Getting Started
using IAR Embedded Workbench for Arm
and Renesas RA6M3
# Table of Contents

1. Introduction .......................................................................................................................... 3  
1.1. Information described in this document .............................................................................. 3 
2. Tools and equipment ............................................................................................................... 3  
  2.1. Evaluation boards ............................................................................................................. 3 
  2.2. Renesas Electronics Code Generate tool .......................................................................... 3 
  2.3. IAR Systems Software Development environment .......................................................... 3 
3. Goal of this Getting Started guide ....................................................................................... 3 
4. Set up EK-RA6M3 evaluation board ..................................................................................... 4 
  4.1. Jumper setting .................................................................................................................. 5 
  4.2. Connect USB cable to host computer .............................................................................. 5 
5. Install RA Smart Configurator ............................................................................................... 6  
  5.1. Setup RA Smart Configurator ........................................................................................... 6 
6. Connect RA Smart Configurator with EWARM ................................................................. 10 
  6.1. Configure EWARM Tools option ..................................................................................... 10 
7. Workflow of EWARM using RA Smart Configurator ........................................................... 12 
  7.1. Overview of the workflow ................................................................................................ 12 
8. Create a new project with RA Smart Configurator ............................................................. 12 
  8.1. Create a new empty project with EWARM ...................................................................... 12 
  8.2. Launch RA Smart Configurator ....................................................................................... 14 
  8.3. Configure project configurations in RA Smart Configurator .......................................... 15 
  8.4. Generate files from RA Smart Configurator ................................................................... 21 
  8.5. Import generated files by using EWARM Project connection ....................................... 23 
  8.6. Save Workspace file ........................................................................................................ 26 
  8.7. Change project options of EWARM .............................................................................. 27 
  8.8. Add user code ................................................................................................................... 29 
  8.9. Build and Debug .............................................................................................................. 29 
9. Control LED on the evaluation board .................................................................................. 32 
  9.1. LED control code ............................................................................................................ 33 
  9.2. Access to PFS resister ...................................................................................................... 34 
  9.3. Run LED blinking code .................................................................................................... 34 
  9.4. Pin initialization ............................................................................................................... 35 
10. User timer interrupt for blinking LED ............................................................................... 36 
  10.1. Add Timer stack in RA Smart Configurator ................................................................. 36 
  10.2. Configure parameters of Timer stack .......................................................................... 37 
  10.3. Review the interrupt allocation ...................................................................................... 38 
  10.4. Re-generate the code .................................................................................................... 39 
  10.5. Implement Timer interrupt callback function ............................................................... 40 
  10.6. Write timer starting code .............................................................................................. 41 
  10.7. Debug the timer interrupt callback function, .............................................................. 42 
11. Summary ............................................................................................................................. 43
1. Introduction

The Renesas Electronics RA family is a 32-bit MCU based on the Arm Cortex-M architecture which provides strong embedded security, high performance and ultra-low power operation. This guide will show the steps to be taken in order to develop software with IAR Embedded Workbench for Arm (EWARM) on a RA6M3 evaluation board. The RA Family has high compatibility so the procedure can be reused with other members of the RA family.

1.1. Information described in this document

The information related to product releases and web content are based on their respective states from when this document was published in June, 2020. If there are any updates, please refer to the latest document.

2. Tools and equipment

Evaluation boards and software tools used in this document.

2.1. Evaluation boards

  - R7FA6M3AH3CFC MCU (120MHz, Arm Cortex®-M4, 2MB Flash, 640KB SRAM)
- JTAG connector for the IAR I-jet debug probe
- Supply 5V power via I-jet or (I-jet Trace)

2.2. Renesas Electronics Code Generate tool

We will use RA Smart Configurator 1.1.0. RA Smart Configurator is an initialization code generate tool. It will generate source code files which are needed for the components which users select with using Flexible Software Package (FSP) which is included in RA Smart Configurator. RTOS (FreeRTOS) based project can also be configured although this guide does not cover RTOS based project. RA Smart Configurator can easily connect to EWARM. During development process, you can re-configure components and re-generate the source code. This significantly reduces the development resource usage and simplify the development workflow.

2.3. IAR Systems Software Development environment

- IAR Embedded Workbench for Arm version 8.50.x
- RA family MCUs are supported from version 8.50.1

If you already have product license, please download the latest version directly from the IAR MyPages. The EWARM evaluation version installer can be download from the IAR Systems site (https://iar.com/arm).

