HAL Module Synergy Lab

Devices: Synergy S7G2 / S5D9

Description:

The objective of this lab session is to introduce you to creating a Synergy application utilizing the HAL modules. The application will periodically generate a random number via the Hardware TRNG (True Random Number Generator) and use this value as output to the DAC.

The output of the DAC will be connected to the input of the ADC. The application will then periodically sample the ADC and the result will be stored in an array. The transfer of the ADC result into memory will be performed via the DTC – Data Transfer Controller.

The process of creating the application will introduce you to adding and configuring HAL modules to the Synergy Configuration window within e2-Studio. This process is the basis of all Synergy projects and is the basis for creating far more advanced applications which will be introduced in the RTOS & Frameworks Lab this afternoon.

The lab is split into several sections:

Section 1: Creating the Synergy Application

Section 2: Add the DAC, Timer & TRNG modules to you Synergy Application.

Section 3: Add application code that shows the API controlling the previously mentioned modules.

Section 4: Add the ADC, Timer & DTC modules to you Synergy Application.

Section 5: Add application code that shows the API controlling the previously mentioned modules.

Lab Objectives
Create a Synergy Application.
Add & Configure DAC, Timer, TRNG modules.
Add API calls to use these modules.
Add ADC, Timer and Transfer modules.
Add API calls to use these modules.

Lab Materials
ISDE e^{2}studio 5.4.0.023
GNU Tools for ARM
(gcc-arm-none-eabi-4_9-2015q3-20150921)
Synergy SSP 1.3.3
Synergy S7G2-SK or S5D9-PK

Skill Level
Programming in C

Time to Complete Lab
90 Minutes

Section 1: Creating the Synergy Application

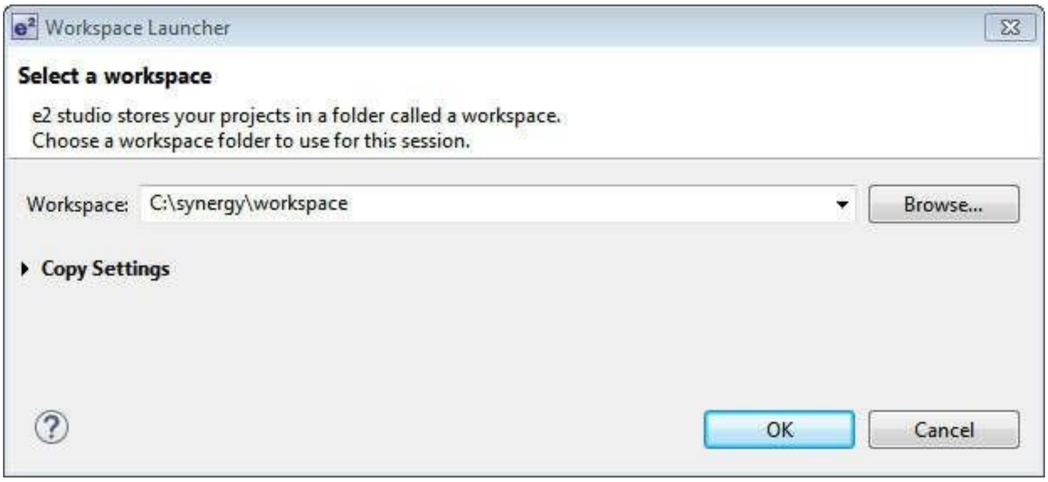
1. Open ISDE by double clicking the icon on the desktop

2. Specify a Workspace directory location.

   This example will use:

   C:\synergy\workspace

   If possible please use this location for your project. If not, then please ensure that the path has no spaces and no extended ASCII characters. The GCC ARM compiler can have issues if the project path contains these.



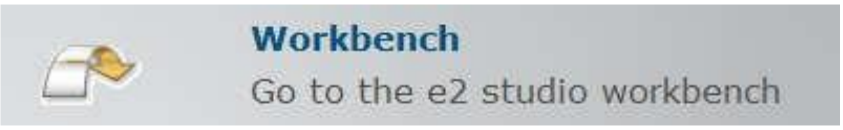
Workspace Launcher

Select a workspace

e2 studio stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\synergy\workspace

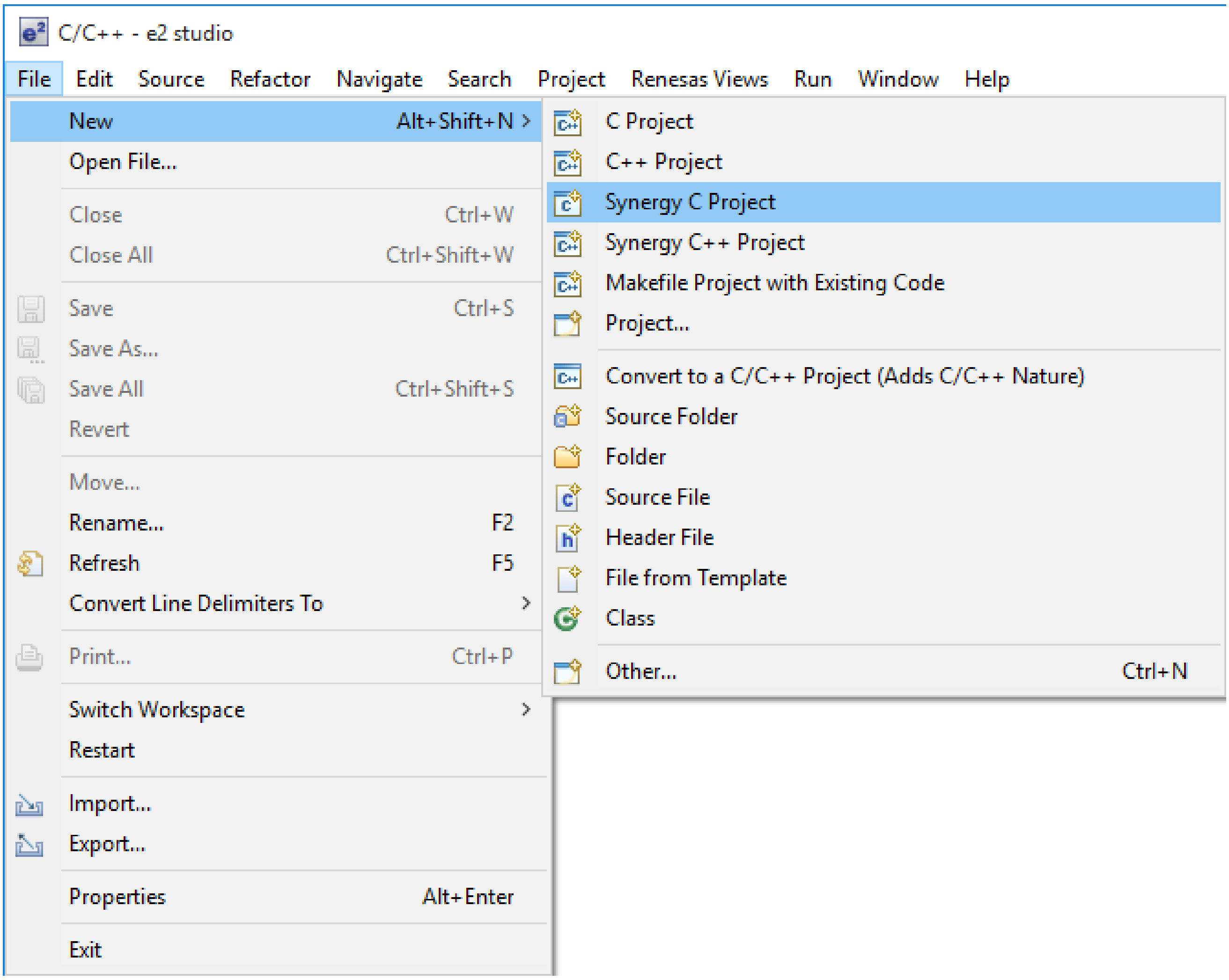
3. When the “Welcome to e2 studio” screen in shown, navigate to the e2 studio workbench



Workbench

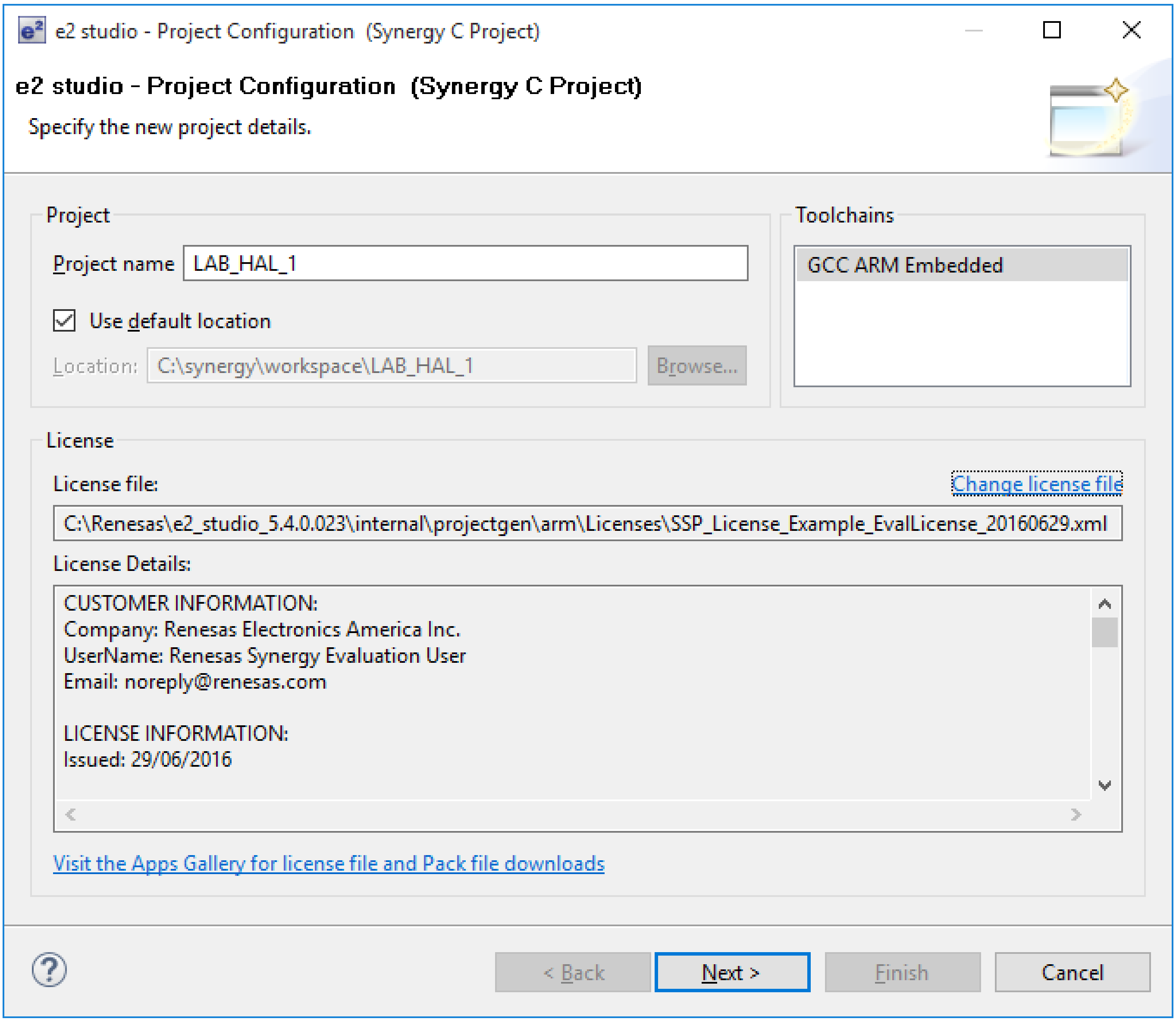
Go to the e2 studio workbench

4. Create a New Synergy C Project



5. Give the project a meaningful name. This example will use LAB_HAL_1.

6. When you create a project for the first time you will have to specify the location of the Synergy license. The SSP comes with an evaluation license which will have been installed when you installed the SSP. Use this license for this lab.



e2 studio - Project Configuration (Synergy C Project)

e2 studio - Project Configuration (Synergy C Project)

Specify the new project details.

Project

Project name: LAB_HAL_1

☑ Use default location

Location: C:\synergy\workspace\LAB_HAL_1

Toolchains: GCC ARM Embedded

License

License file: C:\Renesas\e2_studio_5.4.0.023\internal\projectgen\arm\Licenses\SSP_License_Example_EvalLicense_20160629.xml

License Details:

CUSTOMER INFORMATION:
Company: Renesas Electronics America Inc.
UserName: Renesas Synergy Evaluation User
Email: noreply@renesas.com

LICENSE INFORMATION:
Issued: 29/06/2016

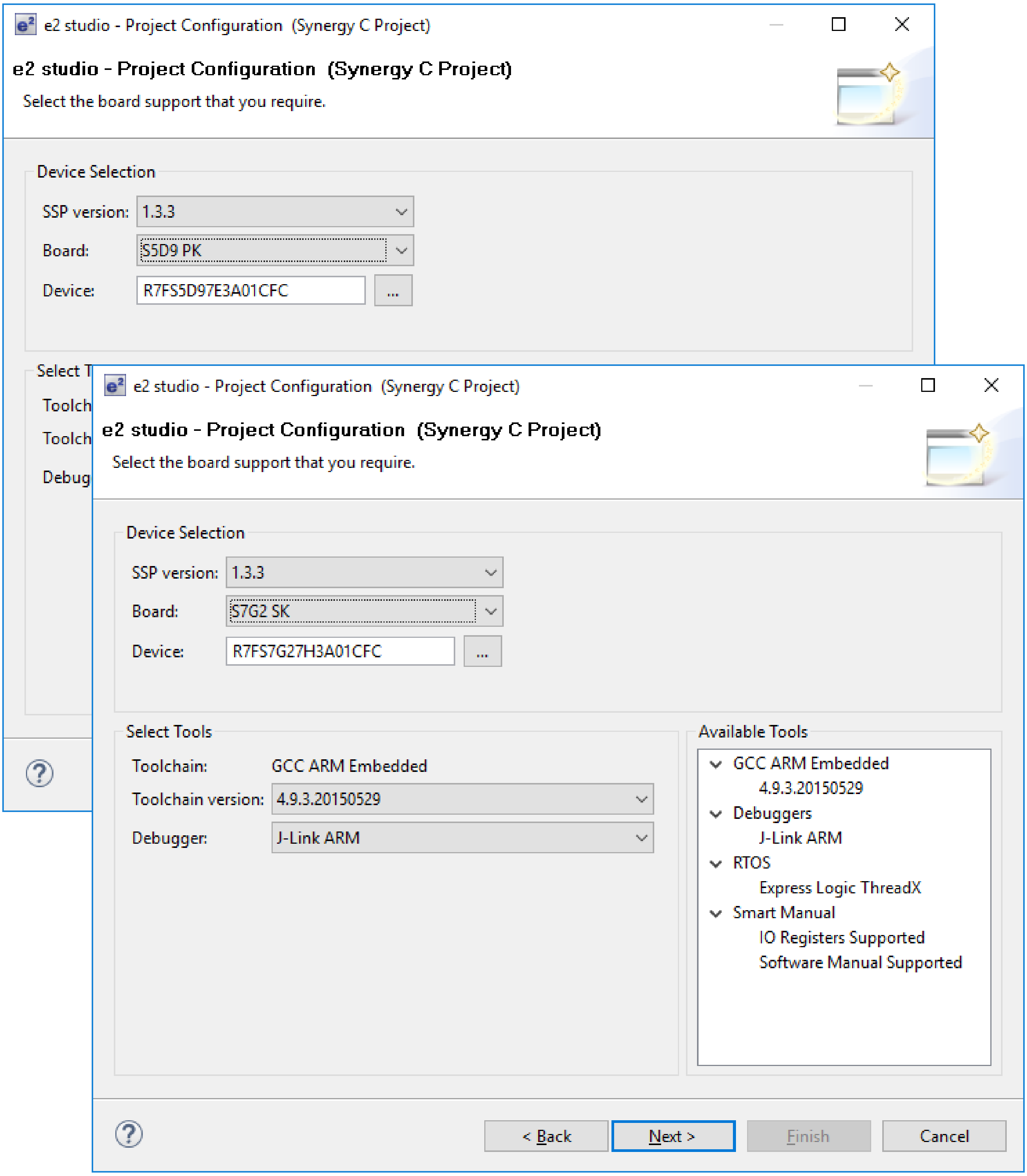
Browse...
Change license file
Visit the Apps Gallery for license file and Pack file downloads
< Back
Next >
Finish
Cancel

