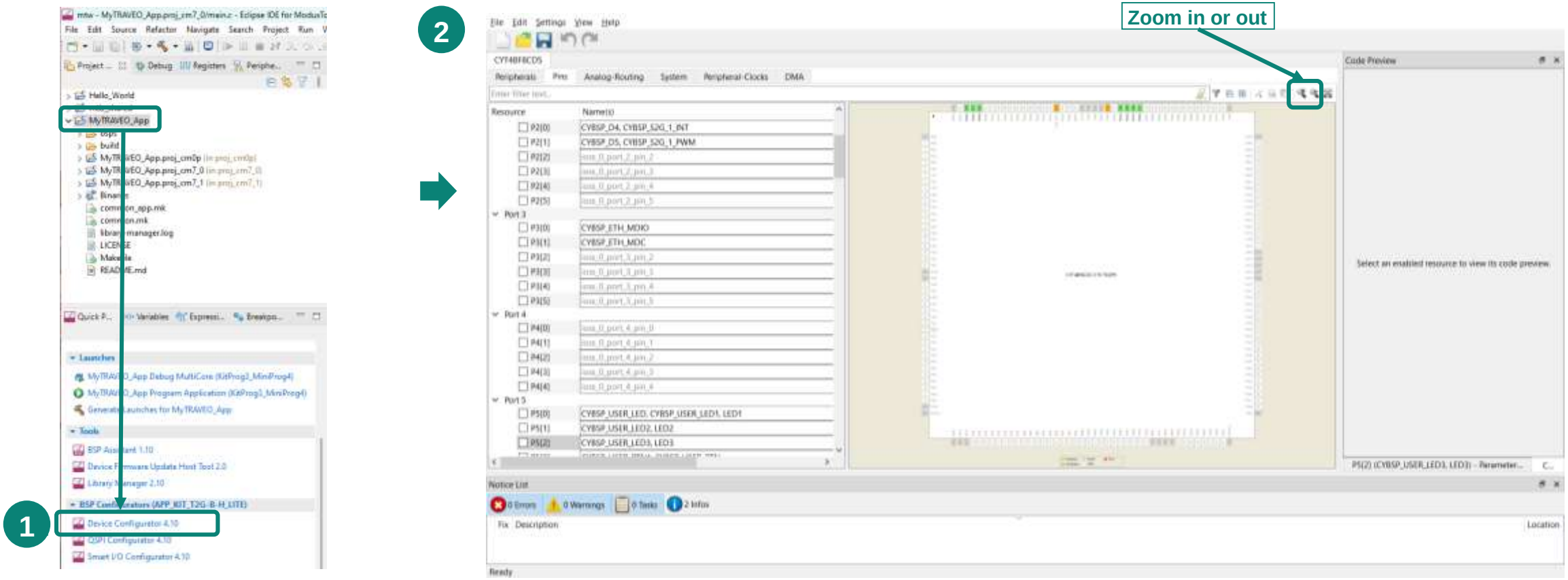
2



LED / User Button Configuration

How to change the **LED / User Button** using **Device Configurator**:

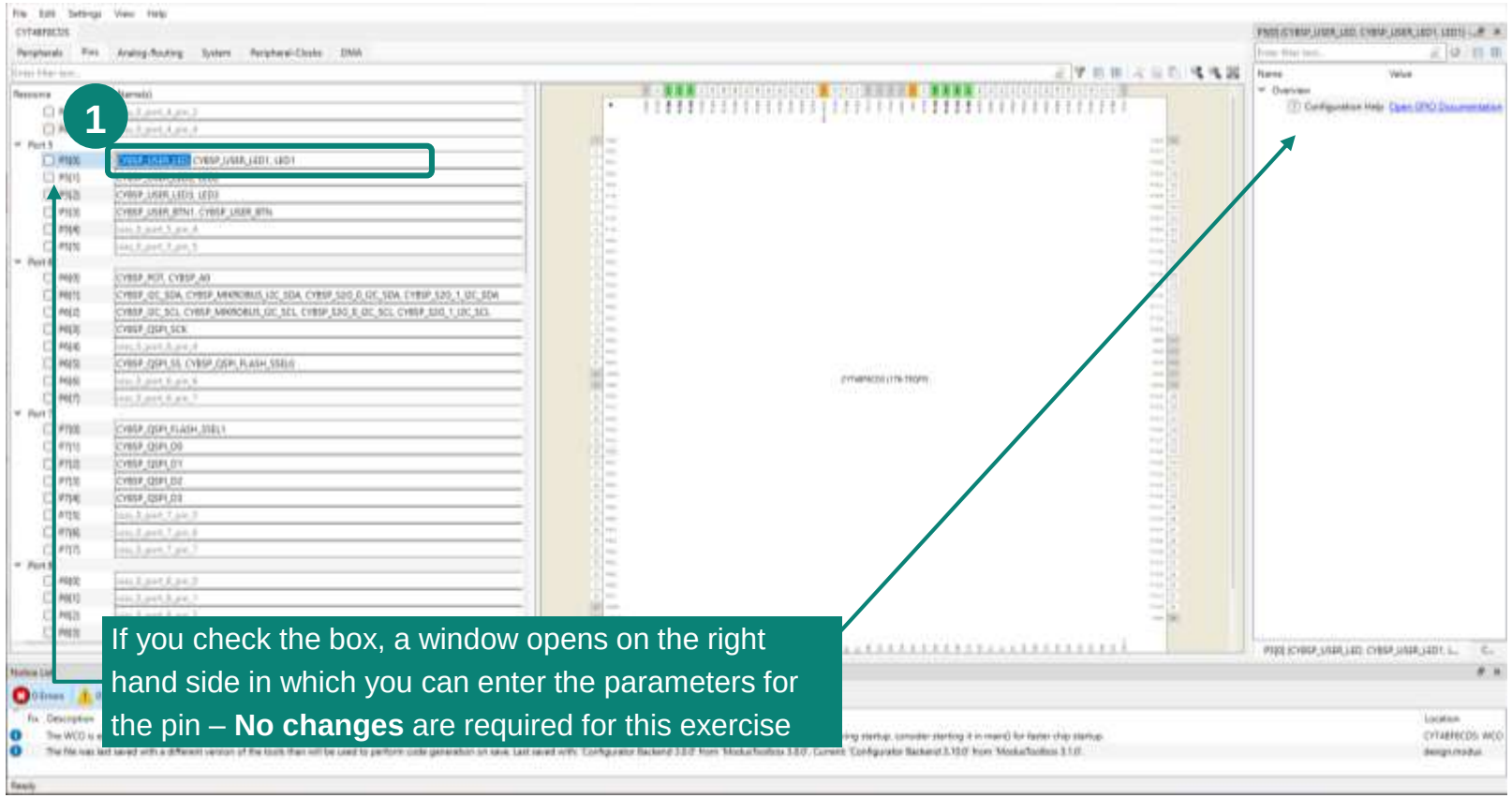
1. Select your application (Click the MyTRAVEO_App), then click the “**Device configurator 4.10**” in the Quick Panel
2. After that, open the Device configurator window and go to the **Pins tab**