After downloading the installer, install the product and activate the license following with the wizards.

3. Goal of this Getting Started guide

The goal is to generate code with RA Smart Configurator and debug the software on the EK-RA6M3 with the EWARM. There are user LEDs mounted on EK-RA6M3. We are going to create the LED blinking software by using GPIO pin control and a timer interrupt.
4. Set up EK-RA6M3 evaluation board
The EK-RA6M3 evaluation kit includes the evaluation board and a micro USB cable.

The user LEDs are mounted on the left-hand side of the evaluation board.
4.1. Jumper setting
Make sure that J15 is fit in the BOOT MODE so it is set as “INTERNAL FLASH”, as shown below:

4.2. Connect USB cable to host computer
Power to the EK-RA6M3 can be supplied directly from the IAR I-jet (or I-jet Trace) debug probe. Connect the J20 MIPI20 header connector to the I-jet or I-jet Trace and then connect the I-jet to a USB port of the host computer. The J20 header supports JTAG, SWD and ETM (TRACE).
5. Install RA Smart Configurator

Download RA Smart Configurator from Renesas Electronics GitHub site. Version 1.1.0 is the latest version in June, 2020.

https://github.com/renesas/fsp/releases

Download the installer named "setup_fsp_v1_1_0_rasc_v2020-04.exe" as displayed above. Make sure that you are NOT downloading the "setup_fsp_v1_1_0_e2s_v7_8_0.exe".

5.1. Setup RA Smart Configurator

Execute the "setup_fsp_v1_1_0_rasc_v2020-04.exe" installer.

The Install Wizard will be launched. Install using the default configuration. Depending on the host computer current state, the installer might require permission to install additional support software such as the Visual Studio runtime libraries.
Click on Next.

Read the Software Agreements, check the “I accept” box and then click Next.
Click on Install.

Wait while installation is performed.
To finish the installation, click **OK**.
If in this step you are required to install additional software, install them using their default settings.

After the installation, you can find RA Smart Configurator (rasc.exe) under C:\Renesas\RA\sc_v2020-04_fsp_v1.1.0\eclipse.

Although, it is not possible to launch rasc.exe directly. Being so, let’s configure the EWARM to invoke it.
6. Connect RA Smart Configurator with EWARM
Configure EWARM tool menu to enable to launch RA Smart Configurator from EWARM menu.

6.1. Configure EWARM Tools option
Launch the IAR Embedded Workbench for Arm 8.50.x from the Windows Start menu shortcut.

Select **Tools > Configure Tools** from the menu bar.

Click on **New**
Set all the parameters as shown below:

- **Menu text:** RA Smart Configurator
- **Command:** C:\Renesas\RA\sc_v2020-04_fsp_v1.1.0\eclipse\rasc.exe
  - If you have installed a different version, specify the actual path of installed rasc.exe
- **Argument:** --compiler IAR configuration.xml
  - Specifies RA Smart Configurator to generate source code files for EWARM
- **Initial Directory:** $PROJ_DIR$
  - Specifies RA Smart Configurator to initiate with the EWARM project directory.
- **Tool Available:** Always

After performing the configuration, click **OK**.
Now launch **Tools > RA Smart Configurator**.
7. Workflow of EWARM using RA Smart Configurator
Here is the workflow of the development with IAR Embedded Workbench for ARM using RA Smart Configurator

7.1. Overview of the workflow
- Create a new empty project with EWARM.
- Launch RA Smart Configurator.
- Configure project settings in RA Smart Configurator.
- Generate source code files from RA Smart Configurator.
- Import generated files with EWARM project connection.
- Change project options if needed.
- Write user code.
- Build and debug the project.

When you change the device configuration:
- Launch RA Smart Configurator.
- Change the project configuration in RA Smart Configurator.
- Generate source code files from RA Smart Configurator.

Then files in the EWARM project will be updated. You can repeat the process as many times as needed to complete your application project.

8. Create a new project with RA Smart Configurator
Create a new project by following the workflow described above.

8.1. Create a new empty project with EWARM
Select [Create New Project] from [Project] menu.
Select **Empty Project** then click **OK**.

![Image of Create New Project dialog box]

Select Empty Project then click OK.