Click Next >

7. Specify the SSP version, development board, device and version of toolchain to use.

   Ensure that SSP version **1.3.3** is selected. Select the appropriate board type. You will have been provided with either a **S7G2 SK** or **S5D9 PK**. You will see that when you select the board type that the device is automatically selected for you.

   Ensure that Toolchain version is 4.9.3.20150529



e2 studio - Project Configuration (Synergy C Project)

e2 studio - Project Configuration (Synergy C Project)

Select the board support that you require.

Device Selection

SSP version: 1.3.3

Board: S5D9 PK

Device: R7FS5D97E3A01CFC

Select T

Toolch

Toolch

Debug

e2 studio - Project Configuration (Synergy C Project)

e2 studio - Project Configuration (Synergy C Project)

Select the board support that you require.

Device Selection

SSP version: 1.3.3

Board: S7G2 SK

Device: R7FS7G27H3A01CFC

Select Tools

Toolchain: GCC ARM Embedded

Toolchain version: 4.9.3.20150529

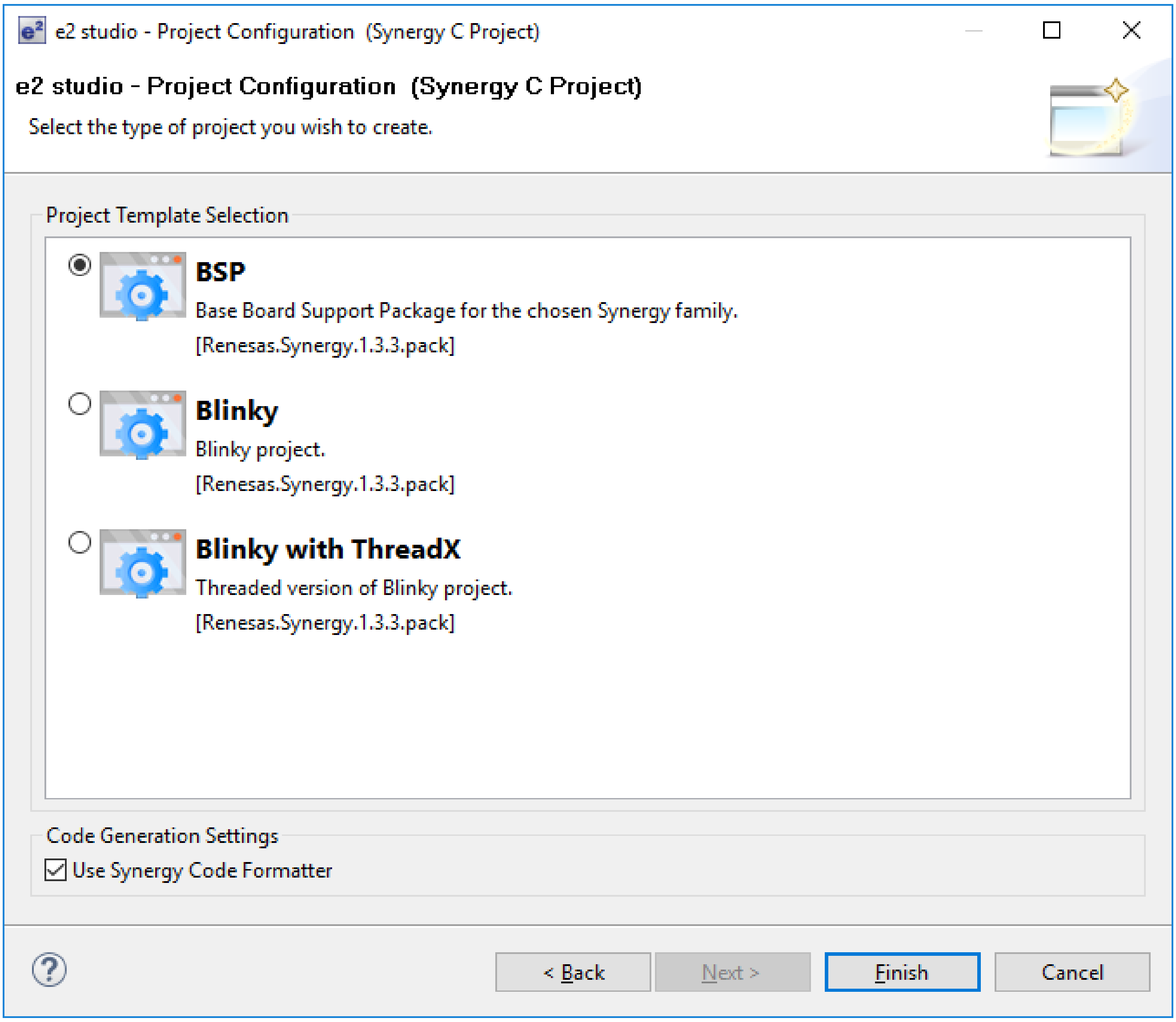
Debugger: J-Link ARM

Available Tools

- GCC ARM Embedded
  - 4.9.3.20150529
- Debuggers
  - J-Link ARM
- RTOS
  - Express Logic ThreadX
- Smart Manual
  - IO Registers Supported
  - Software Manual Supported

Click Next

8. Select the Project Template BSP



e2 studio - Project Configuration (Synergy C Project)

Select the type of project you wish to create.

Project Template Selection

● **BSP**
Base Board Support Package for the chosen Synergy family.
[Renesas.Synergy.1.3.3.pack]

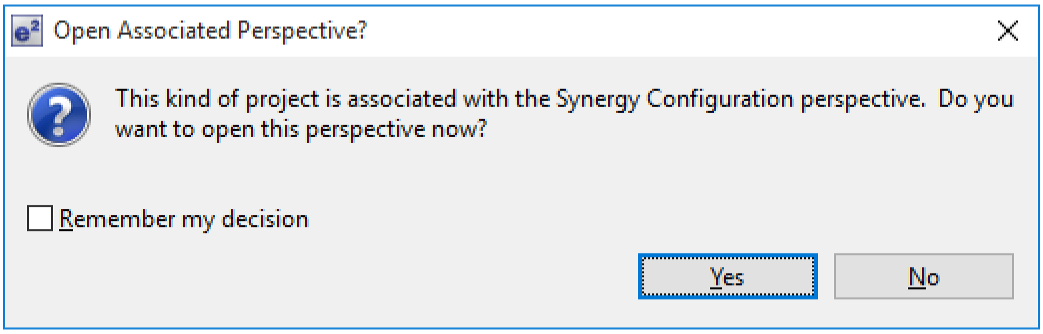
○ **Blinky**
Blinky project.
[Renesas.Synergy.1.3.3.pack]

○ **Blinky with ThreadX**
Threaded version of Blinky project.
[Renesas.Synergy.1.3.3.pack]

Code Generation Settings
☑ Use Synergy Code Formatter

Click Finish

e2 studio will now create the project template, extracting the relevant parts of the SSP based upon the previously selected settings. When prompted, select Yes to open the Synergy Configuration perspective.

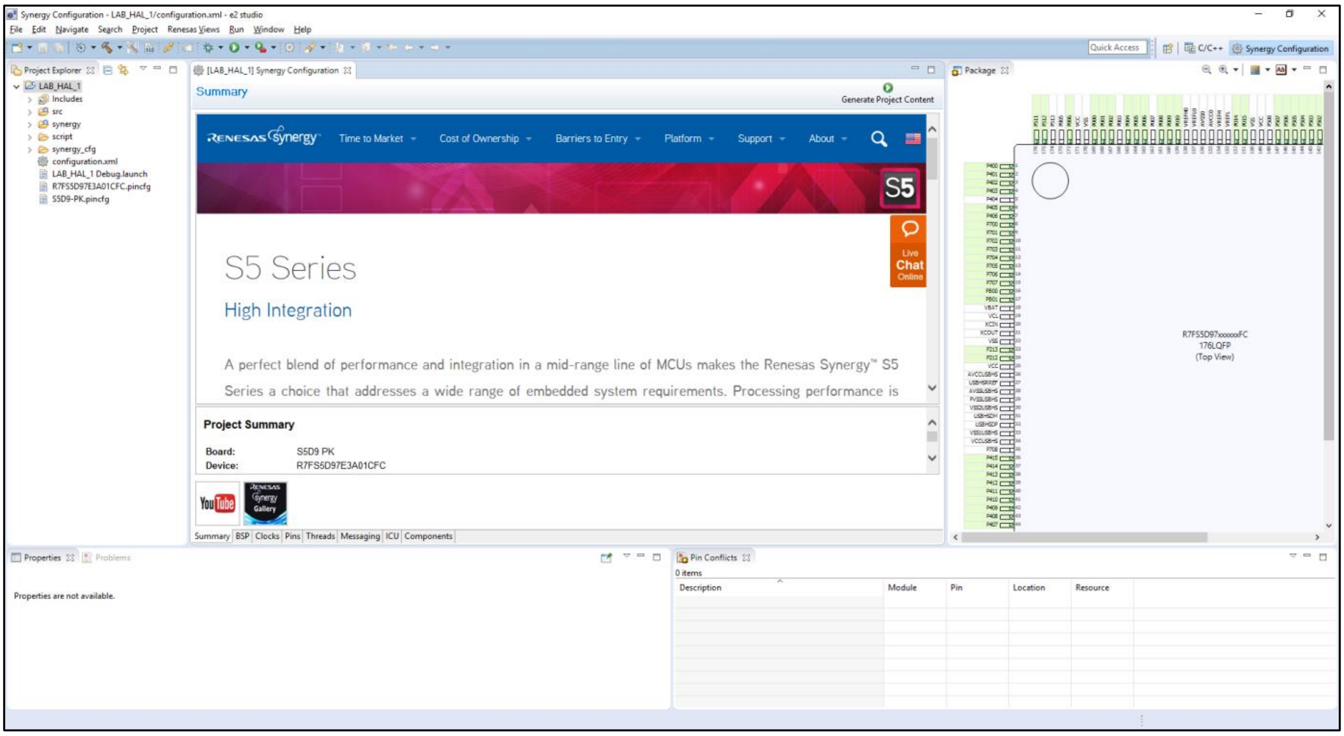


Open Associated Perspective?

This kind of project is associated with the Synergy Configuration perspective. Do you want to open this perspective now?

□ Remember my decision

Once open, the e2 Studio environment will be showing several windows, such as the Project Explorer, Synergy Configuration and Package view.



Synergy Configuration - LAB_HAL_1/configuration.xml - e2 studio

S5 Series

High Integration

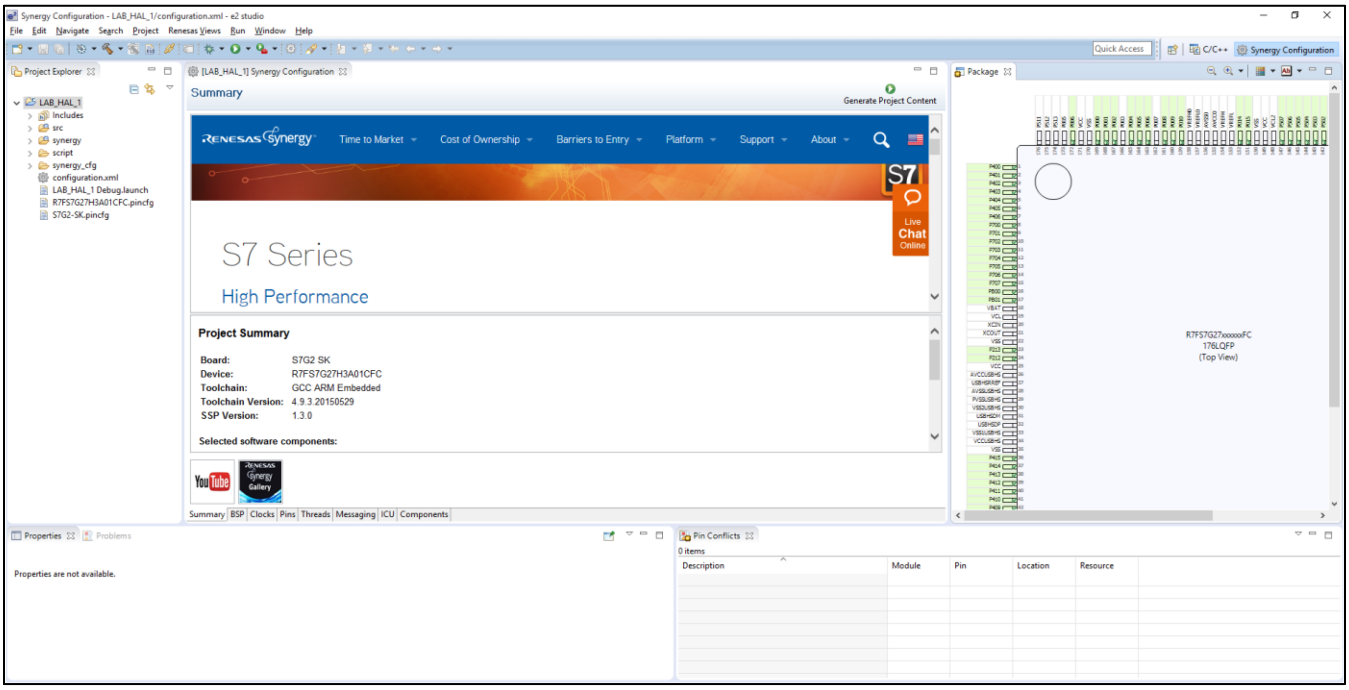
A perfect blend of performance and integration in a mid-range line of MCUs makes the Renesas Synergy™ S5 Series a choice that addresses a wide range of embedded system requirements. Processing performance is

Project Summary

Board: S5D9 PK
Device: R7FS5D97E3A01CFC

Properties are not available.