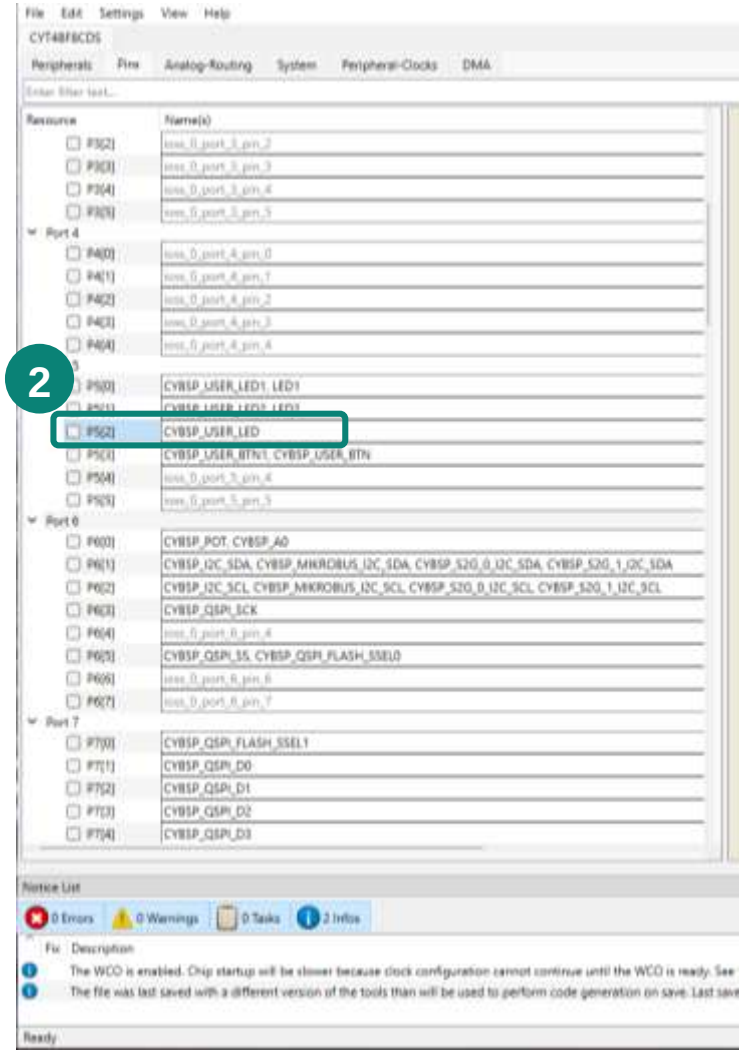


LED / User Button Configuration

Change blinking LED from LED3 to LED5:

1. **Cut** (Ctrl + x) „CYBSP_USER_LED“ **from P5[0]**
2. **Paste** (Ctrl + v) it in **P5[2]**