Save the project to any directory. Here we will save as **MyProject** under **C:\MyWorkspace** directory.

Save the project to any directory. Here we will save as MyProject under C:\MyWorkspace directory.

![Image of Save As dialog box]

Save the project to any directory. Here we will save as **MyProject** under **C:\MyWorkspace** directory.

The new project is created.
Now we can close the IAR Information Center for Arm tab.

8.2. Launch RA Smart Configurator
Launch the RA Smart Configurator from Tools menu.

The Renesas Smart Configurator splash screen will show up.
Initial wizard will be displayed.

If RA Smart Configurator is not launched, double-check the parameters in Tools > Configure Tools as described in section 6.1.

8.3. Configure project configurations in RA Smart Configurator
First of all, we will select the board to be used. When the evaluation board is selected, then corresponding device will be automatically selected.

On the Board field, select EK-RA6M3.
R7FA6M3AH3CFC is automatically selected as Device. We will not use RTOS. Leave RTOS as No RTOS then click Next.

We are going to manually create a LED blinking application. For that, select Bare Metal - Minimal as project template and then click Finish.
The main window of the RA Smart Configurator will be displayed.

The RA Smart Configuration main window is divided as follows:

- **Upper-Left corner**: Summary and settings.
- **Upper-Right corner**: Pin assigns of selected device.
- **Lower-Left corner**: Provides detailed information of a selected item.
- **Lower-right corner**: Error information about any conflicts.

**Summary**

Overview of the project such as board information, FSP versions, and so on.
BSP
You can see the detailed board information in Property Window

Clocks
You can check and change the clock configuration of systems. Clocks need to be configured and supplied to each component, you can configure them with the GUI.

In this example window, we can see that clock which comes from **24 MHz XTAL** is multiplied by PLL then supplied to components. ICLK is the CPU internal clock.
Pins
Here you can check and change the MCU's pin assignments.

We will control user LED1 which is connected to P403.
If we the pin is selected, we can see it comes pre-configured as GPIO in output mode.
On the Package view, light green pins indicate pins which have configured in RA Smart Configurator.
Those configurations will be saved in a file named g_bsp_pin_cfg containing the configuration for initialization code.

Interrupts Configuration
Here will be a list of all the peripheral interrupt which is configured in the RA Smart Configurator. It is also possible to create User Events and assign them to interrupts.
**Event Links**
RA family offers a special feature named ELC (Event Link Controller). Events generated by peripheral modules can be linked as signals to trigger other modules. We will not use the feature for this guide.

---

**Stacks**
By using RA Smart Configurator, you can place software modules as Stacks. This is a unique concept of FSP and RA Smart Configurator. Some stack may have parent-child relations. If stacks which is necessary for other stack is not placed, the error will be displayed. There are several kinds of stack from peripheral drivers to large size middle-wares.

As per default, the `g_ioprot` stack comes enabled in the project. It is used to control the port pins.
Components
You can check the activated components for the projects.

![Components Configuration](image)

We can see that the minimal components for the EK-RA6M3 are already selected. We will start from generating the source files as is.

8.4. Generate files from RA Smart Configurator
After finish configuring in RA Smart Configurator, we will generate the source files. Click **Generate Project Content**.

![Pins Configuration](image)
A progress bar will pop-up and lasts as long as it takes to the project files to be generated.

![Generate Project Content window]

On the Windows Explorer, if we navigate to the EWARM project directory, we will see many new folders and files which were automatically generated.

![Windows Explorer screenshot]

These files are updated every time the **Generate Project Content** button is clicked on the RA Smart Configurator.
8.5. Import generated files by using EWARM Project connection
We will use project connection mechanism to link and import generated files by RA Smart Configurator.

Back to EWARM window, then select Project > Add project Connection.

On the dialog window, connect using IAR Project Connection and click OK.

Select buildinfo.ipcf file and click Open.
The Flex Software group is added to the project.

If you expand Flex Software group, you can see the generated files which populates the EWARM project.

The Build Configuration group typically contains the code related to the board’s specific configuration. The Components group contains the HAL Drivers and others that are used without user modifications. The Generated Data group comes with the main.c module from where the program starts. The Program Entry group comes with the hal_entry.c. The file contains the hal_entry() function, from where the user code should start.
Below you can see how the `hal_entry()` function is called from the `main()` function.