0 items



Synergy Configuration - LAB_HAL_1/configuration.xml - e2 studio

S7 Series

High Performance

Project Summary

Board: S7G2 SK
Device: R7FS7G27H3A01CFC
Toolchain: GCC ARM Embedded
Toolchain Version: 4.9.3.20150529
SSP Version: 1.3.0

Selected software components:

Properties are not available.

0 items

Section 2: Adding the DAC, Timer & TRNG modules

In this section we will add modules that will periodically generate a random number and use this random number as the DAC output value.

The Timer will be configured to generate a 1 second interrupt. The ISR will set a software flag which the application will be waiting on. When the flag is set the TRNG will generate a number. This value will be written to the DAC.

The diagram below shows the basic application flow of the application.

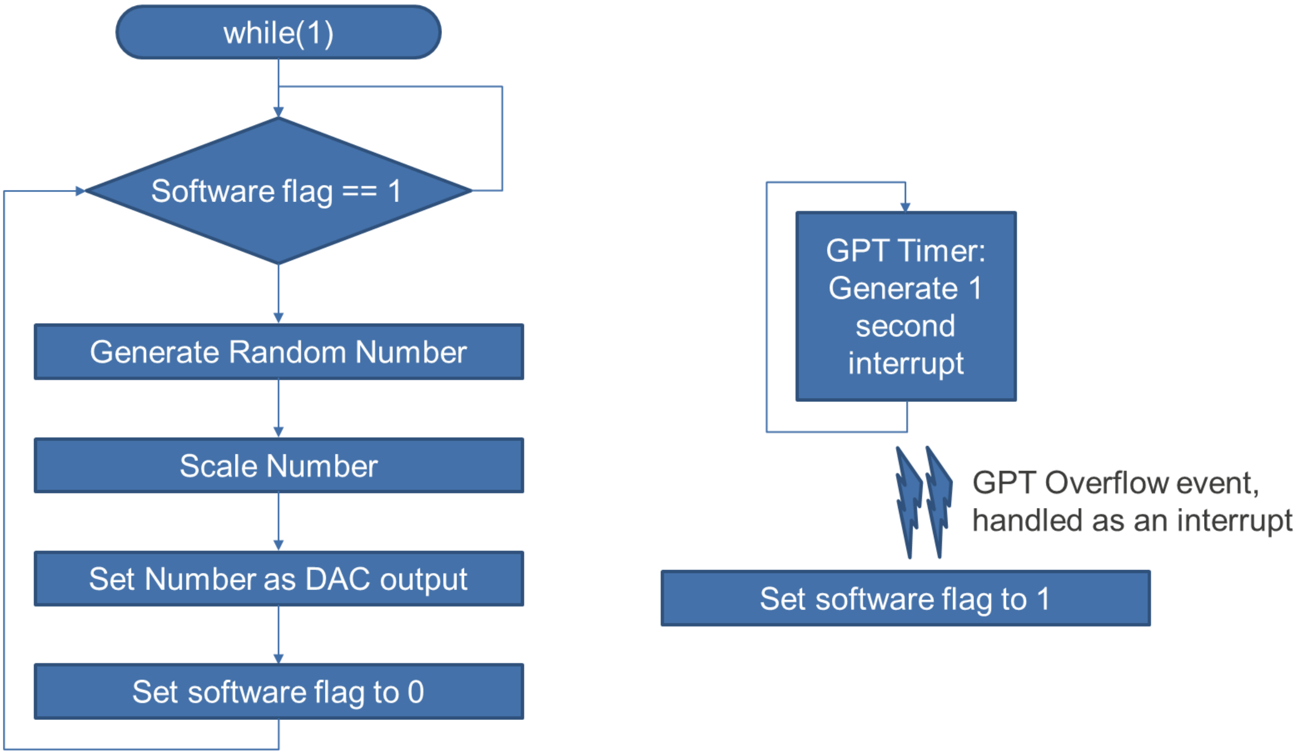
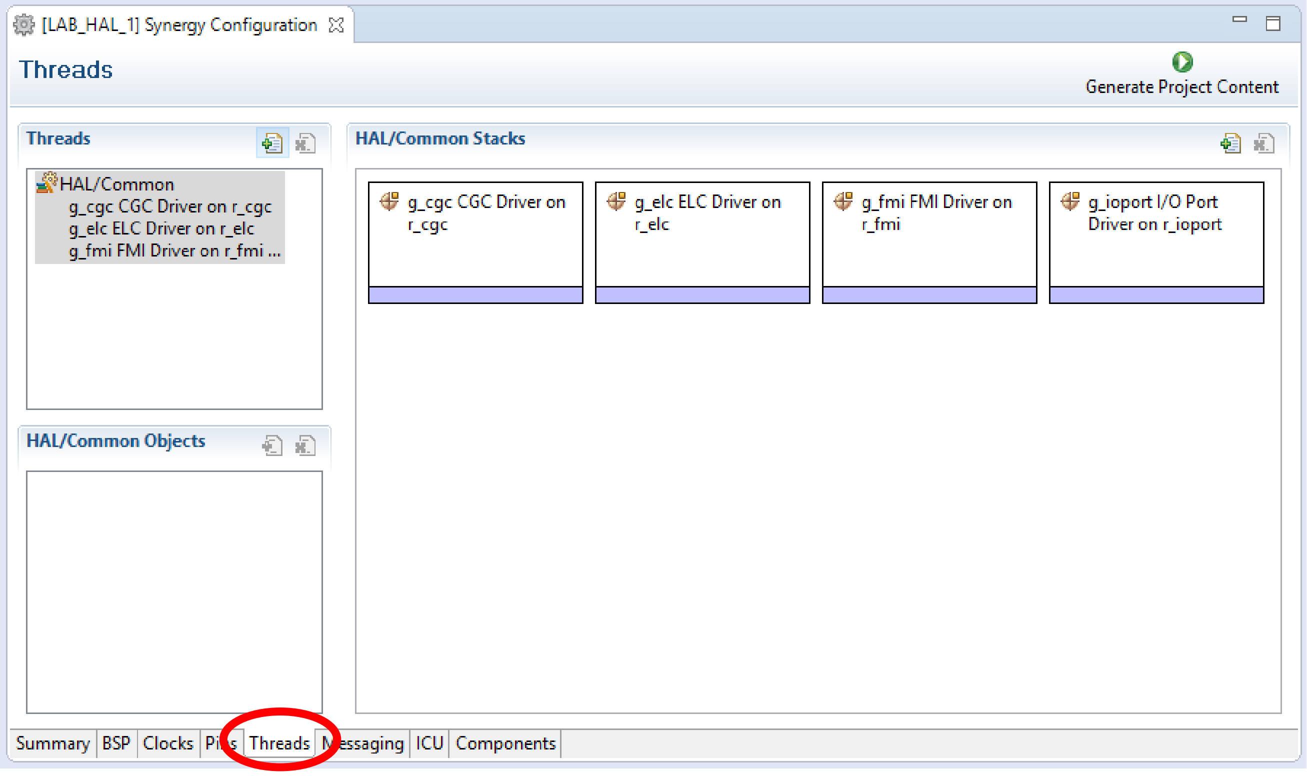


Figure 2.1

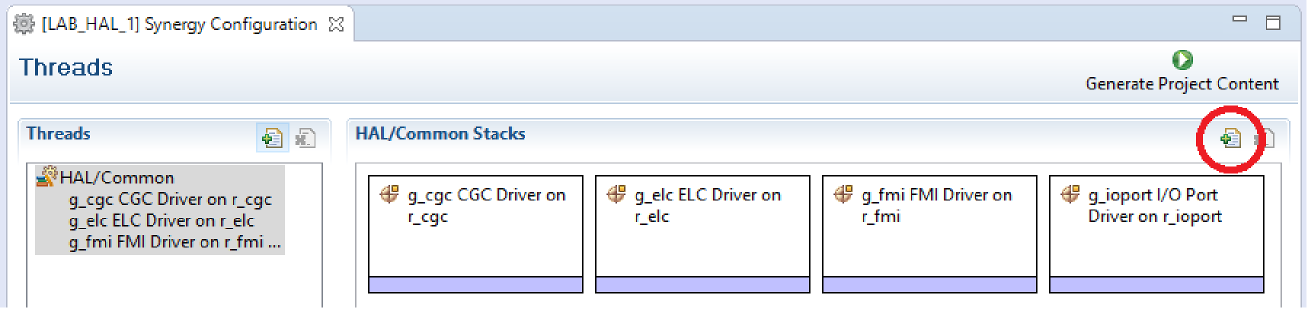
1. In e2 Studio Synergy Configuration view, navigate to the Threads tab.

Clicking on the HAL/Common thread will show that there are 4 modules already added to the project. These 4 modules (CGC, ELD, FMI & I/O Port) are added by default to all projects.


[LAB_HAL_1] Synergy Configuration
Threads
Generate Project Content
Threads
HAL/Common
g_cgc CGC Driver on r_cgc
g_elc ELC Driver on r_elc
g_fmi FMI Driver on r_fmi ...
HAL/Common Objects
HAL/Common Stacks
g_cgc CGC Driver on r_cgc
g_elc ELC Driver on r_elc
g_fmi FMI Driver on r_fmi
g_ioport I/O Port Driver on r_ioport
Summary
BSP
Clocks
Pins
Threads
Messaging
ICU
Components

We will add the DAC, TRNG and Timer modules to this HAL/Common thread.

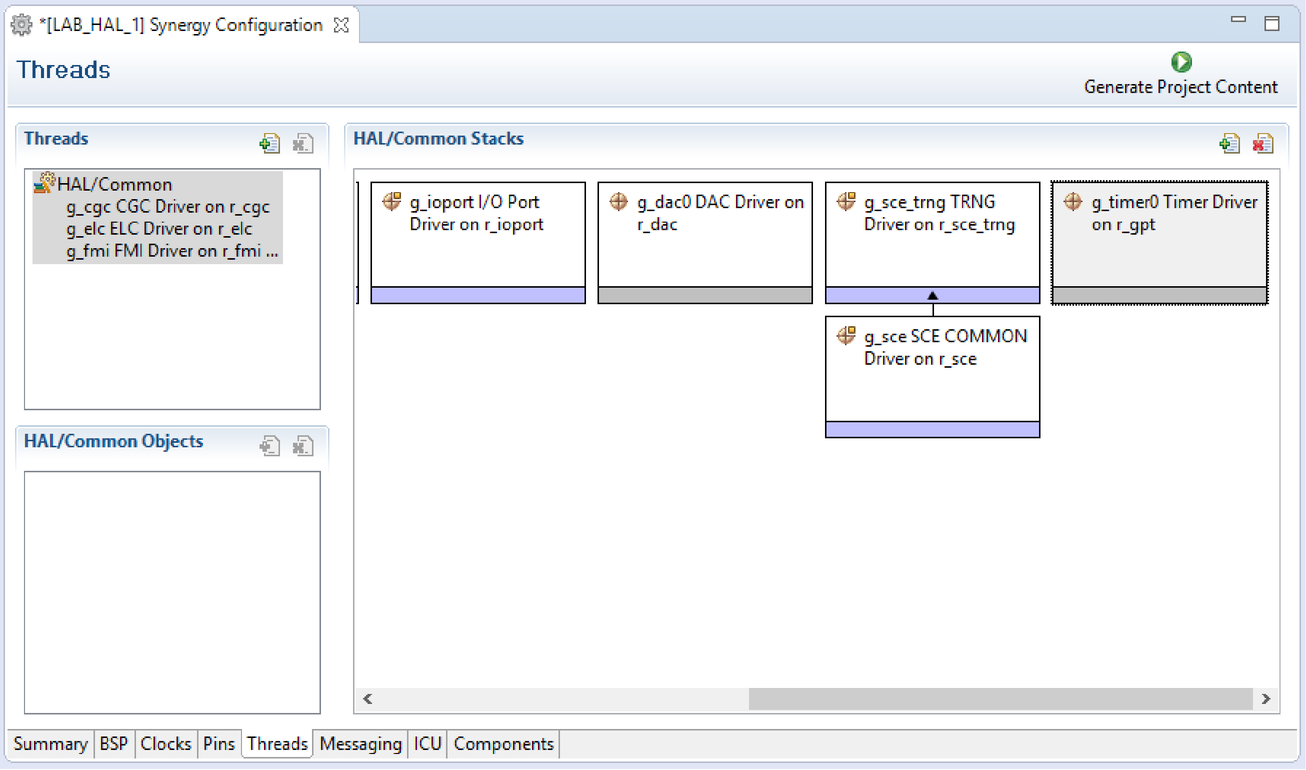
To add a new module to HAL/Common thread, click the + button in the HAL/Common Stacks window


[LAB_HAL_1] Synergy Configuration
Threads
Generate Project Content
Threads
HAL/Common
g_cgc CGC Driver on r_cgc
g_elc ELC Driver on r_elc
g_fmi FMI Driver on r_fmi ...
HAL/Common Stacks
g_cgc CGC Driver on r_cgc
g_elc ELC Driver on r_elc
g_fmi FMI Driver on r_fmi
g_ioport I/O Port Driver on r_ioport

2. Add the modules: Driver > Analog > DAC Driver on r_dac

Driver > Crypto > TRNG Driver on r_sce_trng

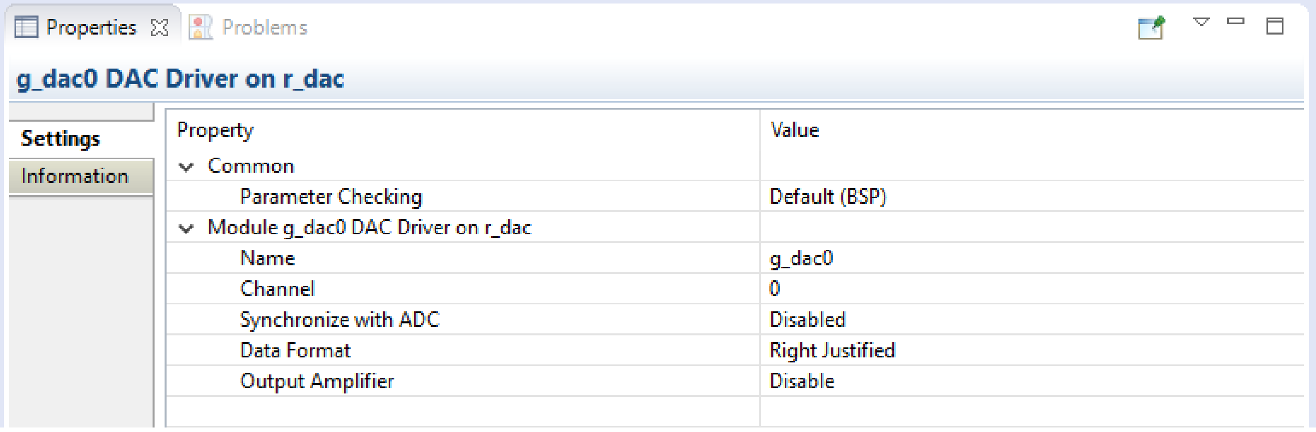
Driver > Timers > Timer Driver on r_gpt


*[LAB_HAL_1] Synergy Configuration
Threads
Generate Project Content
Threads
HAL/Common
g_cgc CGC Driver on r_cgc
g_elc ELC Driver on r_elc
g_fmi FMI Driver on r_fmi ...
HAL/Common Objects
HAL/Common Stacks
g_ioport I/O Port Driver on r_ioport
g_dac0 DAC Driver on r_dac
g_sce_trng TRNG Driver on r_sce_trng
g_timer0 Timer Driver on r_gpt
g_sce SCE COMMON Driver on r_sce
Summary
BSP
Clocks
Pins
Threads
Messaging
ICU
Components

With the modules added, they can now be configured for the desired operation.

