

BASICS OF THE RENESAS SYNERGY™ PLATFORM

Richard Oed



CHAPTER 8

HELLO WORLD! – HELLO BLINKY!

CONTENTS

8 HELLO WORLD! – HELLO BLINKY!	03
8.1 Your First Project Using e ² studio	04
8.1.1 Creating a Project with the Project Configurator	04
8.1.2 Setting Up the Runtime Environment with the Synergy Configurator	06
8.1.3 Writing the First Lines of Code	07
8.1.4 Compiling the First Project	09
8.1.5 Downloading and Debugging the First Project	10
8.2 Your First Project Using IAR Embedded Workbench [®] for Renesas Synergy [™]	12
8.2.1 Creating a New Project	12
8.2.2 Setting Up the Runtime Environment with the Synergy Configurator	14
8.2.3 Writing the First Lines of Code	15
8.2.4 Compiling the First Project	17
8.2.5 Downloading and Debugging the First Project	17
Disclaimer	19

8 HELLO WORLD! – HELLO BLINKY!

What you will learn in this chapter:

- How to create a project for the Renesas Synergy™ Promotion Kit PK-S5D9 from scratch.
- How to change settings for the Synergy Software Package (SSP) in the Synergy Configurator.
- Write code to toggle the LEDs on the PK.
- How to download and test code

The very first program most newcomers to a programming language wrote (and still write) is the one, which simply puts the string “Hello World” to the standard output device. For me, it was typing “Writeln (‘Hello World’)” into the editor, as I started with Pascal. Ever since then, I wrote similar lines in several other languages, mostly as a sanity check for the installation of a new development environment.

When I moved on to program embedded systems in the late 1980’s, there was no screen where the string could be sent to. So how to instruct the processor to give signs of life? LEDs were barely found in these systems, so toggling one of the very few I/O-pins and observing the waveform with an oscilloscope was the way to go. Over the years, LEDs became a commodity item and we placed plenty of them on our boards, using their blinking as the new “Hello World”.

And this is also the objective for this chapter: Toggle an LED on a Promotion Kit (PK), using everything you learned in the chapters before: You will write the code (nearly) from scratch, create a new project using the configurators, employ the APIs of the Synergy Software Package (SSP) and finally download, debug and run the code. This exercise brings everything together.

As a prerequisite, e² studio or the IAR Embedded Workbench® for Renesas Synergy™ (IAR EW for Synergy), as well as the SSP (and the Synergy Standalone Configurator (SSC) if IAR EW for Synergy is used), should be installed on your Windows® workstation (see [chapter 4](#) for details) and you should have verified that your setup is working (as described in [chapter 7](#)). And those of you doing the previous hands-on exercises: Please bear with the author, as he decided to cover again some of the topics already discussed for the sake of those of you moving directly from the foreword to this chapter. And if you do not want to do all the coding on your own: The complete project is available from the book’s website (<https://www.renesas.com/synergy-book>) for both development environments.

For this exercise, we will again use the PK-S5D9, which is extremely well suited for tasks like this, as it allows you to explore the Synergy microcontroller and all its peripherals instantly. External hardware can be easily attached, as more than 80% of the pins are accessible through connectors. And with the Synergy S5D9 Group MCU being the superset device of the S5 Series of Synergy MCUs, every feature of this Series can be evaluated and the results later on applied to the smaller siblings of the Synergy MCU Family as well. Figure 8-1 shows the block diagram of the board, highlighting the main features of it.

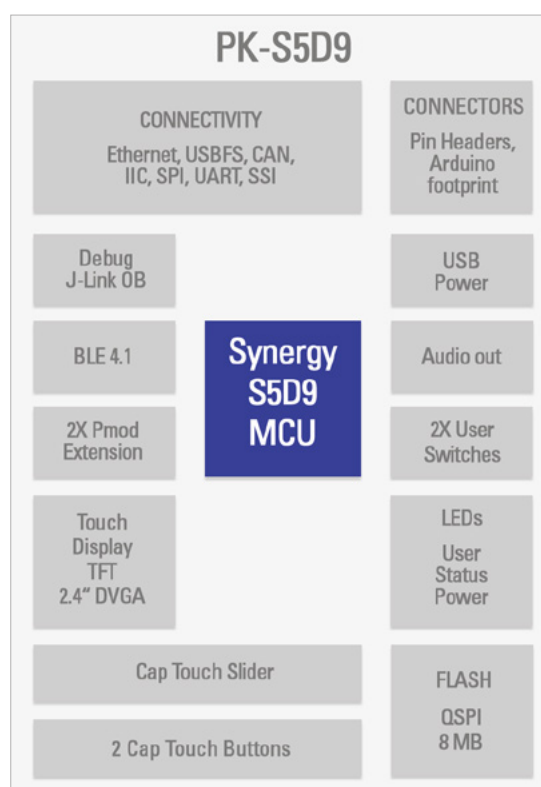


Figure 8-1: Block diagram of the PK-S5D9 Promotion Kit

8.1 Your First Project Using e² studio

8.1.1 Creating a Project with the Project Configurator

If not already done, start e² studio from your Windows[®] workstation Start Menu. Once the ISDE is up and running, dismiss the Welcome screen, if it shows, as it would block other windows from viewing.

Writing a new program for a microcontroller in e² studio always requires to create new project first, so this is the first step you need to take.

For this, go either to *File* → *New* → *Synergy C/C++ Project*, or right-click in the Project Explorer and select *New* → *Synergy C/C++ Project*. Both ways will bring up a dialog asking for the template to be used. Select *Renesas Synergy C Executable Project* and click on *Next*.