There is a comment on the top of the `main.c` file, which can be found is under the **Generated Data** group. The comment clearly states that this file should not be edited.

Many other files bring this kind of comment. These files should not be touched by the user because they might have to be overwritten by the RA Smart Configurator.

It is possible to show the line numbers in the Code Editor by changing the **Tools > Options** from the menu bar.

Check the **Show line numbers** checkbox in the **Editor** category, as below:
8.6. Save Workspace file
Workspace it the top level and projects belong to a workspace in EWARM project structure. By now, we have created a project, but we have not saved workspace. Save the workspace now.

Select [File] > [Save Workspace]

You can select any names, here we will use “MyWorkspace” then save it by clicking Save.

Now we can see workspace name we have chosen displaying on the EWARM title bar.
8.7. Change project options of EWARM
Next, we will check and change the project options.
On the Workspace, right-click on **MyProject** icon then select **Options** from the context menu.

We can see that **R7FA6M3AH** is already selected as a target device. When we added project connection, the most essential and obvious settings come pre-configured.
C/C++ Compiler > Optimizations has its level set as Low for Debug by default. This is the typical setting for a better debugging experience as stepping throughout the generated code will be more related to the original C code.

If you require higher application performance or want to reduce the code size, select a higher optimization level. On highest optimization levels, the code can be transformed and moved drastically in many cases, so stepping throughout it still possible but it might be harder to keep track of the data in the variables as those are kept in registers.

In the Preprocessor tab, you can see the include directories which are used in this project.

You can see that the generated code by RA Smart Configurator is well structured.
Now we need to change the debugger setting.

![Options for node "MyProject"

Change **Debugger > Setup > Driver** from **Simulator** to **I-Jet**.

Click **OK** to finish Option settings.

8.8. Add user code
As we have seen before, application entry function, *hal_entry()* is empty. So add the following snippet of code to begin with:

```c
void hal_entry(void)
{
    static int count;
    while(1)
    {
        for (volatile uint32_t dly = 10000000; dly; dly--){};
        count++;
    }
}
```

This is a simple code which increments a variable count with waiting by software delay within while(1) loop.

8.9. Build and Debug
Click on **Make** button in the tool bar to build the project or press F7.
If everything was done correctly, in the end, we should see the total number of errors equals to 0, which means you successfully built the project.

Before moving forward, make sure that:
1. I-jet is connected to the Computer with an USB cable
2. J20 header is connected to the I-jet debug probe
3. J15 is jumped on the 1-2 position
4. J10 USB port is connected to the Computer with an USB cable
5. J9 is jumped on the 1-2 position
6. J8 is jumped on the 1-2 position

Now it is time to download the application image to the flash memory of the RA6M3 MCU and then start debugging. Click on Download and Debug (CTRL + D).
You can see that the program is stopped at the beginning of main function. You can control debug operations with the buttons on the tool bar.

Toggle Breakpoints
- Make & Restart Debugger
- Restart Debugger
- Stop Debugging
- Go
- Break
- Reset
- Run to Cursor
- Step Out
- Step into
- Step Over

Click [Go],

Then click [Break]
You can see that the CPU was running in the software delay of `hal_entry` function.

![Image of C code snippet]

To check the variable value, right-click on `count` variable then select [add to watch].

![Image of watch window with `count` variable]

Watch window is displayed and, once the execution is paused, it is possible to check the current value.

![Image of watch window with `count` variable value]

You can also add variables by manually in the watch windows.

We have seen basic build and debug process. Click [Stop Debugging] to close the debug session.

![Image of debugging interface with stop debugging button]

9. **Control LED on the evaluation board**

Next, we will control the LED on the board form user code. Peripheral resisters including GPIO is memory mapped, so you can control it by accessing the corresponding address, but it is not effective, so we will use API provided FSP.
9.1. LED control code

Here is the actual LED blinking code:

```c
void hal_entry(void)
{
    static int count;

    bsp_io_level_t pin_level = BSP_IO_LEVEL_LOW;
    uint32_t pin = BSP_IO_PORT_04_PIN_03;

    while(1)
    {
        /* Access to PFS register */
        R_BSP_PinAccessEnable();

        /* Write to this pin */
        R_BSP_PinWrite((bsp_io_port_pin_t) pin, pin_level);

        /* Protect PFS registers */
        R_BSP_PinAccessDisable();

        /* Toggle level for next write */
        if (BSP_IO_LEVEL_LOW == pin_level)
        {
            pin_level = BSP_IO_LEVEL_HIGH;
        }
        else
        {
            pin_level = BSP_IO_LEVEL_LOW;
        }

        for(volatile uint32_t dly=10000000;dly;dly--){};
        count++;
    }
}
```

The red lines are added to control GPIO.