Select the DAC module by clicking on the **g_dac0 DAC Driver r_dac** block in the HAL/Common Stacks view.

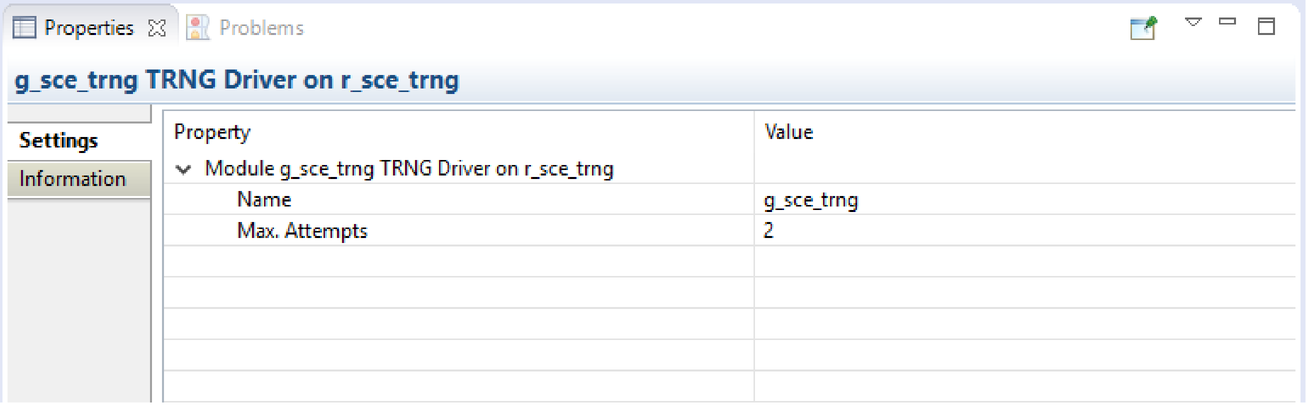
In e2 Studio, the Properties window (typically in the bottom left of the e2 studio environment) will show:



Property	Value
Common	
Parameter Checking	Default (BSP)
Module g_dac0 DAC Driver on r_dac	
Name	g_dac0
Channel	0
Synchronize with ADC	Disabled
Data Format	Right Justified
Output Amplifier	Disable

For this application these **r_dac** default settings do not need to be changed.

Similarly, the default **TRNG** module settings do not need to be changed.



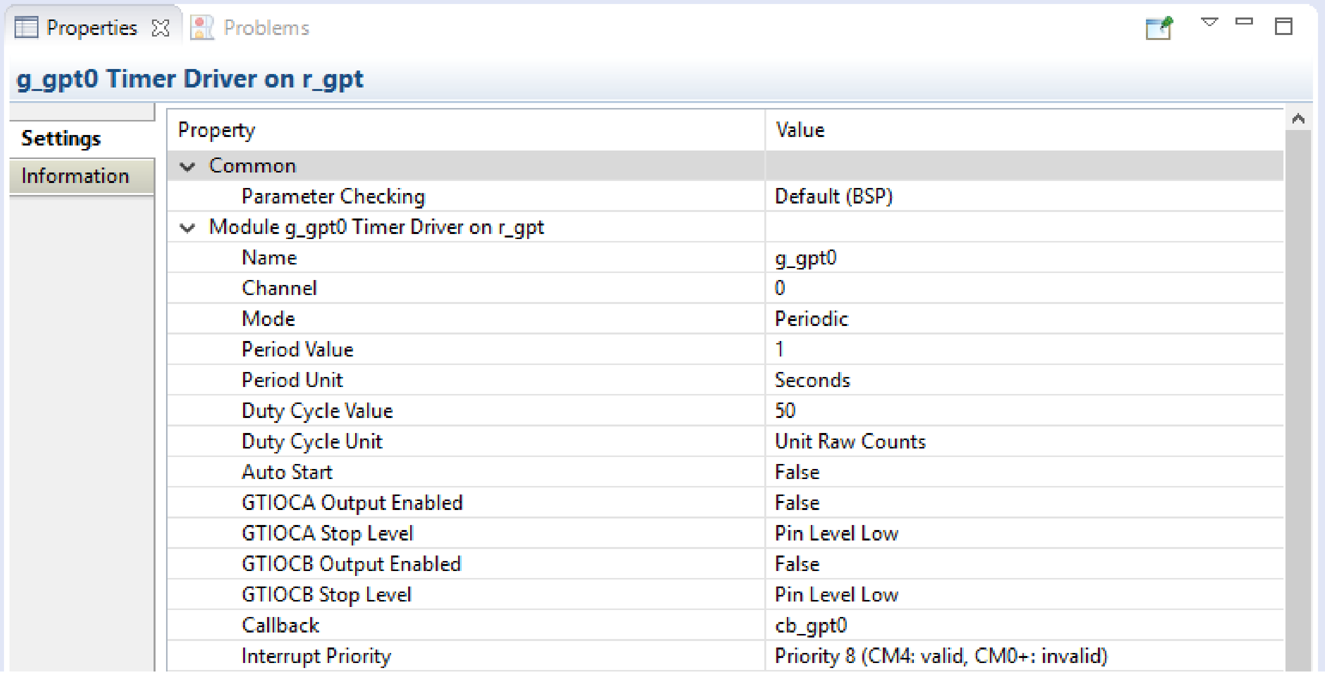
g_sce_trng TRNG Driver on r_sce_trng

Property	Value
Module g_sce_trng TRNG Driver on r_sce_trng	
Name	g_sce_trng
Max. Attempts	2

However, the **r_gpt** module settings do need to be changed.

3. Change:

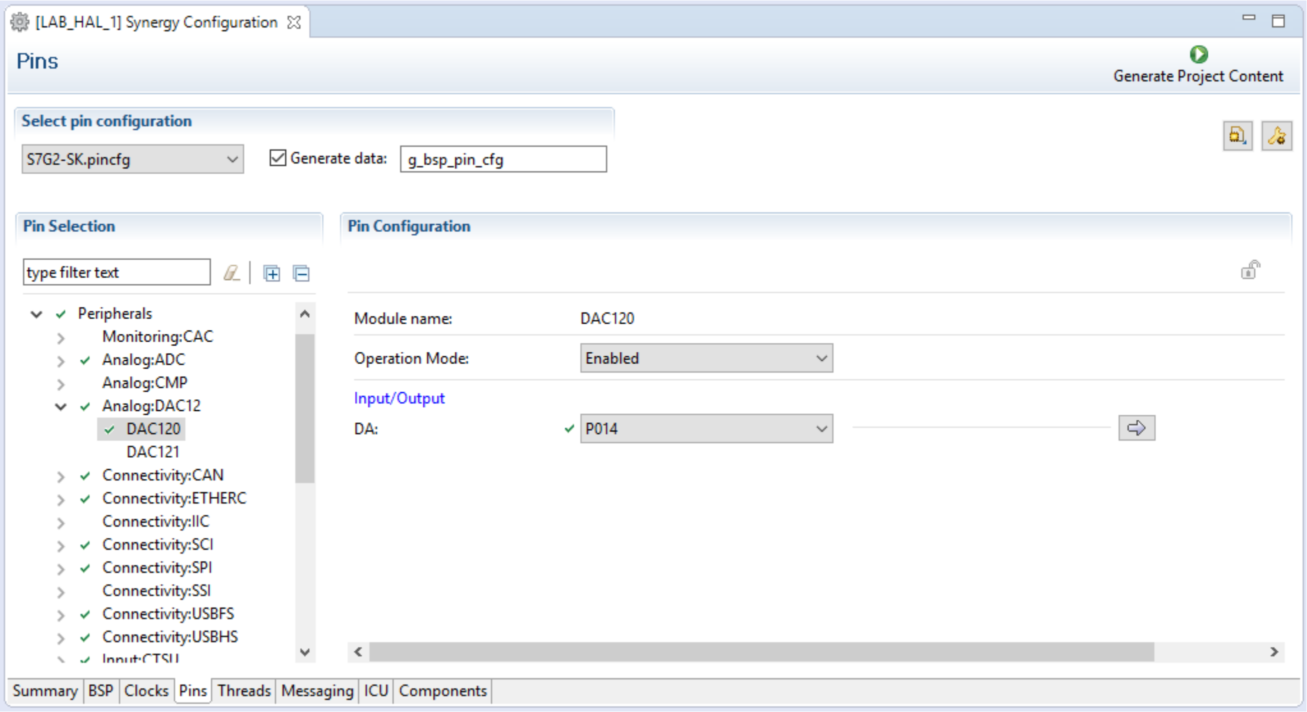
• The name of the timer instance to	**g_gpt0**
• The Period value and Unit to	**1 Second**
• Auto Start to	**false**
• Specify a Callback function to be called: (This function is called when the GPT timer interrupt occurs)	**cb_gpt0**
• Enable the Interrupt be specifying an Interrupt Priority, for example:	**Priority 8**



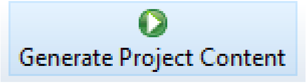
g_gpt0 Timer Driver on r_gpt

Property	Value
Common	
Parameter Checking	Default (BSP)
Module g_gpt0 Timer Driver on r_gpt	
Name	g_gpt0
Channel	0
Mode	Periodic
Period Value	1
Period Unit	Seconds
Duty Cycle Value	50
Duty Cycle Unit	Unit Raw Counts
Auto Start	False
GTIOCA Output Enabled	False
GTIOCA Stop Level	Pin Level Low
GTIOCB Output Enabled	False
GTIOCB Stop Level	Pin Level Low
Callback	cb_gpt0
Interrupt Priority	Priority 8 (CM4: valid, CM0+: invalid)

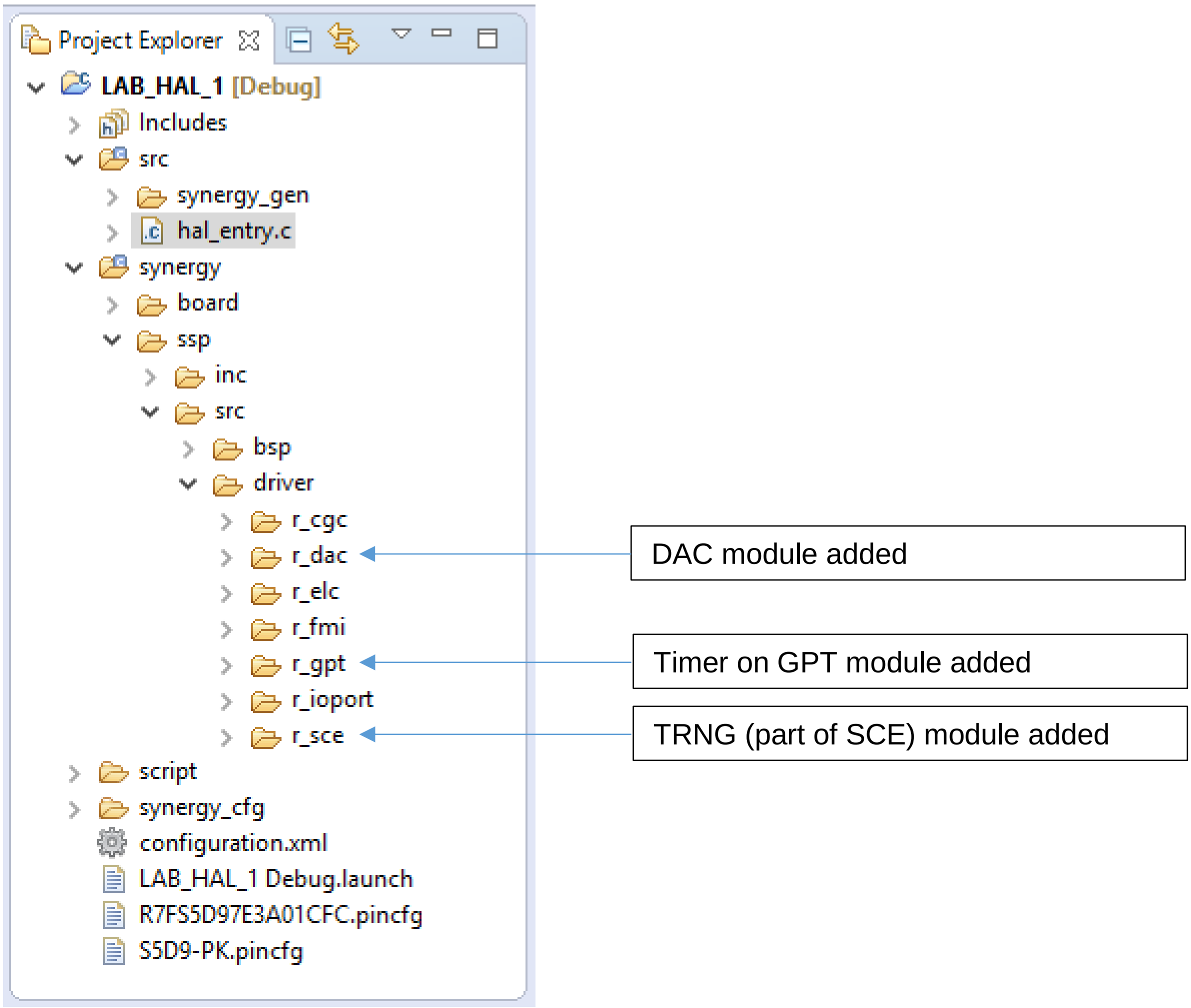
4. Verify that the DAC pin for DAC channel 0 is enabled via the pins tab in the Synergy Configuration. By default this should be enabled as we specified BSP as the template.