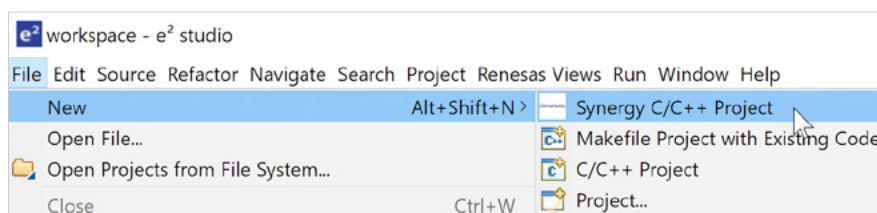


Figure 8-2: First step is to call the Project Configurator

Once the *Project Configuration* window shows, give the project a name, for example *MyBlinkyProject* and check if the *GCC ARM[®] Embedded* is highlighted under *Toolchains*. Next, verify that a license for the SSP installed. If the field with the *License Details* is empty, click on *Change license file* and the dialog window appearing will guide you to the directory where the evaluation license was placed during installation. If you installed e² studio and the Synergy Software Package with the Platform Installer, the path will be *C:\Renesas\Synergy\e2studio_v7.5.1_ssp_v1.7.0\internal\projectgen\licenses*. If you used the Standalone installers, you will find the license file in the *C:\Renesas\e2_studio\internal\projectgen\arm\licenses* directory. Select the file (there should only be this one) named *SSP_License_Example_EvalLicense_20180725.xml* (or similar). If you already requested and received a development and production license either from Renesas or from the Super User of your company, point the file selector to the correct path for that one. Once loaded, you can examine the permissions for the different components of the SSP inside the *License Details* frame. With this done, click on the *Next* button to move on to the *Board Selection* screen.

Under *Device Selection* look out for the field called *SSP version*: It should show the same version of the Synergy Software Package you downloaded before. Under *Board*, select *S5D9 PK*, as this is the hardware we want to use for our small »Hello World« program. Verify that *R7S5D97E3A01CFC* is shown beside *Device*, it should have been automatically selected. If not, navigate through the drop down list until you found it. In the *Select Tools* frame, verify that *Toolchain*, *Toolchain version* and *Debugger* read *GCC ARM[®] Embedded*, *7.2.1.20170904* and *J-Link ARM[®]*. These fields should be pre-populated for you. If not, modify them to match the values given above.

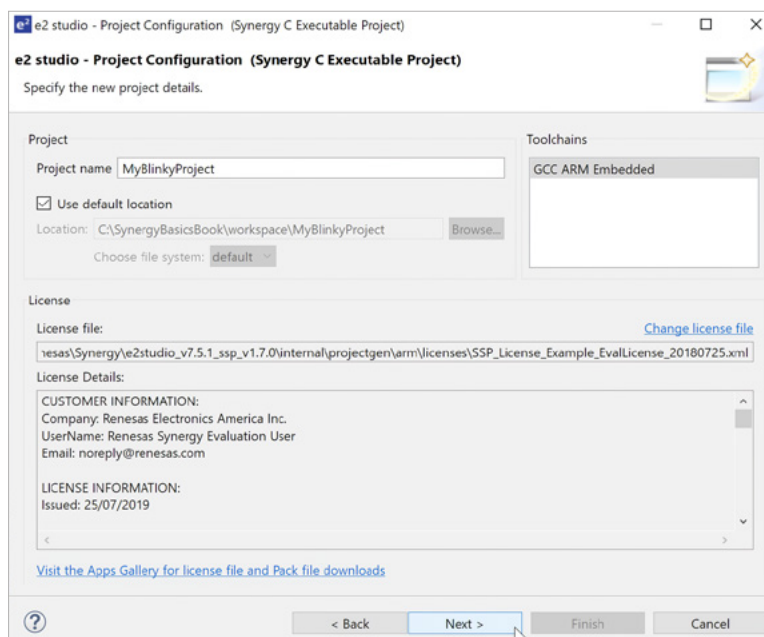


Figure 8-3: The first screen of the Project Configurator mainly asks for the project's name

With this done, click on *Next* to open the *Project Template Selection* screen. A project template may include several items, at least it includes the correct Board Support Package for the selected board / device combination. Some templates even include a complete example project. In our case, select the *BSP* entry, which will load the Board Support Package for the Promotion Kit. Click on *Finish*.

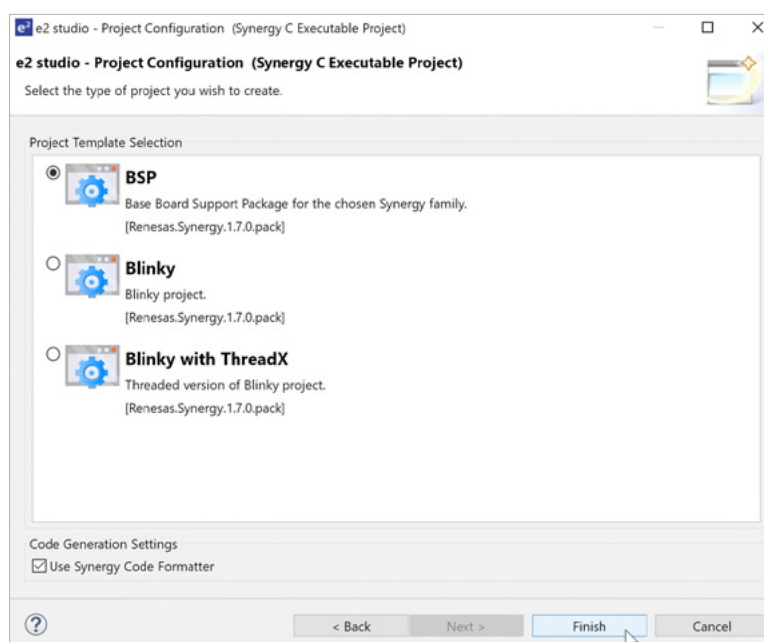


Figure 8-4: Selecting the BSP entry will load the correct Board Support Package

The *Project Configurator* will close and will create all the necessary files for the project in a last step. Once this post-processing is complete, you will be asked if you want to open the *Synergy Configuration* perspective. Select *Yes*.