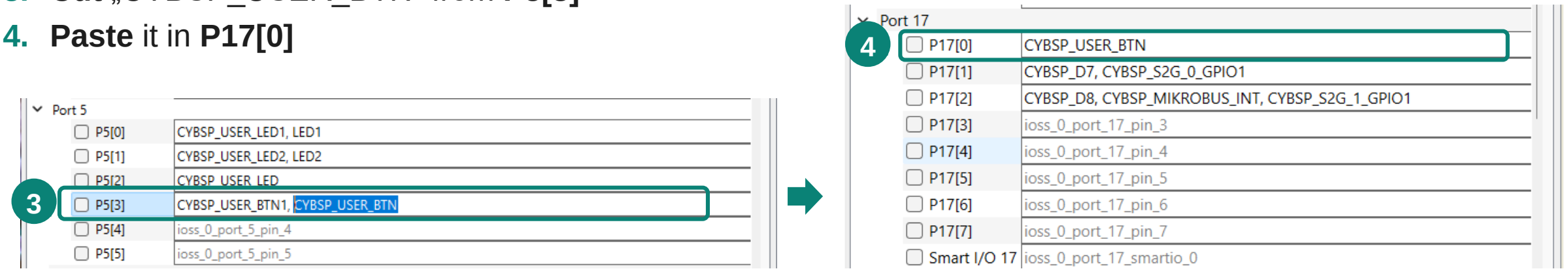




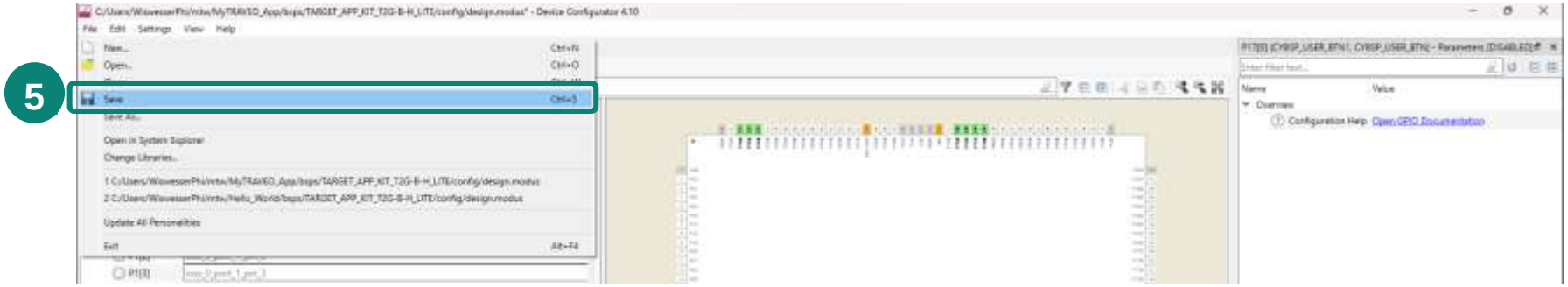
LED / User Button Configuration

Change User Button from USER1 to USER2:

3. **Cut** „CYBSP_USER_BTN“ from **P5[3]**
4. **Paste** it in **P17[0]**

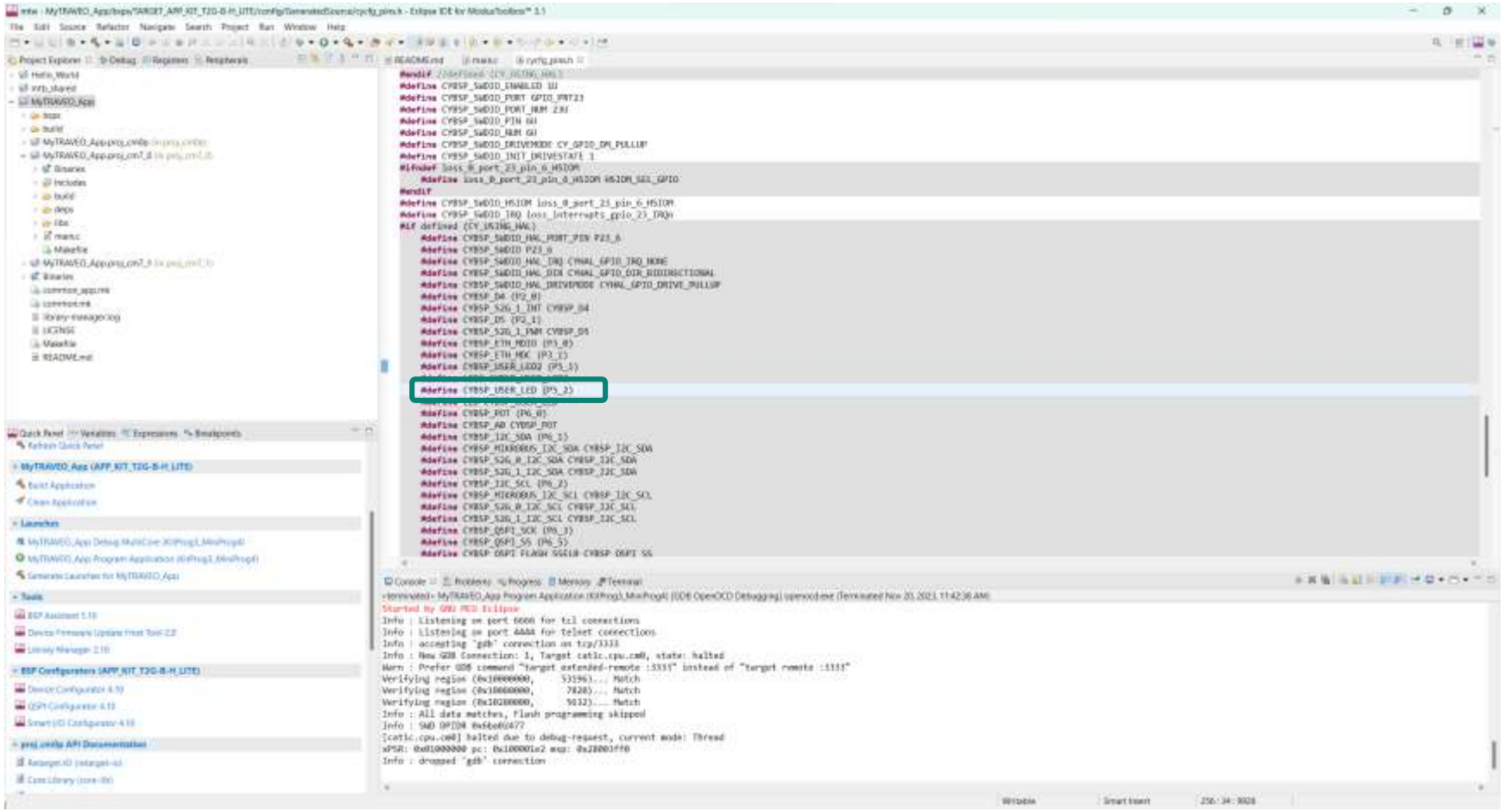


5. Now **save** the Device configuration by going to: **File** ➡ **Save**



LED / User Button Configuration

– After saving, the **header file (cycfg_pibs.h)** will be **updated** based on the device configuration

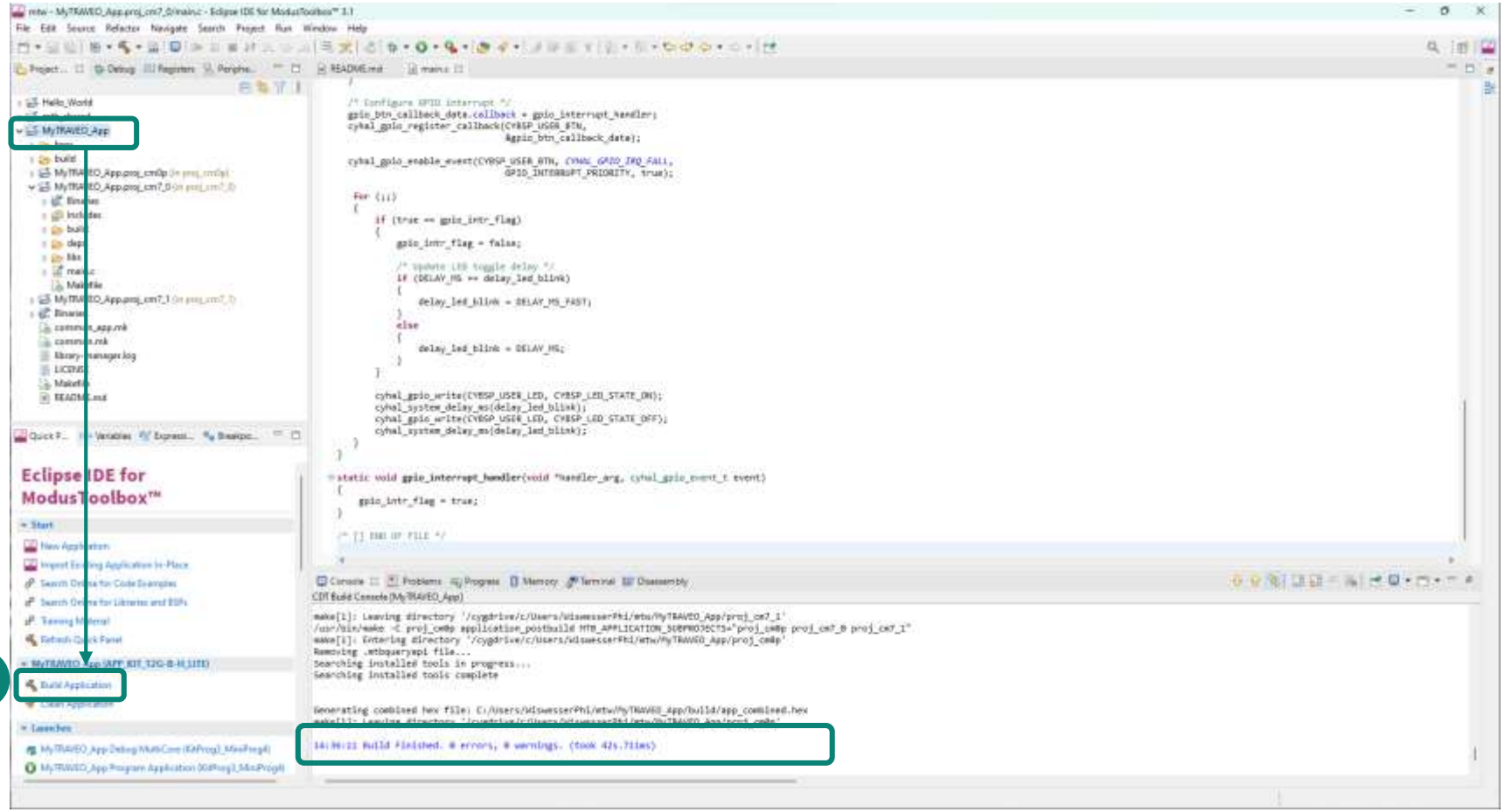


Before:
#define CYBSP_USER_LED (P5_0)