5. With the changes made, generate the project content:



When the project is generated e2 Studio extracts the required module files from the SSP and copies these to the application. This can be seen be looking at the Project Explorer. It will be seen that DAC, GPT and TRNG files have been added to the project.



```
Project Explorer
LAB_HAL_1 [Debug]
  Includes
  src
    synergy_gen
    hal_entry.c
  synergy
    board
    ssp
      inc
      src
        bsp
        driver
          r_cgc
          r_dac
          r_elc
          r_fmi
          r_gpt
          r_ioport
          r_sce
  script
  synergy_cfg
  configuration.xml
  LAB_HAL_1 Debug.launch
  R7FS5D97E3A01CFC.pincfg
  S5D9-PK.pincfg
```

DAC module added

Timer on GPT module added

TRNG (part of SCE) module added

Section 3: Adding APIs for the DAC, Timer & TRNG modules

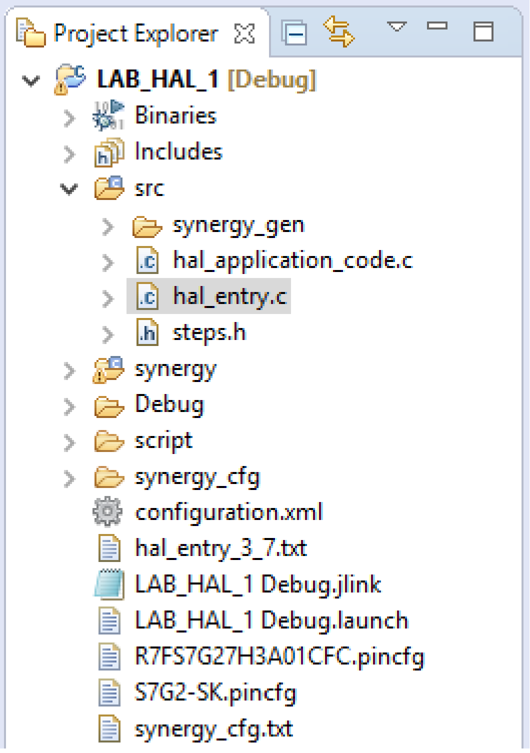
With the modules added to the project and generated, we can now write our application code using the SSP APIs.

The aim of this lab is for you to learn how to create a Synergy project and familiarize yourself with API calls of the SSP. It is not writing application code. Therefore, the application code will be provided for you and you will be responsible for adding some API calls so that you can see the benefit of the SSP.

Application code will be added to the project by you changing a "`STEP_SELECT`" number in a provided header file "step_select.h"

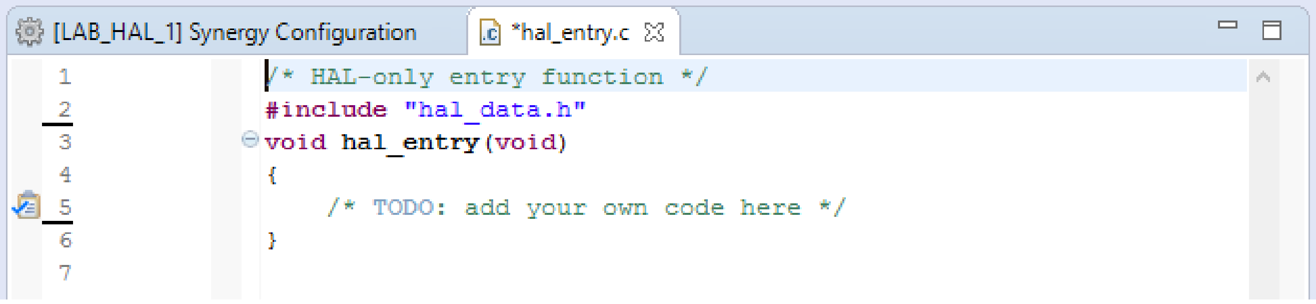
Therefore:

1. Copy the 2 files **`hal_application_code.c`** and **`step_select.h`**, provided on the USB memory stick, to the **`src`** folder.



The provided application code is contained within the file **`hal_application_code.c`**, within the function **`hal_application_code()`**. A call to this function needs to be added to the **`hal_entry.c`** file.

2. Open **`hal_entry.c`** file by double clicking on it.

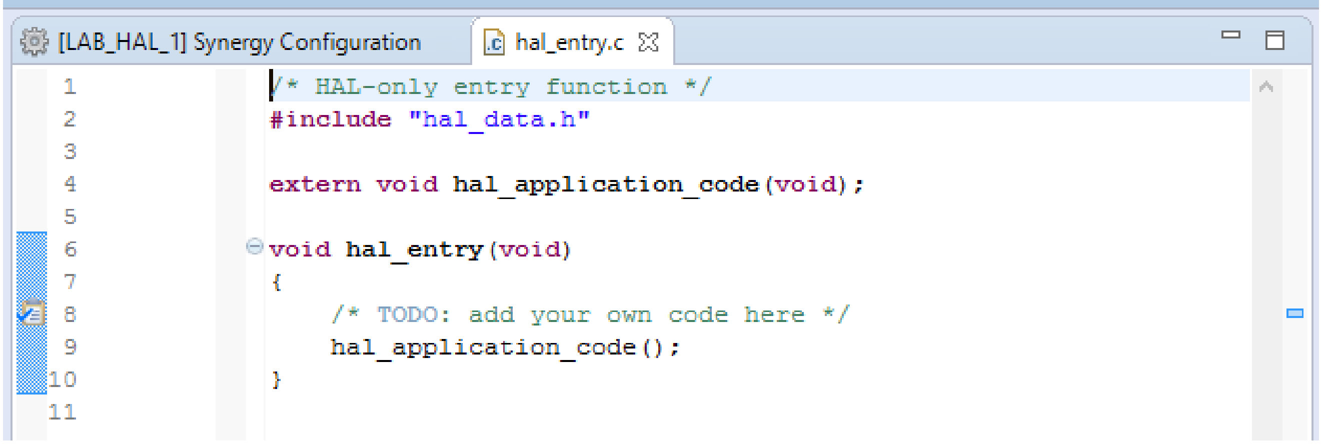


```
/* HAL-only entry function */
#include "hal_data.h"
void hal_entry(void)
{
    /* TODO: add your own code here */
}
```

3. Add the 2 lines of code

```
extern void hal_application_code(void);
hal_application_code();
```

so that **`hal_entry.c`** looks like this:



```
/* HAL-only entry function */
#include "hal_data.h"

extern void hal_application_code(void);

void hal_entry(void)
{
    /* TODO: add your own code here */
    hal_application_code();
}
```

To use a module in the SSP, the module has to be opened.

To familiarize you with the process you will now add the API that will open the GPT, DAC and TRNG modules.

4. Open the header file "**`step_select.h`**" file and set the **`"STEP_SELECT"`** macro to 1 and save the file.

```
#define STEP_SELECT   (1)
```

NOTE: Saving the file is important. Doing so will refresh the editor window so you will see the relevant parts of code to edit. Please remember to save the file when you change the number.

5. Open the file **`hal_application_code.c`**
6. Where indicated, look for the comments `/*** ENTER API CALL HERE TO OPEN THE XXX ***/`

   add the API to open the modules.

The format of the open APIs is the same for all modules

<*module name* >.p_api -> open (<*module name*>.p_ctrl, <*module name*>.p_cfg);

For example:

```
g_gpt0.p_api->open(g_gpt0.p_ctrl, g_gpt0.p_cfg);

g_dac0.p_api->open(g_dac0.p_ctrl, g_dac0.p_cfg);
```

The trng is an example of one module that requires 2 open calls. One to open the SCE (Secure Crypto Engine) and the other to open the TRNG itself.

```
g_sce.p_api->open(g_sce.p_ctrl, g_sce.p_cfg);

g_sce_trng.p_api->open(g_sce_trng.p_ctrl, g_sce_trng.p_cfg);
```

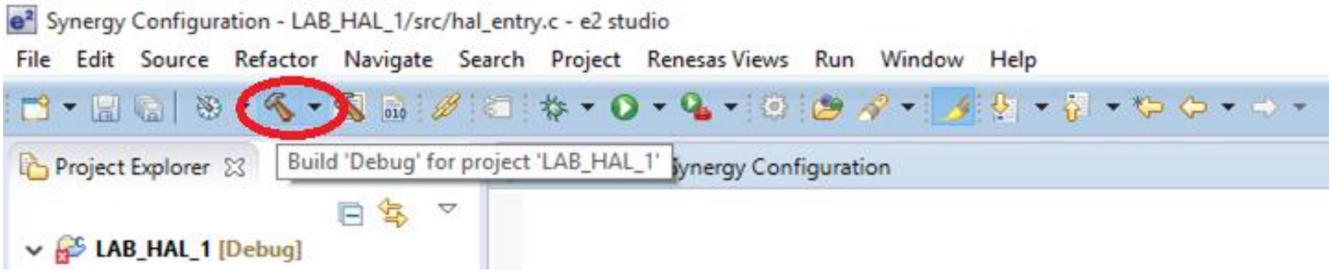
It should be noted that all SSP API return an error code of type ssp_err_t. If the module opens correctly the error code will be SSP_SUCCESS. It is good coding practice to check the API return calls. Therefore, when checking the error code our application code could look like this:

```
ssp_err = g_gpt0.p_api->open(g_gpt0.p_ctrl, g_gpt0.p_cfg);

ssp_err = g_dac0.p_api->open(g_dac0.p_ctrl, g_dac0.p_cfg);

ssp_err = g_sce.p_api->open(g_sce.p_ctrl, g_sce.p_cfg);

ssp_err = g_sce_trng.p_api->open(g_sce_trng.p_ctrl, g_sce_trng.p_cfg);
```

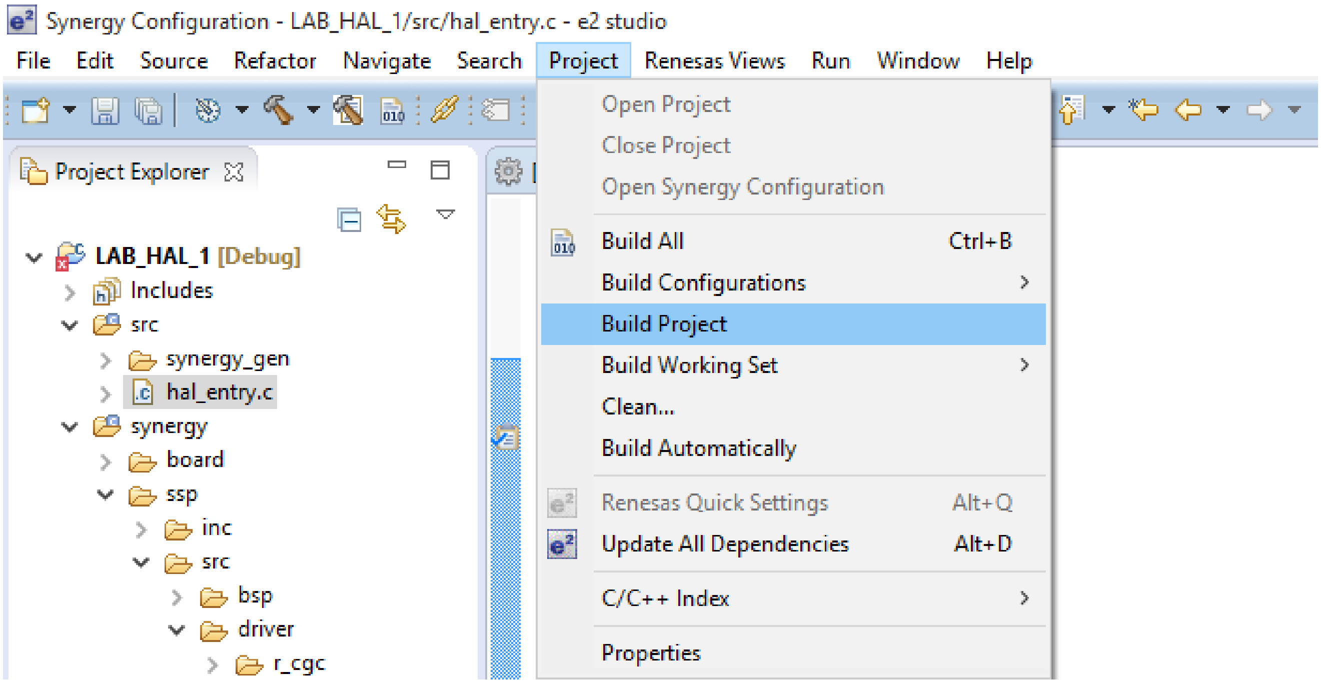
Add the 4 open calls to your code with the appropriate error checking

Build your application

Building the code can be done in several ways. Using the Hammer icon or the menu option.