R_BSP_PinWrite() function actually toggles LED (GPIO). Definitions used in this code are described in Components > bsp_io.h

R_BSP_PinWrite() receives the pin number and level (high or low) as function parameters.

BSP_IO_LEVEL_LOW, BSP_IO_LEVEL_HIGH, BSP_IO_PORT_01_PIN_12 are defined in bsp_io.h as an enumerated type. BSP_IO_PORT_01_PIN_12 indicates P1 12 pin which is connected to LED.
9.2. Access to PFS resister

R_BSP_PinAccessEnable() and R_BSP_PinAccessDisable() are called before and after R_BSP_PinWrite() function call.

PFS stands for Pin Function Select module. When you configure pins, to avoid unexpected or conflicting settings, you need to set Write Enable.

R_BSP_PinAccessEnable() ensures exclusive access and set to write-enable for pin configurations.

R_BSP_PinAccessDisable() does the opposite. It disables PFS resister writing.

This mechanism offers safe pin configurations.

9.3. Run LED blinking code

Make (F7) the project again.
Then, **Download and Debug (CTRL + D).**

You will see the LED on the board blinking every 1 second.

Try commenting out the `R_BSP_PinAccessEnable()` and `R_BSP_PinAccessDisable()` functions. Then rebuild the project and watch what happens to the LED. It should not blink anymore.

**9.4. Pin initialization**

Normally, when GPIO pins are controlled, pin initialization code is required. However, there are no initialization code in `hal_entry()` function nor on the `main()` function. The Initialization is performed in `R_BSP_WarmStart()` function in the `hal_entry.c` file before the `main()` function is called.

![C Code Example]

Not only LED control pins but also all other pins configured in RA Smart Configurator are initialized from `R_BSP_WarmStart()`. If you want to see this process in details, set a code breakpoint in the `R_BSP_WarmStart()` function as above, you just click on the left side of the line number and a red circle will appear.
Then use the **Step Into (F11)** command in the **Debug Toolbar**.

---

**10. User timer interrupt for blinking LED**

The previous code uses software delay for blinking LED. FSP also provides software delay API.

```c
const bsp_delay_units_t bsp_delay_units = BSP_DELAY_UNITS_MILLISECONDS;
/* Set the blink frequency must be <= bsp_delay_units */
const uint32_t freq_in_hz = 2;
/* Calculate the delay in terms of bsp_delay_units */
const uint32_t delay = bsp_delay_units / freq_in_hz;
/* Delay */
R_BSP_SoftwareDelay(delay, bsp_delay_units);
```

By using `R_BSP_SoftwareDelay()`, you can execute precious interval software delay. However, software delay has some demerits, so using timer interrupt is recommended to execute periodical process. This time, we will use General Purpose Timer interrupts of RA family to implement 100 ms LED blinking.

---

**10.1. Add Timer stack in RA Smart Configurator**

Once again, activate RA Smart Configurator window. If RA Smart Configurator is not already open, launch it from **Tools > RA Smart Configuration**.

Click on the **Stacks** tab.
Now select New Stack > Driver > Timers > Timer Driver on r_gpt.

Timer Driver is added to the project. Parameters are displayed in the property window.

10.2. Configure parameters of Timer stack
Change the parameter to generate a timer interrupt every 100 ms.
Change the parameter as below:

This time, we will use repeat timer, so leave Mode as Periodic. As a default period unit is Cycle count, but it is not easy to understand, so we will change Period Unit to Milliseconds and Period to 100.

With that, even if we change the CPU clock, the LED will keep blinking with a period of 100 ms.

Finally we set g_timer0_callback to Interrupts > Callback. The function name can be any. When RA Smart Configurator and FSP are used, we will not write the actual process in the interrupt handler directly but we will register callback function then write process code in the callback function.

We will implement the g_timer0_callback() function. The General Purpose Timer interrupt generates an overflow. We will set the priority for the overflow interrupt. Here we will set as Priority 1 which will be the second highest priority in our project.