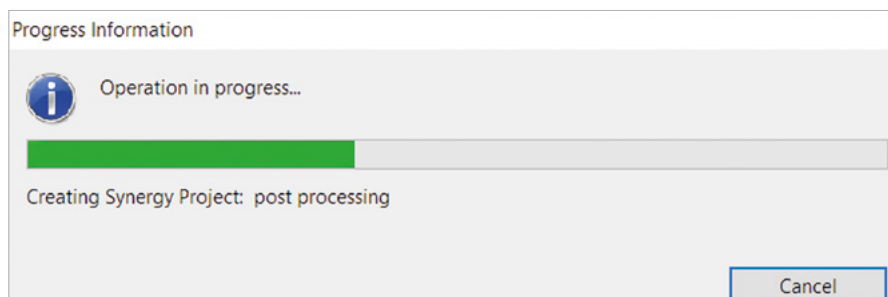


Figure 8-5: The post-processing step of the Project Configurator will create the settings and files for our project

8.1.2 Setting Up the Runtime Environment with the Synergy Configurator

Once the Synergy Configurator has started, it presents you a summary of the project and a short overview over the Synergy microcontroller selected. It also provides a convenient shortcut to the Renesas Synergy Support page on [Renesas.com](https://www.renesas.com), where you can access the documentation, the Knowledge Base and the Renesas Rulz Forum.

The following tab, called *BSP*, allows you to view and edit aspects of the board setup, while in the *Clocks* tab, the initial clock configuration for the project can be set. Any potential problem will be highlighted in red and hovering with the mouse over a highlight will display an explanation. The fourth tab in the configurator, called *Pins* is covering the initial pin setup of the Synergy device in the project, including the logic level they should have at startup. Pins can be listed either based on ports or peripherals. A package view, again with error marking, is available just at the right hand side of the configurator, reducing possible mistakes to a minimum.

The *Threads* tab, allows to add and configure different components. Since we do not use an RTOS in this project, there is only a *HAL/Common* entry, showing the necessary basic modules. In the *Messaging* tab following, you can configure the Messaging Framework for ThreadX® based projects. The final tab *Components* displays the elements available in the SSP and which of them are currently included in the project. It also allows to switch between different versions of the same module, in case you have multiple versions of the SSP installed. Modifications like adding or removing components shouldn't be made here, but preferably in the *Threads* tab, as they can be configured there as well.

For our project there is no need to change anything in this configurator, as all necessary settings have been done for us already by the *Project Configurator*. As the final step in the Synergy Configuration, additional source code based on the current configuration needs to be created. Click on the *Generate Project Content* button at the top right hand side of the Synergy configurator. With that, the required files will be extracted from the SSP, adjusted to the settings made in the configurator and added to the project.

8.1.3 Writing the First Lines of Code

With all the automatically generated files being now in place, it is time to have a look on what was created. The *Project Explorer* at the left hand side of the ISDE lists everything currently included. The *src* folder contains a subfolder called *synergy_gen*, holding configuration sets such as channel number etc. The *src* folder also includes a file named *hal_entry.c*. This is the one you will edit later on. Please note that while there is a file called *main.c* in the *synergy_gen* folder, your user code must go to *hal_entry.c*. Otherwise you will loose your changes, if you make modifications in the Synergy Configurator and re-created the project contents, as this file will be overwritten each time you click on *Generate Project Content*.

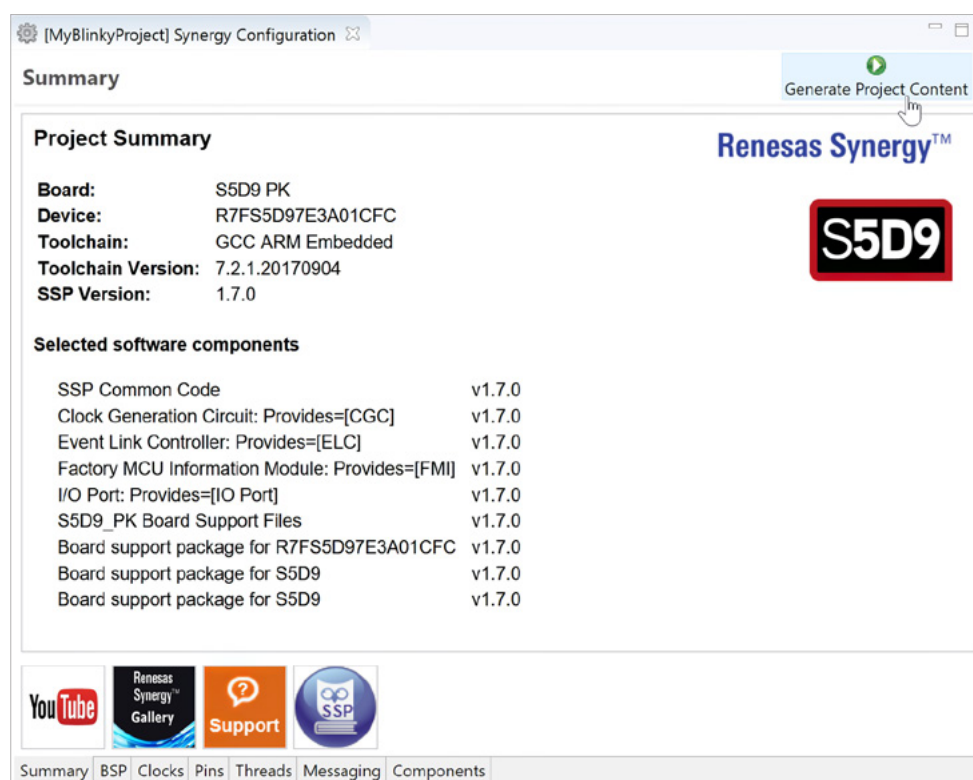


Figure 8-6: The summary tab of the Synergy Configurator

The project also contains several directories with »synergy« in the name, containing source-, include- and configuration files for the SSP. It is a general rule that the contents of these folders (and subfolders) should not be modified. They contain files generated by the configurator and any modification made there will be lost the next time the project content is generated or refreshed. The user editable source files are those directly in the root of the `\src` folder or any other folder added by you.