Synergy Configuration - LAB_HAL_1/src/hal_entry.c - e2 studio
File Edit Source Refactor Navigate Search Project Renesas Views Run Window Help
Project Explorer
Build 'Debug' for project 'LAB_HAL_1'
Synergy Configuration
LAB_HAL_1 [Debug]

Form the menu option Project >Build Project



Building the project should result in 2 errors being generated, displayed in the Console window

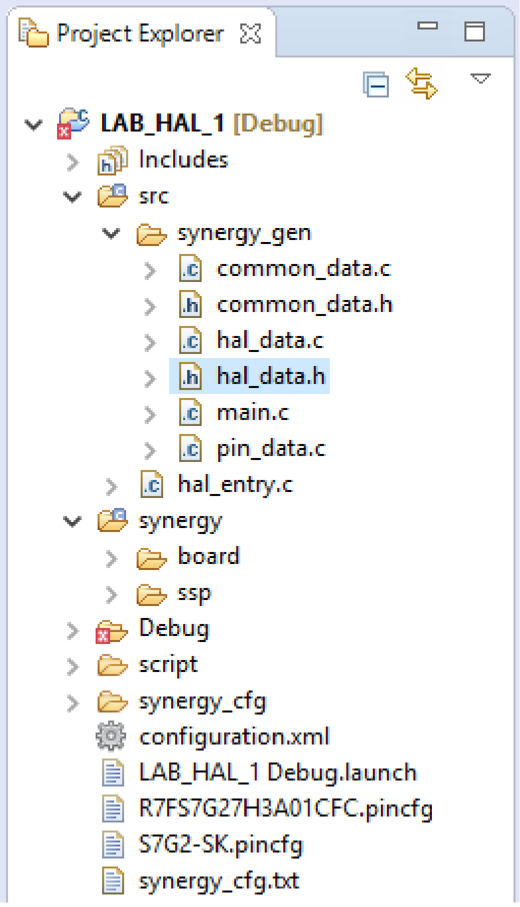
```
'Building target: LAB_HAL_1.elf'
'Invoking: Cross ARM C Linker'
arm-none-eabi-gcc @"LAB_HAL_1.elf.in"
./src/synergy_gen/hal_data.o:(.rodata.g_gpt0_cfg+0x14): undefined reference to
`cb_gpt0'
collect2.exe: error: ld returned 1 exit status
make: *** [LAB_HAL_1.elf] Error 1
makefile:68: recipe for target 'LAB_HAL_1.elf' failed

13:23:37 Build Failed. 2 errors, 0 warnings. (took 7s.204ms)
```

This error is because a callback function has been declared in the Synergy configuration (section 2.3 – configuring the GPT timer) but has not been written in the application code.

Therefore, a callback function **`cb_gpt0`** has to be written.

7. To do this change the **`”STEP_SELECT”`** macro to 2 and save the file.

If you were writing the callback function yourself, the format of the callback function needs to be known. Opening the header file **hal_data.h**, which is included at the top of **hal_entry.c** details the function prototype. The file can be located in the **scr\synergy_gen** folder.



```
Project Explorer
LAB_HAL_1 [Debug]
  Includes
  src
    synergy_gen
      common_data.c
      common_data.h
      hal_data.c
      hal_data.h
      main.c
      pin_data.c
    hal_entry.c
  synergy
    board
    ssp
  Debug
  script
  synergy_cfg
  configuration.xml
  LAB_HAL_1 Debug.launch
  R7FS7G27H3A01CFC.pincfg
  S7G2-SK.pincfg
  synergy_cfg.txt
```

This file details the function prototype. In this case:

```
void cb_gpt0(timer_callback_args_t * p_args)
```

8. Building the application should now result in no errors, no warnings.

Once a module has been opened it can be used.

To familiarize you with the process you will now add the API that will start the GPT running, read data from the TRNG and write data to the DAC.

9. To do this change the **”STEP_SELECT”** macro to 3 and save the file.

10. Where indicated add the API to start the GPT

```
ssp_err = g_gpt0.p_api->start(g_gpt0.p_ctrl);
```

11. Where indicated add the API to read a value from the TRNG

```
ssp_err = g_sce_trng.p_api->read(g_sce_trng.p_ctrl, &g_trng_result,
TRNG_NUMBERS_TO_READ);
```

12. Where indicated add the API to start write a value to the DAC

```
ssp_err = g_dac0.p_api->write(g_dac0.p_ctrl, (dac_size_t)g_trng_result);
```

13. Build your application

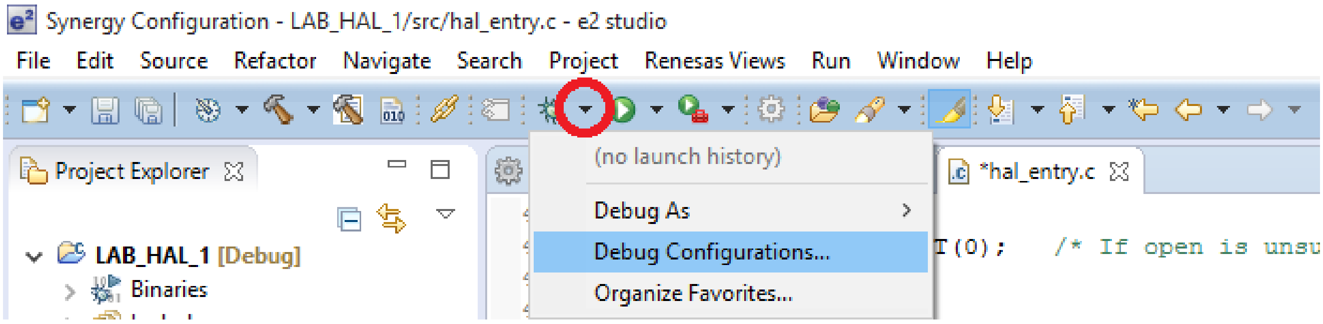
Section 4: Debugging the application

1. Debug your application

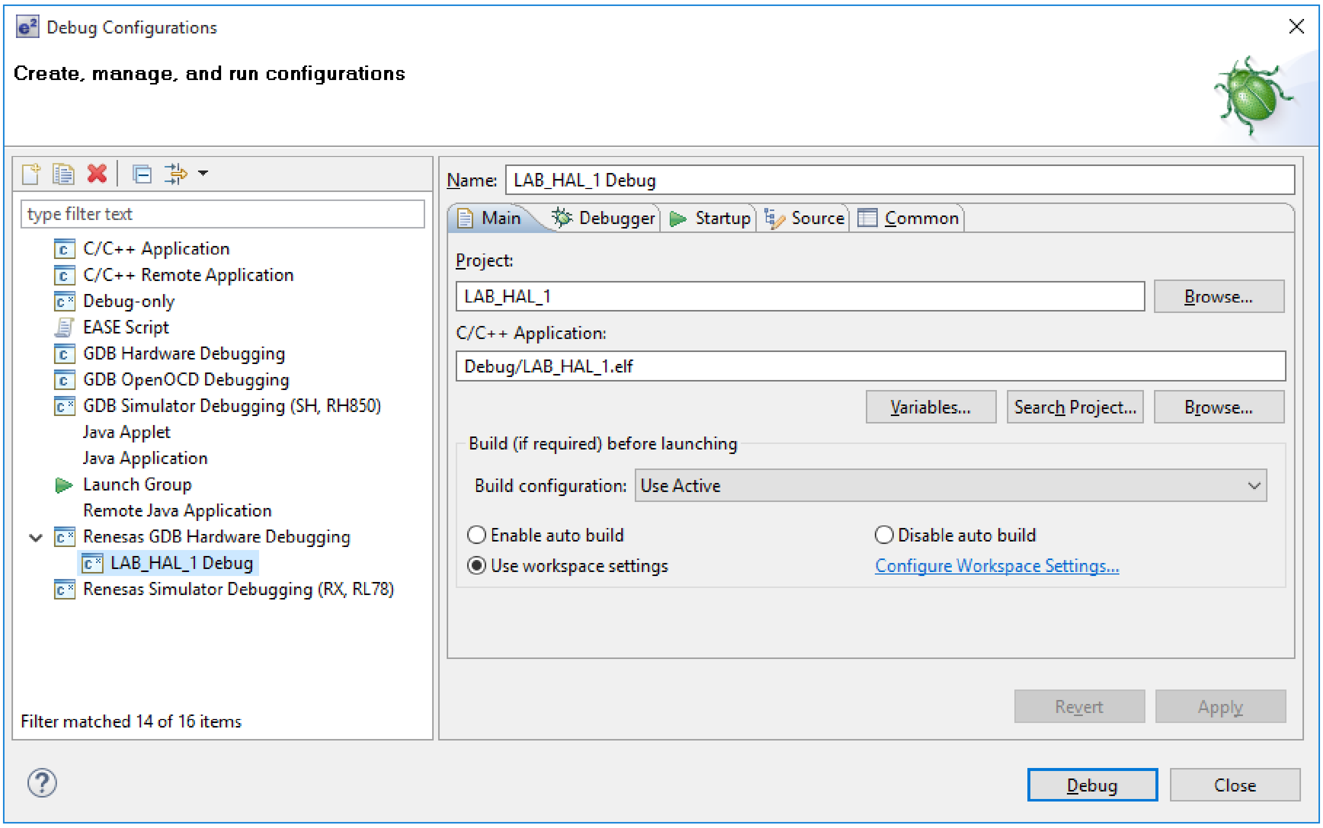
   The first time you debug your application a debug configuration has to be selected.

2. Expand the debug dropdown and select Debug Configurations…

   At first the is no launch history



3. Expand Renesas GDB Hardware Debugging and select the project LAB_HAL_1 debug


Debug Configurations

Create, manage, and run configurations

type filter text

- C/C++ Application
- C/C++ Remote Application
- Debug-only
- EASE Script
- GDB Hardware Debugging
- GDB OpenOCD Debugging
- GDB Simulator Debugging (SH, RH850)
- Java Applet
- Java Application
- Launch Group
- Remote Java Application
- Renesas GDB Hardware Debugging
  - LAB_HAL_1 Debug
- Renesas Simulator Debugging (RX, RL78)

Filter matched 14 of 16 items

Name: LAB_HAL_1 Debug

Main | Debugger | Startup | Source | Common

Project:
LAB_HAL_1

C/C++ Application:
Debug/LAB_HAL_1.elf

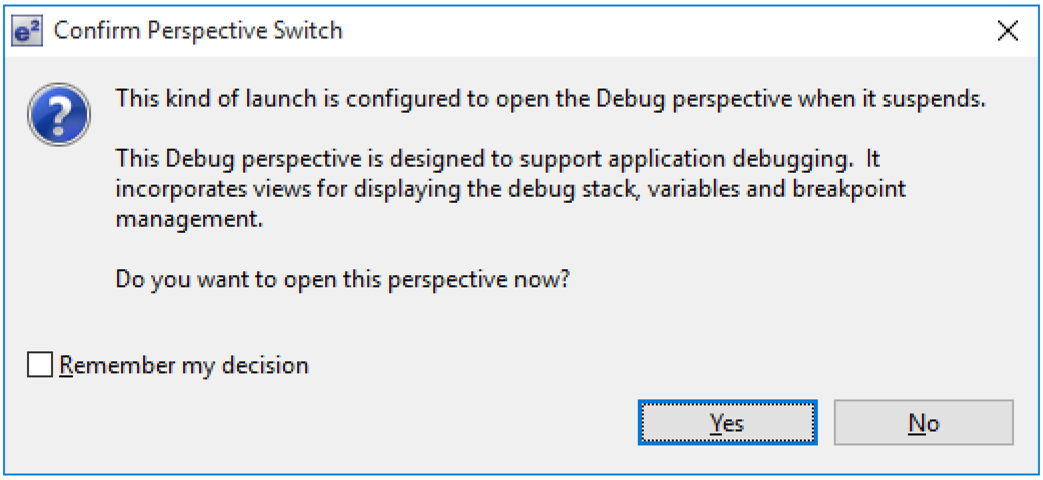
Build (if required) before launching

Build configuration: Use Active

○ Enable auto build ○ Disable auto build
● Use workspace settings Configure Workspace Settings...

At this point you can specify the debug files, debugger, and debugging behavior from the various options. However, the default settings can be used so click Debug.

When asked change perspective to the Debug perspective.



Confirm Perspective Switch

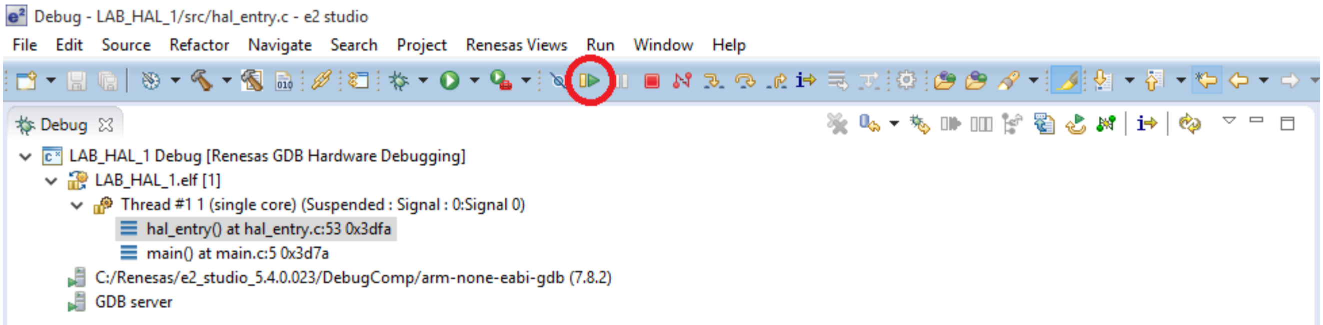
This kind of launch is configured to open the Debug perspective when it suspends.

This Debug perspective is designed to support application debugging. It incorporates views for displaying the debug stack, variables and breakpoint management.

Do you want to open this perspective now?

□ Remember my decision

4. Run the code and execution will halt at main().

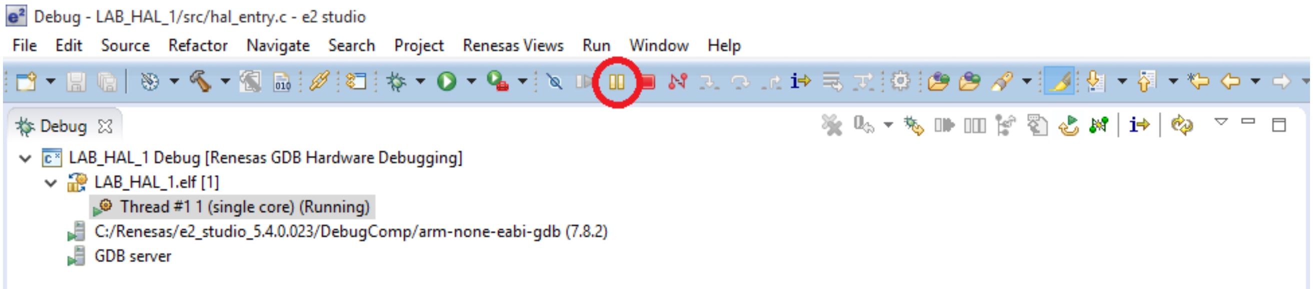


```
LAB_HAL_1 Debug [Renesas GDB Hardware Debugging]
  LAB_HAL_1.elf [1]
    Thread #1 1 (single core) (Suspended : Signal : 0:Signal 0)
      hal_entry() at hal_entry.c:53 0x3dfa
      main() at main.c:5 0x3d7a
  C:/Renesas/e2_studio_5.4.0.023/DebugComp/arm-none-eabi-gdb (7.8.2)
  GDB server
```

Pressing run again will begin execution of the application code.