After:
#define CYBSP_USER_LED (P5_2)

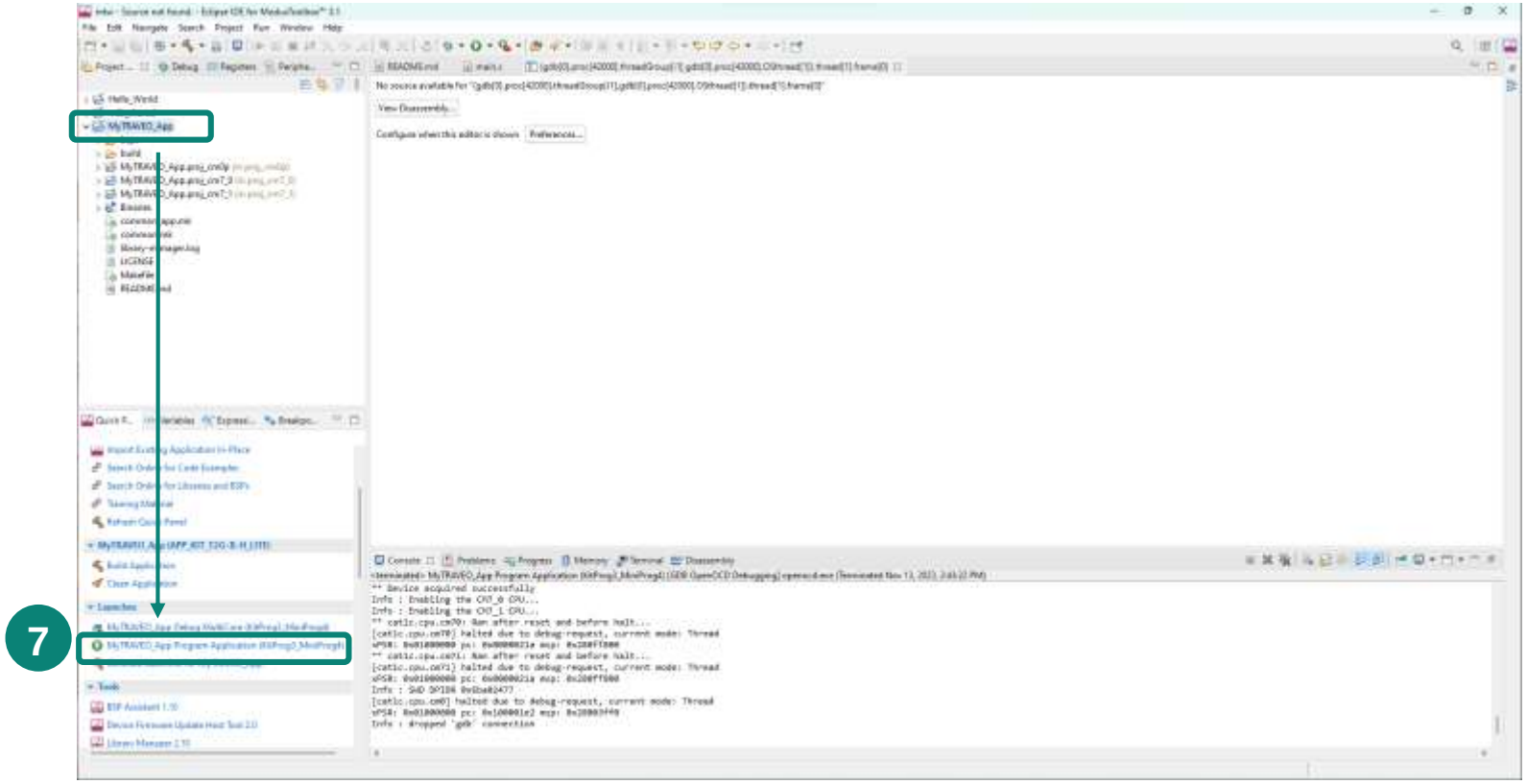
LED / User Button Configuration

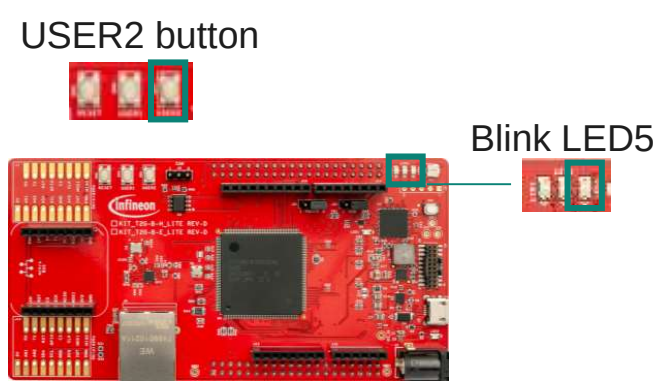
6. Select your application (MyTRAVEO_App), then click the “**Build Application**” in the Quick panel



LED / User Button Configuration

7. Select your application (Click the MyTRAVEO_App), then click the “**MyTRAVEO_App Program Application (KitProg3_MiniProg4)**” in the Quick panel.
8. You can observe blinking LED (now LED5 instead of LED3)
   Also, you can confirm that the LED blinking speed changes by pressing the USER2 button each time.