Now it is time for you to write the first real Synergy Platform source code. The plan is to alternate between the green LED1 and the orange LED3 on the S5D9 Promotion Kit every second, so you will have to add code for turning them on and off and for a delay loop. How to do that?

There are actually two options for that: One is using the API of the Hardware Abstraction Layer directly and one is using the HAL together with the BSP. Which one do you think is the better one? You can review [chapter 2](#), if you are unsure about the answer.

Looking at the code in the file `\src\synergy_gen\common_data.c`, we find that there is the following definition for the I/O port driver instance `g_ioport`:

```
const ioport_instance_t g_ioport =
{ .p_api = &g_ioport_on_ioport, .p_cfg = NULL };
```

`g_ioport_on_ioport` is a structure, which declares the possible actions for the ports and is assigned to the API-pointer of the `g_ioport` instance. The contents of the structure can easily be viewed by hovering with the mouse over it, which will reveal that one of its members, `.pinWrite`, is a pointer to the pin write function.

So turning an LED on, you could write:

```
g_ioport.p_api->pinWrite(ioport_port_pin_t pin,
                        IOPORT_LEVEL_LOW);
```

But this means that you would actually need to know which I/O-ports LED1 and LED3 are connected to and how many LEDs are available for use! For that, we could either read the documentation of the board or scrutinize the schematics to find the correct port. Or simply use the BSP API, which provides the structure `bsp_leds_t` for that purpose. Calling the BSP function:

```
R_BSP_LedsGet(bsp_leds_t * p_leds);
```

will then populate a user defined variable of the type `bsp_leds_t` (e.g. `Leds`) with the necessary values.

This means that you can turn on LED1 by writing:

```
g_ioport.p_api->pinWrite(Leds.p_leds[BSP_LED_LED1],
                        IOPORT_LEVEL_LOW);
```

This statement needs to be followed by a second one to turn LED3 off by setting its pin-level to high.

Finally, you need to provide a delay to have the LEDs toggle in a user friendly way. For that, again call a BSP_API:

```
R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_SECONDS);
```

This will create a one second delay loop. The macro `BSP_DELAY_UNITS_SECONDS` defines the units for the delay, either in seconds, milliseconds or microseconds, while the 1 stands for the number of units to delay. More information about each function call or variable in the code can be viewed by using the Smart Manual feature of e²studio.

All what is needed once this is done, is to copy/paste the three lines of code and to reverse the pin-levels of the LEDs in the second set. And finally, as we want to run the program indefinitely, a `while(1)`-loop needs to be created around the code.

What is left now, is to insert the following lines of code into the *hal_entry.c* file:

```
void hal_entry(void)
{
    bsp_leds_t Leds;
    R_BSP_LedsGet(&Leds);

    while (1)
    {
        g_ioport.p_api->pinWrite(Leds.p_leds[BSP_LED_LED1],
                                IOPORT_LEVEL_HIGH);
        g_ioport.p_api->pinWrite(Leds.p_leds[BSP_LED_LED3],
                                IOPORT_LEVEL_LOW);

        R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_SECONDS);

        g_ioport.p_api->pinWrite(Leds.p_leds[BSP_LED_LED1],
                                IOPORT_LEVEL_LOW);
        g_ioport.p_api->pinWrite(Leds.p_leds[BSP_LED_LED3],
                                IOPORT_LEVEL_HIGH);

        R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_SECONDS);
    }
}
```

While writing the code, you can always use the auto-completion feature of e² studio. Just press <ctrl>-<space> and a window displaying possible completions for the structure or function will appear. If you click on an entry, it will be automatically inserted into the code.


Another helpful tool while writing your code is the Developer Assistance, which can be accessed from the Project Explorer. After having configured the software stacks for your project with the Synergy Configurator, this tool supports you in getting started quickly with your application code. To access the Developer Assistance, first expand your project in the Project Explorer so that the tool is shown. With the tool visible, expand the tree further until you see the stack module and its APIs of your choice. Select the API you want to use and drag and drop the call to it into your source file.

It is now your turn: Please enter the lines of code above into the *hal_entry.c* file in your project. For that, expand the *src* folder of your project and double-click on the file. This will open it in the editor. If you do not want to type everything, you can also download a complete project from the Website of this book (www.renesas.com/synergy-book).

8.1.4 Compiling the First Project

When you did all the typing, the program is ready to be built. There are two different configurations for a build: Debug and Release. The Debug configuration will include all information necessary for debugging a program, like variable and function names and will also turn off certain optimizations of the compiler, for example loop unrolling. This makes debugging easier,

but will create larger and slower code. The Release configuration will strip all this information from the output file and turn on full optimization, thereby creating smaller and faster code, but you will no longer, for example, be able to view variables, unless you know their addresses in memory.

For your first test, the Debug configuration, which is also the default, is the way to go. To build your project, click on the build-button on the main menu bar  and the process will start. If you did everything right, the compilation will finish with 0 errors and 0 warnings. If there are compile-errors, you need to go back to your code and double-check, if you entered everything correctly. If not, change your code accordingly.

After the build succeeds the output file *MyBlinkyProject.elf* has been created, which needs to be downloaded to the processor before we can run and debug it.

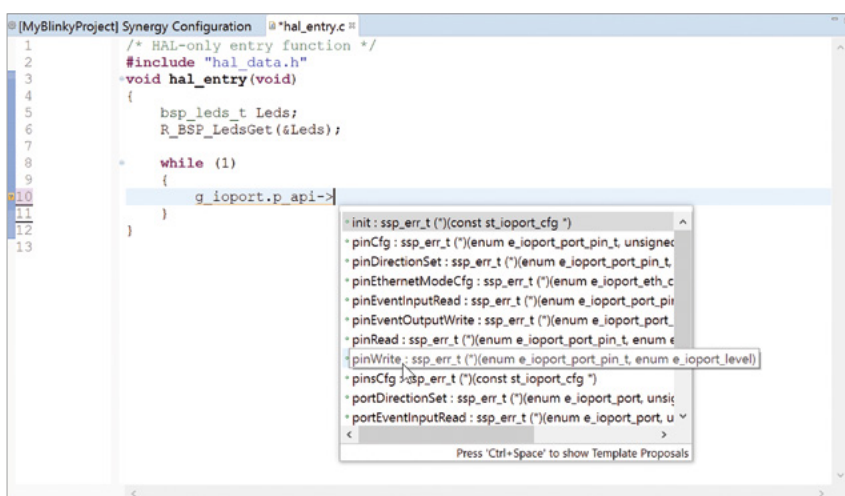



Figure 8-7: Pressing <ctrl>-<space> at a variable or function will activate the code completion feature of the editor