Pressing pause will most likely be in the while(1) loop, whilst waiting for the g_gpt0_flag to be set.

```
while(1)
{
    #if (STEP_SELECT > 2)
    /* Wait for the g_gpt0_flag to be set.
     * This will be set in the g_gpt0 callback function that will be called
     * every second
     */
      while(0 == g_gpt0_flag);
```

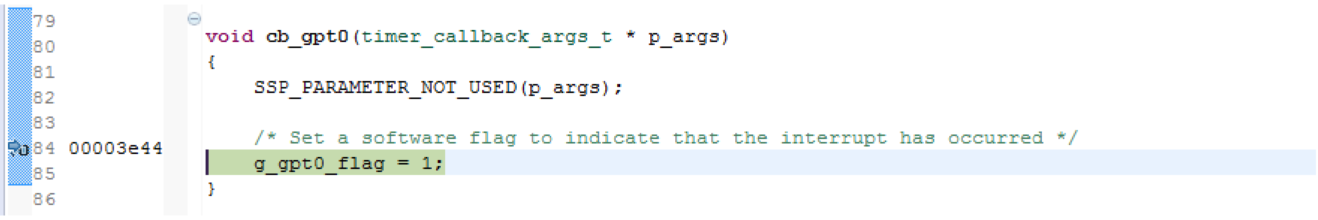


```
LAB_HAL_1 Debug [Renesas GDB Hardware Debugging]
  LAB_HAL_1.elf [1]
    Thread #1 1 (single core) (Running)
  C:/Renesas/e2_studio_5.4.0.023/DebugComp/arm-none-eabi-gdb (7.8.2)
  GDB server
```

Several things can be done to see if the code is executing as expected.

5. First, set a breakpoint on the gpt callback function.

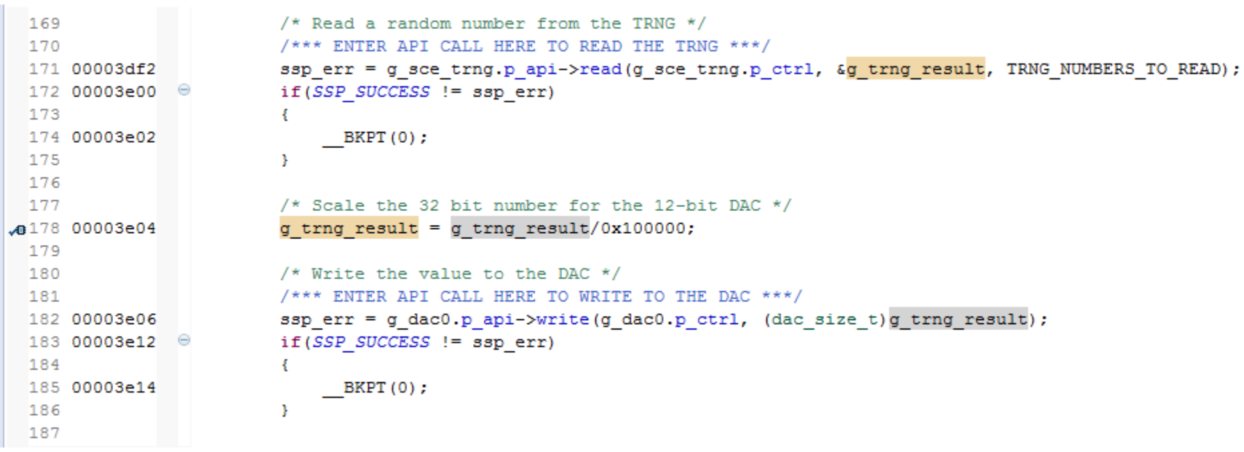
   Double click in the gutter next to the line of code `g_gpt0_flag = 1;` to set the breakpoint.



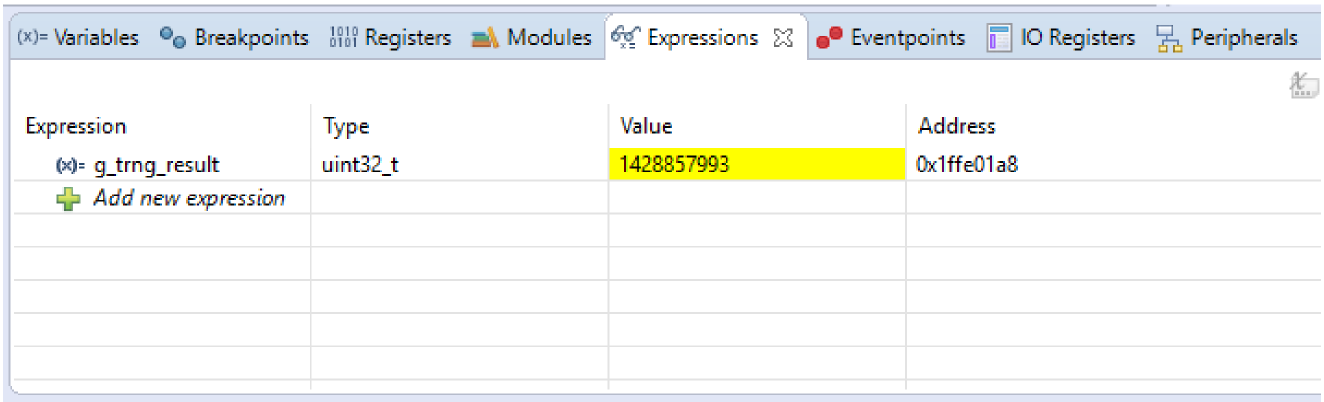
```
          void cb_gpt0(timer_callback_args_t * p_args)
          {
              SSP_PARAMETER_NOT_USED(p_args);

              /* Set a software flag to indicate that the interrupt has occurred */
00003e44      g_gpt0_flag = 1;
          }
```

Now running the code execution will stop every second. This will indicate that the GPT is running and that the interrupt is being generated.

6. Second, add the global variable `g_trng_result` to the Expressions window and set a breakpoint on `g_trng_result = g_trng_result/0x100000;` and remove the previous breakpoint.



```
                /* Read a random number from the TRNG */
                /*** ENTER API CALL HERE TO READ THE TRNG ***/
00003df2        ssp_err = g_sce_trng.p_api->read(g_sce_trng.p_ctrl, &g_trng_result, TRNG_NUMBERS_TO_READ);
00003e00        if(SSP_SUCCESS != ssp_err)
                {
00003e02            __BKPT(0);
                }

                /* Scale the 32 bit number for the 12-bit DAC */
00003e04        g_trng_result = g_trng_result/0x100000;

                /* Write the value to the DAC */
                /*** ENTER API CALL HERE TO WRITE TO THE DAC ***/
00003e06        ssp_err = g_dac0.p_api->write(g_dac0.p_ctrl, (dac_size_t)g_trng_result);
00003e12        if(SSP_SUCCESS != ssp_err)
                {
00003e14            __BKPT(0);
                }
```

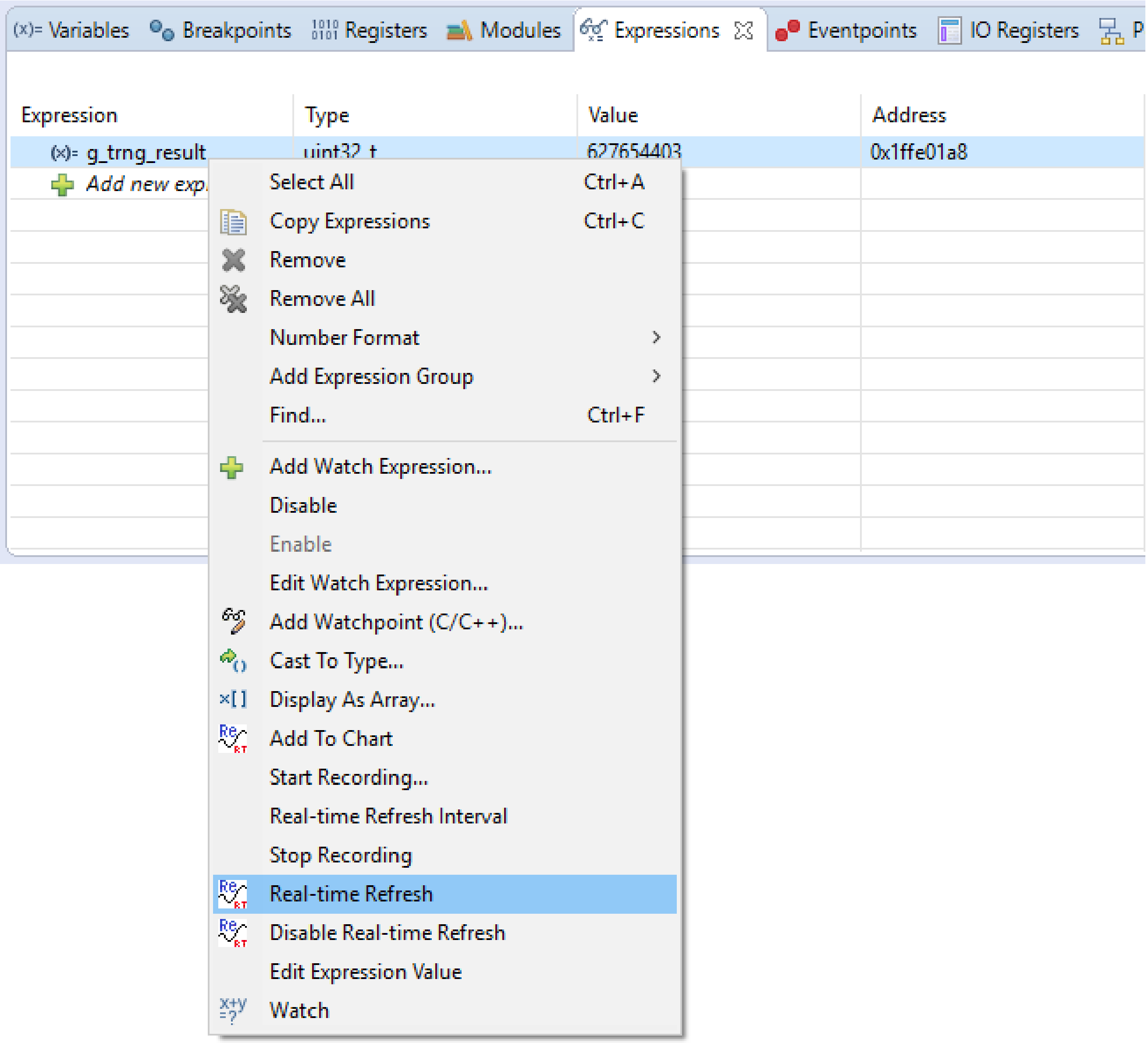


Expression	Type	Value	Address
(x)= g_trng_result	uint32_t	1428857993	0x1ffe01a8
Add new expression			

Again, running the code execution will stop every second. You will be able to observe the g_trng_result value being displayed in the expressions window.

7. Change the Expression to a real-time view

   Open the context menu by right-clicking on g_trng_result



Expression	Type	Value	Address
(x)= g_trng_result	uint32_t	627654403	0x1ffe01a8
Add new exp			

8. Remove the breakpoint.

9. Run the code.

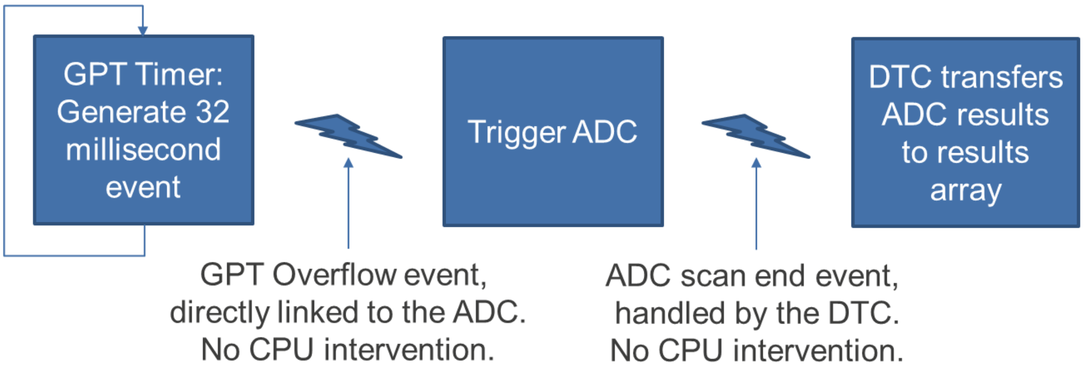
10. You should see the g_trng_result changing every second.

Section 5: Adding the ADC, Timer & Transfer modules

In this section we will add modules that will periodically trigger the ADC and transfer the ADC result to an array.

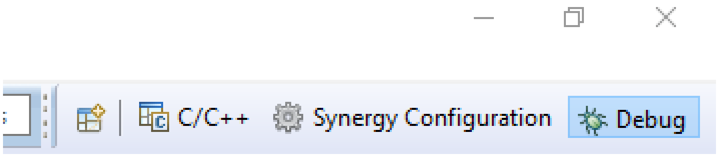
The Timer will be configured to generate a 32 microsecond interrupt. The timer event will be linked, via the ELC (Event Link Controller), to trigger the ADC. The ADC Scan End interrupt will be handled by the DTC (Data transfer Controller) which will transfer the result from the ADC to an array in memory.

The diagram below shows the basic application flow of the application.

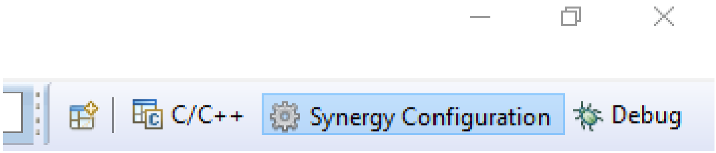

GPT Timer: Generate 32 millisecond event
Trigger ADC
DTC transfers ADC results to results array
GPT Overflow event, directly linked to the ADC. No CPU intervention.
ADC scan end event, handled by the DTC. No CPU intervention.

1. Switch from the Debug perspective to the Synergy Configuration perspective.

   Perspectives can be easily changes via the buttons in the top right of e2 Studio.