Appendix

Appendix A – ask ChatGPT

What is HAL ?

– The **Hardware Abstraction Layer (HAL)** is a software layer that serves as an interface between highler-level software and the hardware of a computing system.

– In the context of **GPIO** (General-Purpose Input/Output) handling, the HAL is used to abstract the low-level hardware details of interacting with GPIO pins. GPIO pins are often used for tasks like reading sensor data, controlling LEDs, or interfacing with external devices. The HAL provides a uniform way for software to interact with these pins regardless of the specific hardware platform.

What is PDL ?

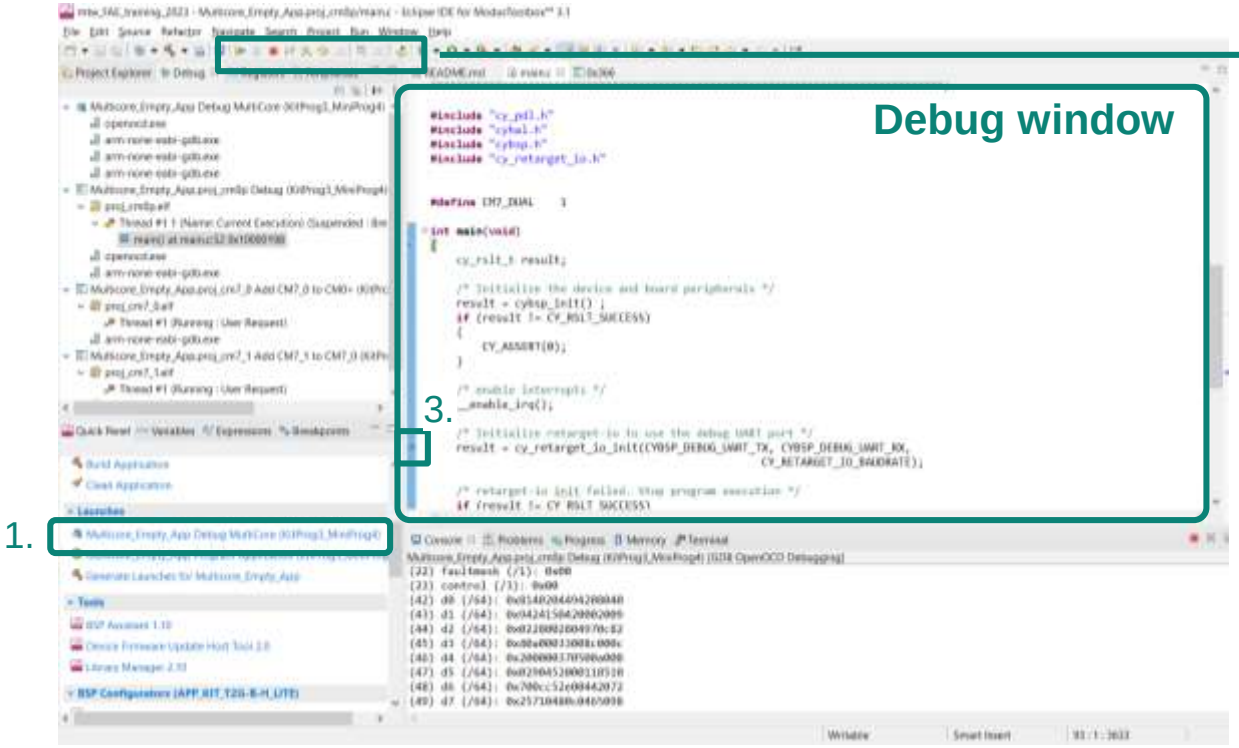
– The **Peripheral Driver Library (PDL)** is a set of software routines or functions provided to facilitate the interaction between a microcontroller and its peripherals, such as timers, GPIO, UART etc.

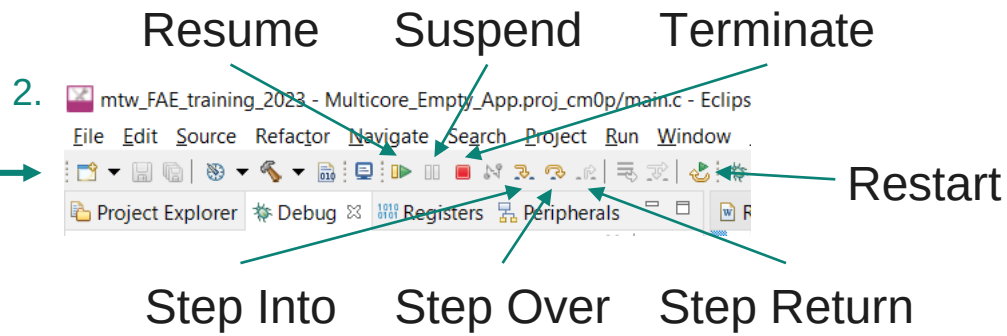
– These libraries make it easier for developers to configure and control the microcontrollers‘s peripherals without having to write low-level register manipulation code from scratch.