8.1.5 Downloading and Debugging the First Project

The next step will be to actually run the program on the Promotion Kit. And this is now the right moment to connect the kit to your Windows® workstation: Insert the micro-B end of the USB-cable delivered with the board into the connector called *DEBUG_USB* (J19) and the other end into the PC. The green LED4 above the top right hand corner of the LCD-display should be lit, indicating that the board has power. If the kit just came out of the box, the pre-programmed demo will run, signalling that everything is working as expected. Windows® might display a dialogue indicting the installation of the J-Link® OB Debugger, which should be completed automatically. Additionally, a window asking for the update of the J-Link® Debug Probe may appear. It is strongly recommended to allow this update to happen.

DOWNLOADING

To download our program, we will have to create a debug configuration first. Click on the small arrow beside the Debug symbol  and select *Debug Configurations* from the drop down list.

In the window showing up, highlight *MyBlinkyProject Debug* under Renesas *GDB Hardware Debugging*. Finally click on *Debug* at the lower right corner of the window. This will launch the debugger, download the code to the Synergy device and will ask you, if you want to want to switch to the *Debug* perspective. Answer *Yes*. The *Debug* perspective will open and the program counter is set to

the entry point of the program, the reset handler. This configuration needs to be done only once. The next time you can start the debugger by only clicking on the Debug symbol.

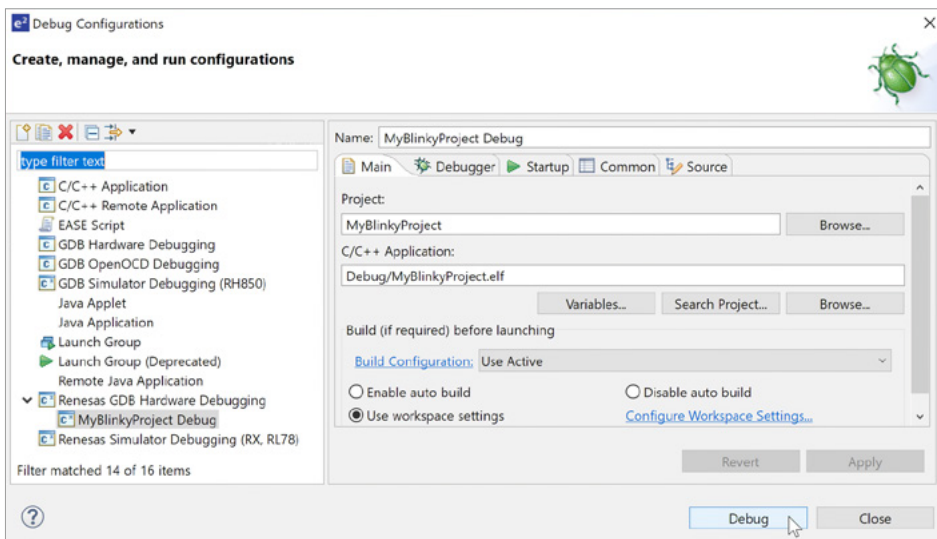




Figure 8-8: Once MyBlinkyProject is selected, no change needs to be made on the different tabs

RUNNING

Click on the Resume button  and the next stop will be at `main()`, at the call to `hal_entry()`. Click again and the program continues to execute, toggling the green and the orange LED on the Promotion Kit in a one second interval as intended.

WATCHING THE RESULTS

If everything is working as expected, click on the Suspend button  on the main menu bar. This will stop the execution of the program without terminating it. In the editor view, activate the tab with the file `hal_entry.c`, and right-click in one of the lines with a write to the ports; in the menu showing, select *Run to line*. Execution will resume and the program will stop at the line you clicked in. Now have a look at the view with the variables tab at the right. You will see the `Leds` structure listed. Expand it and browse and analyse the different fields. This view will come handy when debugging a larger project

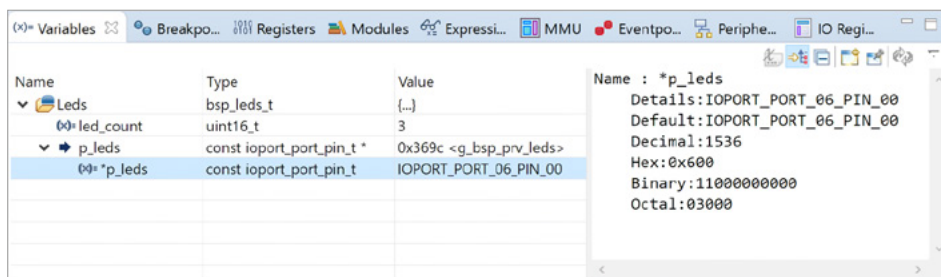


Figure 8-9: Variables and their values can be observed in the variables tab

The final step is to disconnect the debugger from the board by clicking on the *Terminate* button , stopping the execution of the program.

CONGRATULATIONS!

You mastered your first program for the Synergy Platform!

8.2 Your First Project Using IAR Embedded Workbench® for Renesas Synergy™

8.2.1 Creating a New Project

If not already done, start IAR Embedded Workbench® for Renesas Synergy™ (EW for Synergy) from your Windows® workstation Start Menu. Once the IDE is up and running, you might want to dismiss the *Information Center* by clicking on the *X* at the far right hand side, as you do not need it for this exercise.

Writing a new program for a microcontroller in EW for Synergy requires you to create a new project first, so this is the first step you need to take.