Debug Perspective selected



Synergy Configuration Perspective selected

2. Open the Synergy Configuration tab.

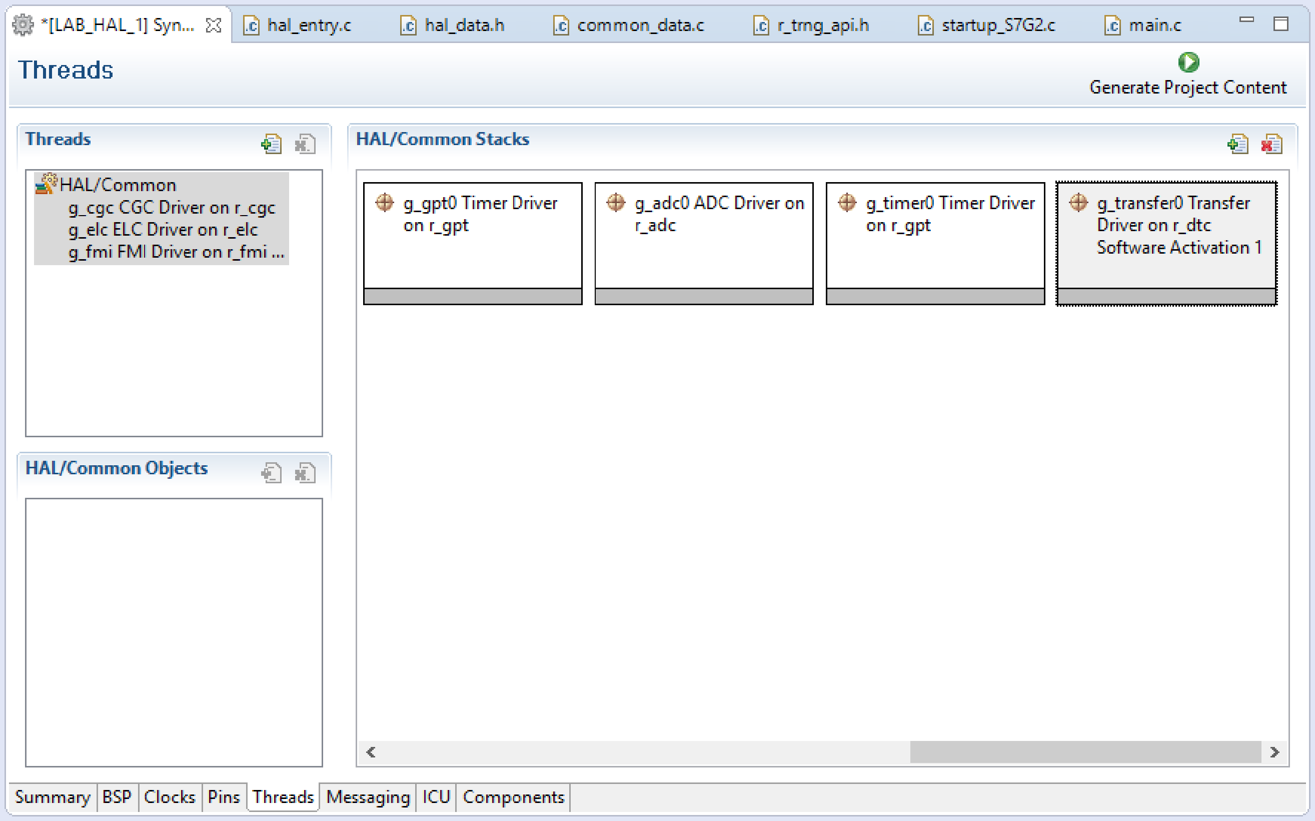
   If you have closed this tab, it can be reopened by double clicking on the configuration.xml file in the Project Explorer window.

3. As you did in section 2, add to the HAL/Common thread, the modules:

   Driver > Analog > ADC Driver on r_adc

   Driver > Timers > Timer Driver on r_gpt

   Driver > Transfer > Transfer Driver on r_dtc



HAL/Common
g_cgc CGC Driver on r_cgc
g_elc ELC Driver on r_elc
g_fmi FMI Driver on r_fmi ...

g_gpt0 Timer Driver on r_gpt

g_adc0 ADC Driver on r_adc

g_timer0 Timer Driver on r_gpt

g_transfer0 Transfer Driver on r_dtc Software Activation 1

4. Make the following changes to the ADC, Timer and Transfer modules.

g_adc0 ADC Driver on r_adc Modifications	
Resolution	12-Bit
Channel 0	Use in Normal/Group A
Normal/Group A Trigger	ELC Event
Scan End Interrupt Priority	Priority 6 (CM4: valid, CM0+: invalid)

g_timer0 Timer Driver on r_gpt Modifications	
Name	g_gpt1
Channel	1
Periodic Value	31250
Periodic Unit	Microseconds

g_transfer0 Transfer Driver on r_rtc Software Activation 1 Modifications	
Mode	Repeat
Destination Address Mode	Incremented
Repeat Area	Destination
Number of transfers	4
Activation Source (Must Enable IRQ)	Event ADC0 SCAN END

5. With the changes made, generate the project content:

Section 6: Adding APIs for the ADC, Timer & Transfer modules

1. Change the **”STEP_SELECT”** macro to 4 and save the file.

This will add the code that opens and configures the ADC, opens and configures the DTC and opens and starts the GPT1.

2. Where indicated, add the API call that creates an ELC link to link the triggering of the ADC to the overflow of the GPT1.

```
ssp_err = g_elc.p_api->linkSet(ELC_PERIPHERAL_ADC0, ELC_EVENT_GPT1_COUNTER_OVERFLOW);
```

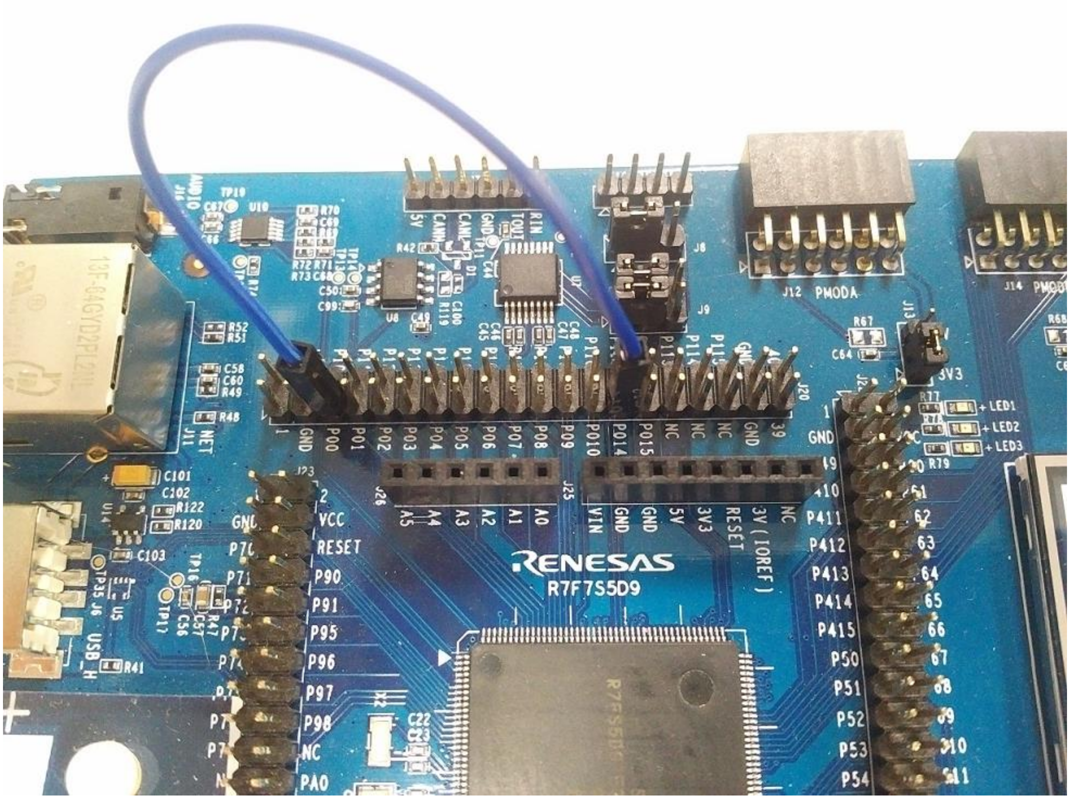
3. Build your application.

In the application the ADC will convert the DAC output value. For this to happen the DAC output pin and the ADC input pins have to be connected.

4. With the provided wire, connect pin P0_00 (AN000) to P0_14 (DAC0)

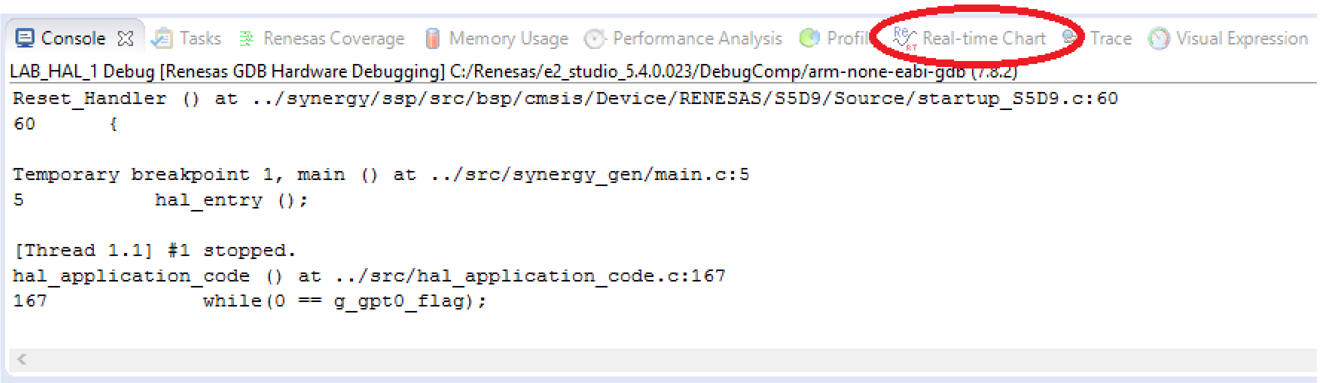
   These pins can be located on connector J20 of the S7G2 SK / S5D9 PK board.

   Connect pin 5 to pin 27. The pins are also labeled P0_00 & P0_14



5. Debug your application.

6. Navigate to the Real-time Chart window

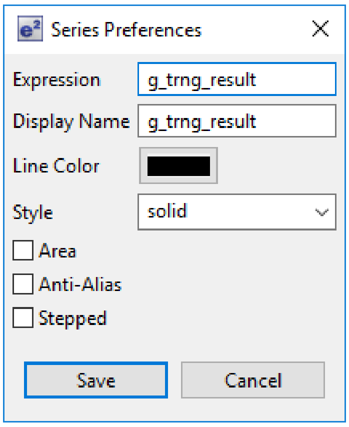


LAB_HAL_1 Debug [Renesas GDB Hardware Debugging] C:/Renesas/e2_studio_5.4.0.023/DebugComp/arm-none-eabi-gdb (7.8.2)

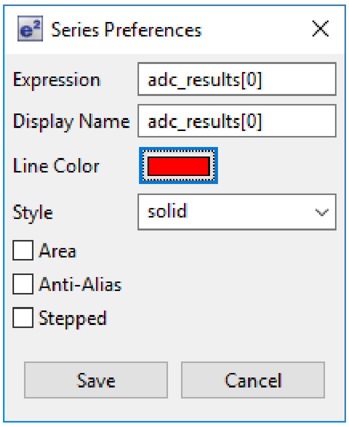
```
Reset_Handler () at ../synergy/ssp/src/bsp/cmsis/Device/RENESAS/S5D9/Source/startup_S5D9.c:60
60      {

Temporary breakpoint 1, main () at ../src/synergy_gen/main.c:5
5           hal_entry ();

[Thread 1.1] #1 stopped.
hal_application_code () at ../src/hal_application_code.c:167
167             while(0 == g_gpt0_flag);
```

and add 2 series (right click in the window to open the menu)

Add the series `g_trng_result` and `adc_results[0]`

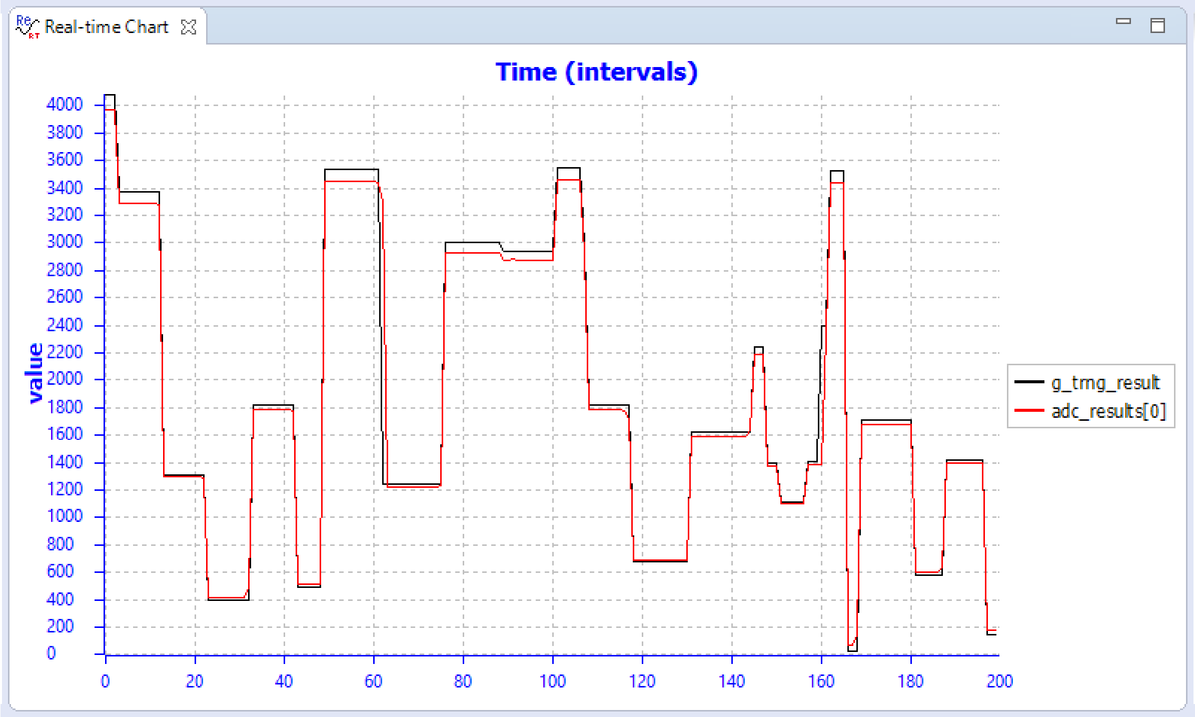


Expression: g_trng_result
Display Name: g_trng_result
Line Color: ■
Style: solid
□ Area
□ Anti-Alias
□ Stepped



Expression: adc_results[0]
Display Name: adc_results[0]
Line Color: 🟥
Style: solid
□ Area
□ Anti-Alias
□ Stepped

When the application runs you will hopefully see the adc_results following the g_trng_result.



Congratulations. You have finished the first lab.

You are should now be comfortable with adding modules to a Synergy project and using the API to control these modules.

We will now move onto more complex applications that will utilize the Thread-X operating system.

But before that some RTOS theory.