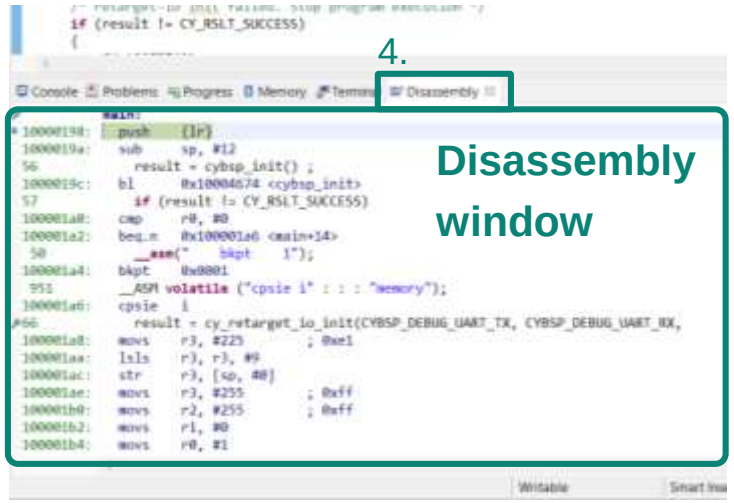
Appendix B - Debugging

– Debug connection and execution

1. When you want to debug your application, select your application (MyTRAVEO App), then click the “**MyTRAVEO_App Debug Multicore (KitProg3_MiniProg4)**” in the Quick panel. After that, open the debug window.
2. You can execute Resume, Suspend, Step Into, Step Over, Step Return, Terminate, and Restart.
3. Also, you can set a hardware breake point
4. Select Disassembly tab, you can debug with disassemble code