For this, go to *Renesas Synergy* → *New Synergy Project* and the *Save Workspace As* dialog will show. Navigate to a directory of your choice and give the workspace a name, for example *MyBlinkyWorkspace*. Click on *OK*. Next, a new dialog named *Renesas Synergy Settings* will show. If you did the previous exercises, all necessary settings should have been already made, otherwise you need to fill in the information required. First, you need to enter the directory, into which the Renesas Synergy™ Standalone Configurator (SSC) has been installed. If you used the Platform Installer, this was *C:\Renesas\Synergy\ssc_v7.5.1_ssp_v1.7.0* by default, or if you decided to go for the Standalone Installer it was *C:\Renesas\Synergy\SSC_v7_5_1_V20190813*. If you used your own path, enter this one. If you are unsure, refer to chapter 4 for more details.

Next you need to provide the correct path to the license file for the Synergy Software Package (SSP). The navigation button will guide you to the directory where the evaluation license was placed during installation. You can find it in the *\internal\projectgen\licenses* subdirectory of one of the installation directories given above (see Figure 8-10). Select the file (there will be only one) named *SSP_License_Example_EvalLicense_20190725.xml* (or similar). If you already requested and received a development and production license either from Renesas or from the Super User of your company, point the file selector to the correct path for that one. Once one of the license files is loaded, you can examine the permissions for the different components of the SSP inside the License Information frame.

The last item in the dialog asks you whether or not you want to like to replace any encrypted source files with decrypted ones. As this option is only applicable, if your license is a Renesas Synergy development and production license, you can leave this item unchecked, Click on *OK*.

A second *Save As* dialog will show, where you need to provide a location and a name for your project. It is important to create a new directory named for example *MyBlinky* and not to reuse one of the previous exercises. Give the project a name, maybe *MyBlinkyProject*. Click on *Save* and the SSC will start. Be patient, this might take a couple of seconds.

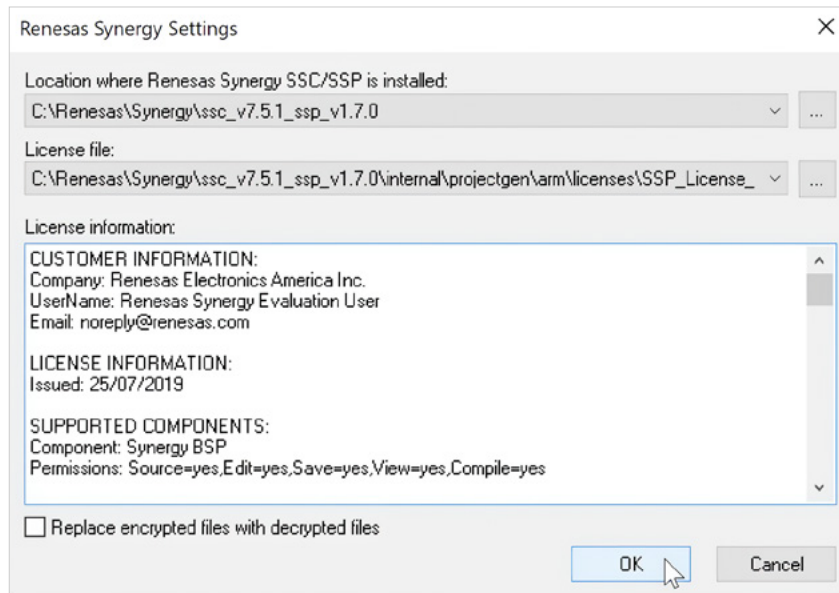


Figure 8-10: You will need to enter the correct paths for the SSC and the license file

In the Synergy Standalone Configurator look out under *Device Selection* for the field called SSP version: It should show the same version of the Synergy Software Package you downloaded before. Under *Board*, select *S5D9 PK*, as this is the hardware we want to use for our small »Hello World« program. Verify that *R7FS5D97E3A01CFC* is shown beside *Device*; it should have been automatically selected. If not, navigate through the drop down list until you found it. With this done, click on *Next* to open the *Project Template Selection* screen (see Figure 8-12). A project template may include several

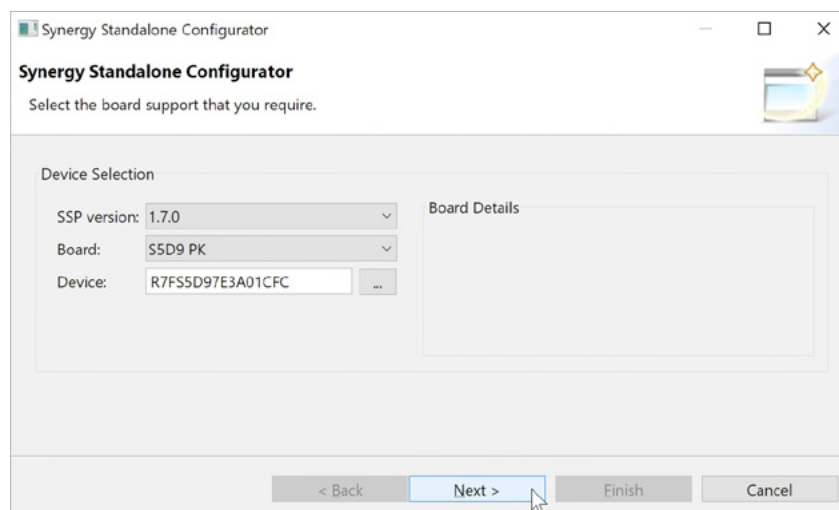


Figure 8-11: Make sure to select the S5D9 PK, as this is the board used for the exercise

items; at the very least it includes the correct Board Support Package for the selected board / device combination. Some templates even include a complete example project. In our case, select *BSP*, which will load the Board Support Package for the Promotion Kit. Click on *Finish*.