10.3. Review the interrupt allocation
See in Interrupts tab to check the timer interrupt is allocated or not.

Now we see gpt_counter_overflow_isr() is assigned to GPT0_COUNTER_OVERFLOW event.

The RA Smart Configurator generates interrupt handlers suffixed with _isr. These handlers call their respective callback functions.
10.4. Re-generate the code

Make sure the EWARM is not in debugging then click Generate Project Content.

When you activate EWARM, then you will see some low level driver modules such as r_gpt and r_timer_api are added to the project.

It is recommended to use the API wrapped into r_timer_api for better maintainability. The hal_entry.c file won’t be overwritten even if the RA Smart Configurator re-generate files.

So you can continue application development. Again, please note that the files under Generated Data group can be overwritten, so we should not edit the files.
10.5. Implement Timer interrupt callback function
Implements g_timer0_callback() function which we specified in RA Smart Configurator.

We can use the Find in Files feature to investigate the definition of functions. On the IDE main menu, proceed to Edit > Find and Replace > Find in Files (CTRL + SHIFT + F).

All the code which contain the keyword in the project are listed.
Some interrupts may use `p_args` to get parameters, but this time, we do not use it, so we only check if it is NULL or not.

```c
uint32_t g_timer_overflow_flg = 0;
void g_timer0_callback (timer_callback_args_t * p_args)
{
    if (NULL != p_args)
    {
        g_timer_overflow_flg = 1;
    }
}
```

We will also add a flag variable to indicate if timer interrupt occurs. The callback function set the value 1.

### 10.6. Write timer starting code

Timer module does not start automatically, so user code is required to start it. Here are the code includes some error check.

```c
fsp_err_t err = FSP_SUCCESS;

//err = R_GPT_Open (&g_timer0_ctrl, &g_timer0_cfg);
err = g_timer0.p_api->open(&g_timer0_ctrl, &g_timer0_cfg);
if (FSP_SUCCESS != err)
{
    while(1);
}

//err = R_GPT_Start(&g_timer0_ctrl);
err = g_timer0.p_api->start(&g_timer0_ctrl);
if (FSP_SUCCESS != err)
{
    while(1);
}
```

Write this code in `hal_entry` function before while(1) loop. Notice that the `R_GPT_xxx` functions also does work.

Here is the `hal_entry` function which toggles LED with using timer interrupt flag.

```c
void hal_entry(void)
{
    static int count;

    bsp_io_level_t pin_level = BSP_IO_LEVEL_LOW;
    uint32_t pin = BSP_IO_PORT_01_PIN_12;

   fsp_err_t err = FSP_SUCCESS;
    err = g_timer0.p_api->open(&g_timer0_ctrl, &g_timer0_cfg);
    if (FSP_SUCCESS != err)
    {
        while(1);
    }

    err = g_timer0.p_api->start(&g_timer0_ctrl);

    if (FSP_SUCCESS != err)
    {
        while(1);
    }
}
```
R_BSP_PinAccessEnable();
R_BSP_PinWrite((bsp_io_port_pin_t) pin, pin_level);
R_BSP_PinAccessDisable();

if (BSP_IO_LEVEL_LOW == pin_level)
{
    pin_level = BSP_IO_LEVEL_HIGH;
}
else
{
    pin_level = BSP_IO_LEVEL_LOW;
}

while (g_timer_overflow_flg == 0);
g_timer_overflow_flg = 0;
count++;}

When the code is executed, the LED1 will blink exactly at 100 ms period.

10.7. Debug the timer interrupt callback function,
Let's see how the callback function is called with debugger.
Set a breakpoint in the callback function.

When the interrupt happens and stopped at breakpoint, select View > Call Stack.

The Call Stack window shows a list of function calls, with the current function at the top. In the above picture, the g_timer0_callback() function was called from interrupt handler gpt_counter_overflow_isr. The interrupt occurred while executing hal_entry().
11. Summary
In this document, we have seen how it is easy to develop with the IAR Embedded Workbench for ARM using the RA Smart Configurator and the EK-RA6M3 evaluation board.

The Renesas RA family offers so many other powerful features. By using the RA Smart Configurator GUI, you can very quickly configure the desired settings and enable them by using the FSP API function calls.

Today we have seen how to control a GPIO port and a timer interrupt.

Many other modules can be configured and used by applying the same principles you have seen in this workflow.