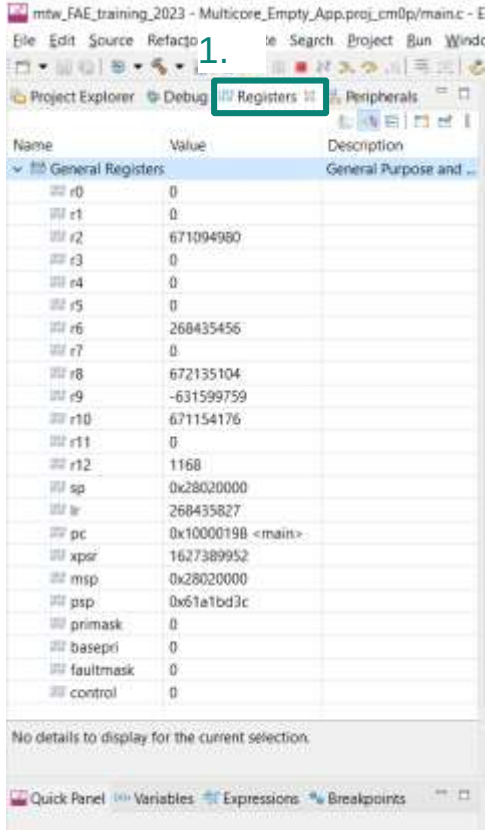


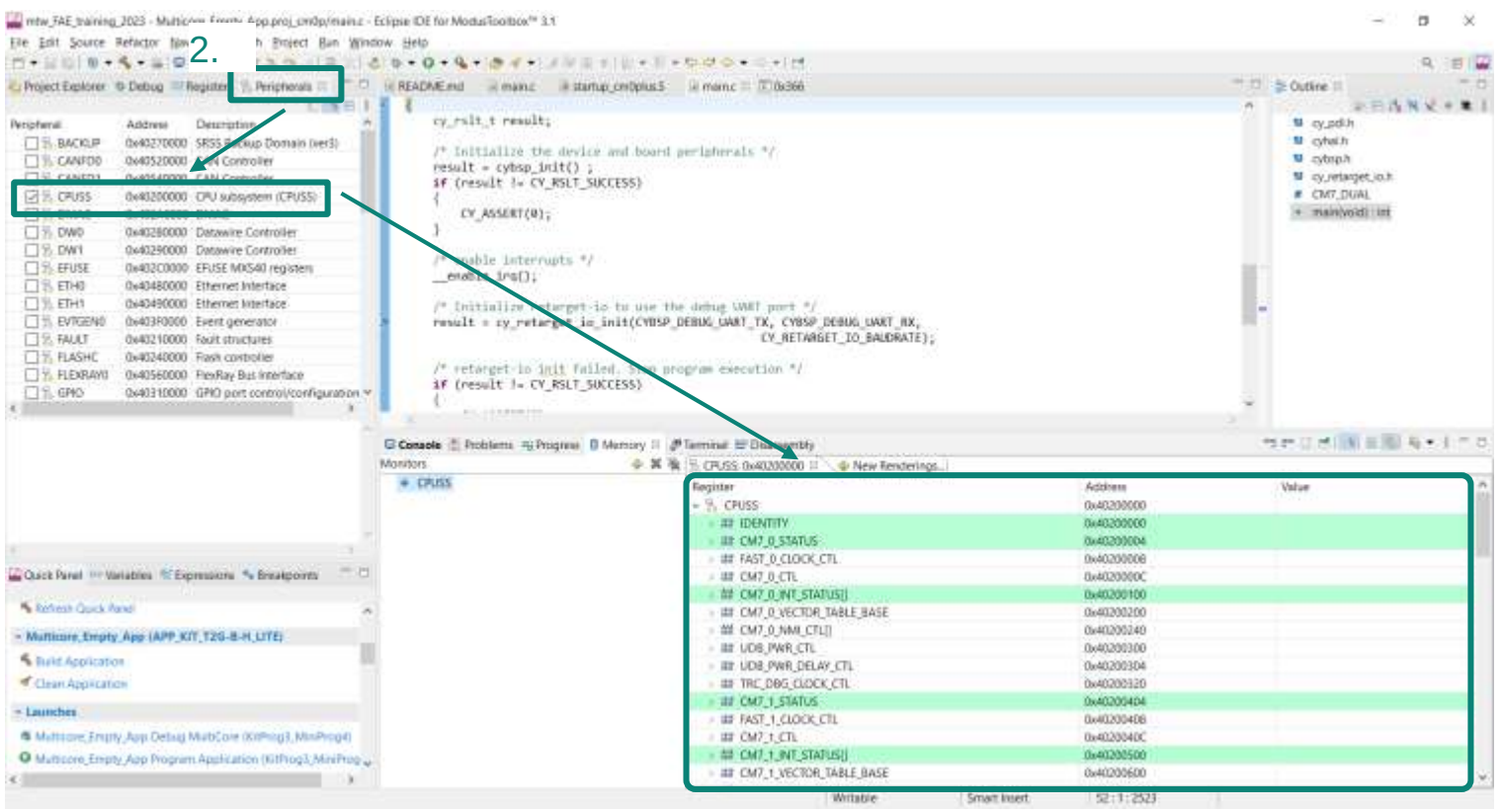


Appendix B - Debugging

– Register view

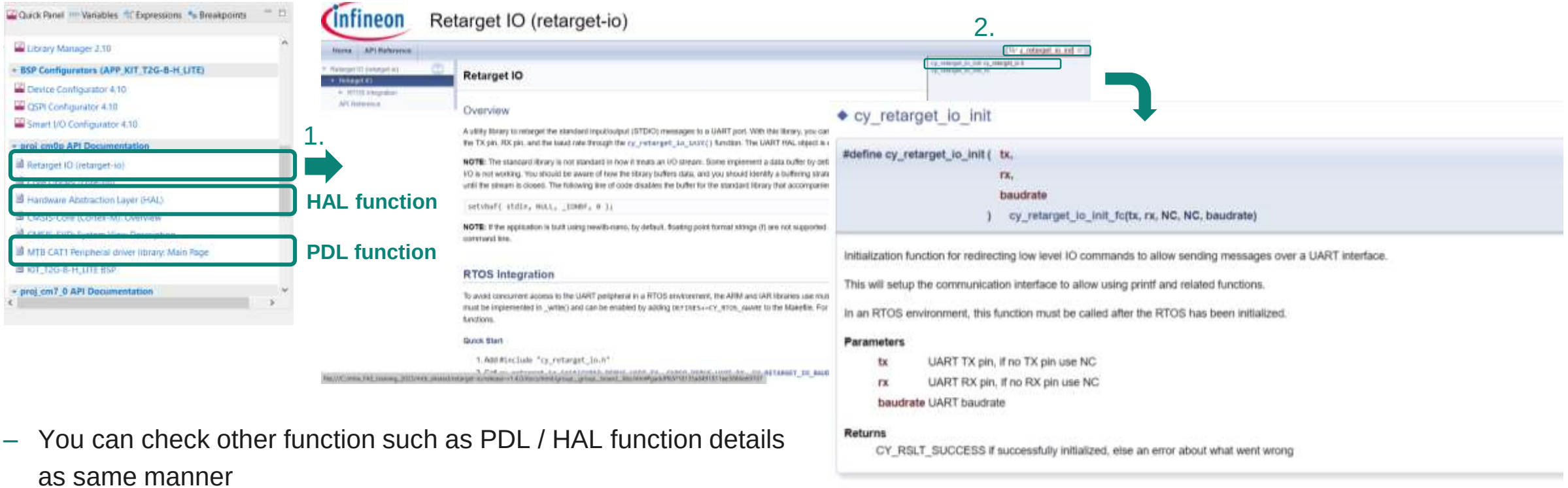
1. Select “**Registers**” tab, you can show CPU specific register
2. Select peripheral what you want to see in “**peripherals**” tab, then open the peripheral register window





Appendix C - Documentation

- Refer the function details
  - You can see the function details from Modus Toolbox
  - If you want to check the "***cy_retarget_io_init()***":
  1. Click the "**Retarget IO (retarget-io)**" in Quick panel
  2. Input "**cy_retarget_io_init**" in search box, and click the function you want to search



  - You can check other function such as PDL / HAL function details as same manner

Appendix D – Additional Information

- **Infineon Websites:**
  - 32-bit TRAVEO™ T2G Arm® Cortex® Microcontroller
  - KIT_T2G-B-H_LITE
- **MyICP:**
  - MyICP KIT_T2G-B-H_LITE
- **Application note:**
  - AN235305: Getting started for ModusToolbox™ in TRAVEO™ T2G family MCUs
- **Modus Toolbox™ usage Training**
  - ModusToolbox™ Software (User Manuals, Getting Started, Application notes etc.)
- **Code examples:**
  - To access additional code examples, please see the GitHub page below.
  - https://github.com/Infineon/TRAVEO_T2G_code_examples
- **Start Guides** for TRAVEO™ T2G Body High Lite Kit:
  - Will be updated soon – stay tuned!


infineon