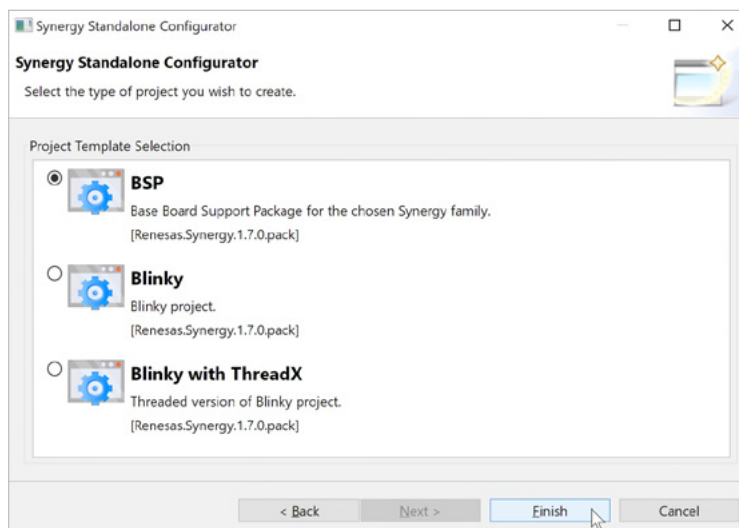


Figure 8-12: Selecting the BSP entry will load the correct Board Support Package

The *Project Configurator* will close and will create all the necessary files for the project as a last step. Once this postprocessing is complete, the SSC will open its *Synergy Configuration* perspective.

8.2.2 Setting Up the Runtime Environment with the Synergy Configurator

Once the Synergy Configurator has started, it presents you a summary of the project and a short overview over the Synergy microcontroller selected (refer to Figure 8-6 for a screenshot). It also provides a convenient shortcut to the support page on the Renesas Internet site, from where you can access various help features.

The following tab, called *BSP*, allows you to view and edit aspects of the board setup, while in the *Clocks* tab, the initial clock configuration for the project, can be set. Any potential problem will be highlighted in red, and hovering the mouse over a highlight will display an explanation. The fourth tab in the configurator, called *Pins*, covers the initial pin setup of the Synergy device in the project. Pins can be listed either based on ports or peripherals.

A package view, again with error marking, is available just at the right hand side of the configurator, reducing possible mistakes to a minimum. The *Threads* tab, allows you to add and configure different components. Since we do not use an RTOS in this project, there is only a *HAL/Common* entry, showing the necessary basic modules. In the *Messaging* tab following, you can configure the Messaging Framework for ThreadX® based projects. The final tab *Components* displays the elements available in the SSP and which of them are currently included in the project. It also allows you to switch between different versions of the same module, in case you have multiple versions of the SSP installed. Modifications like adding or removing components shouldn't be made here, but preferably in the *Threads* tab, as they can be configured there as well.

For our project, there is no need to change anything in this configurator, as all necessary settings have been done for us already by the *Project Configurator*. As a final step in the Synergy Configuration, additional source code based on the current configuration needs to be created. Click on the *Generate Project Content* button at the top right hand side of the *Synergy Configurator*.

With that, the required files will be extracted from the SSP, adjusted to the settings made in the configurator and added to the project. Once this process has finished, close the SSC by clicking on the close button of the window (the 'X' at the top right hand side). This will get you back to the IAR Embedded Workbench® for Renesas Synergy™ and will update the workspace of the IDE. You can open the SSC anytime again by clicking on the *Synergy Configuration* icon on the toolbar, selecting *Renesas Synergy* → *Configurator* in the menu or *by right-clicking* on the *Synergy* entry in the project tree and selecting *Open Renesas Synergy Configurator*.

8.2.3 Writing the First Lines of Code

With all the automatically generated files now in place, it is time to have a look on what was created. The *Workspace* window at the left hand side of the IDE lists everything currently included. Expand all entries under *Synergy*. The *src* folder contains a subfolder called *synergy_gen*, holding configuration sets such as channel number etc. The *src* folder also includes a file named *hal_entry.c*. This is the one you will edit later on. Please note that while there is a file called *main.c* in the *synergy_gen* folder, your user code must go to *hal_entry.c*. Otherwise you will lose your changes if you make modifications in the Synergy Configurator and re-created the project contents, as this file will be overwritten each time you click on *Generate Project Content*.

The project also contains several directories with »synergy« in the name, containing source-, include- and configuration files for the SSP. It is a general rule that the contents of these folders (and subfolders) should not be modified. They contain files generated by the configurator and any modification made there will be lost the next time the project contents are generated or refreshed. The user editable source files are those directly in the root of the *\src* folder or any other folder added by you.

Now it is time for you to write the first real Synergy Platform source code. The plan is to alternate between the green LED1 and orange LED3 on the S5D9 Promotion Kit every second, so you will have to add code for turning them on and off and for a delay loop. How to do that?

There are actually two options for that: One is using the API of the Hardware Abstraction Layer directly and one is using the HAL together with the BSP. Which one do you think is the better one? You can review [chapter 2](#), if you are unsure about the answer.

Looking at the code in the file *\src\synergy_gen\common_data.c*, we find that there is the following definition for the I/O port driver instance *g_ioport*:

```
const ioport_instance_t g_ioport =
{ .p_api = &g_ioport_on_ioport, .p_cfg = NULL };
```

g_ioport_on_ioport is a structure, which declares the possible actions for the ports and is assigned to the API-pointer of the *g_ioport* instance. The contents of the structure can easily be viewed by hovering with the mouse over it, which will reveal that one of its members, (*.pinWrite*), is a pointer to the pin write function.

So turning an LED on, you could write:

```
g_ioport.p_api -> pinWrite(ioport_port_pin_t pin,
                          IOPORT_LEVEL_LOW);
```

But this means that you would actually need to know which I/O-ports LED1 and LED3 are connected to and how many LEDs are available for use! For that, we could either read the documentation of the board or scrutinize the schematics to find the correct port. Or simply use the BSP API, which provides the structure `bsp_leds_t` for that purpose. Calling the BSP function:

```
R_BSP_LedsGet(bsp_leds_t * p_leds);
```

will then populate an user defined variable of the type `bsp_leds_t` (e.g. `Leds`) with the necessary values.

This means that you can turn on LED1 by writing:

```
g_ioport.p_api->pinWrite(Leds.p_leds[BSP_LED_LED1],
                        IOPORT_LEVEL_LOW);
```

You will need a second statement right after this, turning LED3 off, by setting its pin-level to high.

Finally, you need to provide a delay to have the LEDs toggle in a user friendly way. For that, call again a *BSP API*:

```
R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_SECONDS);
```

This will create a one second delay loop. The macro `BSP_DELAY_UNITS_SECONDS` defines the units for the delay, either in seconds, milliseconds or microseconds, while the 1 stands for the number of units to delay.

All what is needed once this is done, is to copy/paste the three lines of code and to reverse the pin-levels of the LEDs in the second set. And finally, as we want to run the program indefinitely, a `while(1)`-loop needs to be created around the code.

What is left now is to insert the following lines of code into the `hal_entry.c`:

```
void hal_entry(void)
{
    bsp_leds_t Leds;
    R_BSP_LedsGet(&Leds);

    while (1)
    {
        g_ioport.p_api->pinWrite(Leds.p_leds[BSP_LED_LED1],
                                IOPORT_LEVEL_HIGH);
        g_ioport.p_api->pinWrite(Leds.p_leds[BSP_LED_LED3],
                                IOPORT_LEVEL_LOW);

        R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_SECONDS);

        g_ioport.p_api->pinWrite(Leds.p_leds[BSP_LED_LED1],
                                IOPORT_LEVEL_LOW);
        g_ioport.p_api->pinWrite(Leds.p_leds[BSP_LED_LED3],
                                IOPORT_LEVEL_HIGH);

        R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_SECONDS);
    }
}
```


While writing the code, you can always use the auto-completion feature of the editor. Just press `<ctrl>-<space>` and a window displaying possible completions for the function or structure will appear. If you click on an entry, it will automatically be inserted into the code.

It is your turn now. Please enter the lines of code above into the `hal_entry.c` file in your project. For that, expand the `src` folder of your project and double-click on the file. This will open it in the editor. If you do not want to type everything, you can also download a complete project from the Website of this book (<https://www.renesas.com/synergy-book>).

8.2.4 Compiling the First Project

When you did all the typing, the program is ready to be built. There are two different configurations for a build: Debug and Release. The Debug configuration will include all information necessary for debugging a program, like variable and function names, and will also turn off certain optimizations of the compiler, for example loop unrolling. This makes debugging easier, but will create larger and slower code. The Release configuration will strip all this information from the output file and turn on full optimization, thereby creating smaller and faster code, but you will no longer, for example, be able to view variables, unless you know their address in memory.

For your first test, the Debug configuration, which is also the default, is the way to go. To build your project, click either on the make-button on the toolbar, or select *Project* → *Make* or simply hit the *F7*-key and the process will start.

If you did everything right, the compilation will finish with 0 errors and 0 warnings. If there are compile-errors, you need to go back to your code and double-check that you entered everything correctly. If not, change your code accordingly.

After the build succeeds, the output file `MyBlinkyProject.out` is created, which needs to be downloaded to the processor before we can run and debug it.

8.2.5 Downloading and Debugging the First Project

The next step will be to actually run the program on the Promotion Kit. And this is now the right moment to connect the kit to your Windows® workstation: Insert the micro-B end of the USB-cable delivered with the board into the connector called `DEBUG_USB` (J19), and the other end into the PC. The green LED4 at the top right hand corner of the PCB should be lit, indicating that the board has power. If the kit just came out of the box, the pre-programmed demo will run, signaling that everything is working as expected. Windows® might display a dialog indicating the installation of the J-Link® OB Debugger, which should be completed automatically. Additionally, a window asking for the update of the J-Link® Debug Probe might show. It is strongly recommended to allow this update to happen.

To start your debugging session, select either *Project* → *Download and Debug* from the menu or click on the *Download and Debug* icon on the toolbar. The IAR Embedded Workbench® for Renesas Synergy™ will download your code, switch to the debugger and stop at the entry point of the program, the call to `main()`.

Click twice on the *Step Into* icon on the toolbar of the debugger and you will enter the routine `hal_entry()`. Next step is to finally run the program. For that, click on the *Go* icon on the debugger toolbar and the program continues to execute, toggling the green and the orange LED on the Promotion Kit at a one second interval as programmed.

If everything is working as expected, click on the *Break* button on the debugging toolbar. This will stop the execution of the program without terminating it. In the editor window, activate the tab with the file `hal_entry.c`, and right-click in one of the lines with a write to the ports; in the menu showing, select *Run to Cursor*. Execution will resume and the program will stop at the line you clicked in. Now call up one additional window to watch the local variables. Click on *View* → *Locals* in the main menu and a window displaying the `Leds` structure will show. Expand it and browse and analyze the different fields. This view will come handy when debugging a larger project.

Variable	Value	Location	Type
Leds	<struct>	0x1FFE0FE0	bsp_leds_t
led_count	3	0x1FFE0FE0	uint16_t
p_leds	g_bsp_prv_leds (0x2848)	0x1FFE0FE4	ioport_port_p...
IOPORT_PORT_06_PIN_00		0x00002848	ioport_port_p...

Figure 8-13: Local variables and their values can be observed in this window

The final step is to terminate the debugging session by clicking on the *Stop Debugging* icon on the toolbar of the debugger. This will remove the debugger from the target, which now runs freely and returns the IDE to editing mode.

CONGRATULATIONS!

You mastered your first program for the Synergy platform!

Points to take away from this chapter:

- The Project Configurator creates all files and settings necessary for a new project.
- The Synergy Configurator allows the Synergy Software Package and the runtime environment to be configured easily based on a graphical user interface.
- A Debug Configuration is necessary for debugging the project, but it is created automatically.
- Only very few lines are needed for the implementation of the actual code.

Copyright: © 2020 Renesas Electronics Corporation

Disclaimer:

This volume is provided for informational purposes without any warranty for correctness and completeness. The contents are not intended to be referred to as a design reference guide and no liability shall be accepted for any consequences arising from the use of this book.