

## RTOS Module Synergy Lab

Device: Synergy S7G2 / S5D9

---

### Description:

The objective of this lab session is to introduce you to the creating a multi-threaded Synergy application utilizing the HAL modules, frameworks, X-Ware components and the ThreadX operating system. The operation of this lab will replicate that of the previous lab, but will add extra functionality.

The application will periodically generate a random number via the Hardware TRNG and this value will be used as the output of the DAC.

The output of the DAC will be connected to the input of the ADC. The application will 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 results will be buffered to memory and then copied to an USB memory device if inserted into the S5D9 PK USB host port.

The process of creating the application will introduce you to adding and configuring HAL modules, creating threads and thread objects via the Synergy Configuration window within E2Studio.

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 e2studio 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

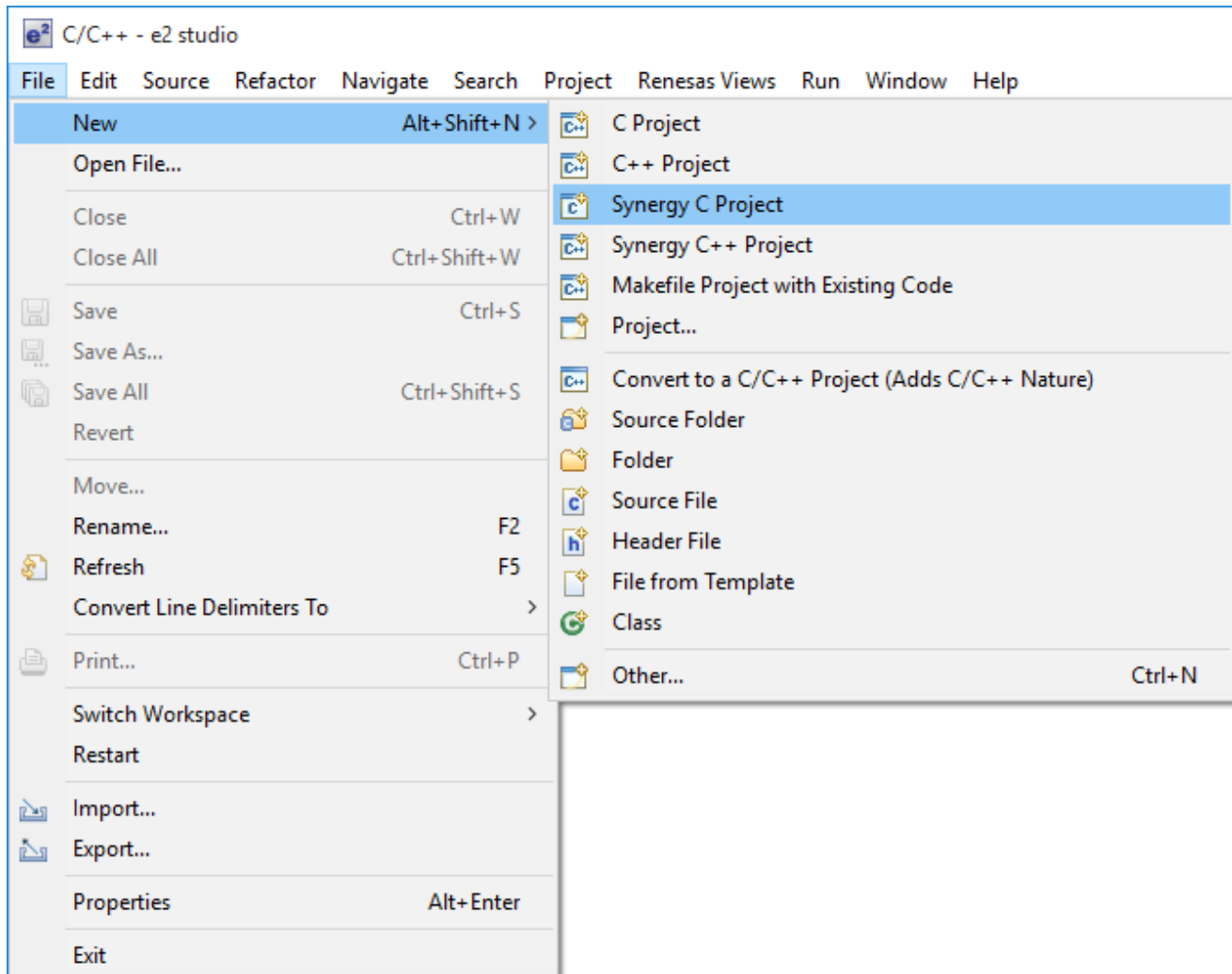
#### Time to Complete Lab

90 Minutes

Section 1: Creating the Synergy Application.....	3
Section 2: Creating a new thread – DAC, GPT & TRNG modules.....	8
Section 3.1: Adding APIs for the DAC, Timer & TRNG modules .....	14
Section 3.2: Debugging the application .....	15
Section 3.3: Removing the Timer Module.....	15
Section 4: Adding the ADC Periodic Framework .....	17
Section 5: Adding APIs for the ADC Periodic Framework.....	21
Section 6: Add USB Host functionality.....	22
Section 7: Add USB Host Code .....	27
Section 8: Inter-thread communication with a Queue .....	28

## Section 1: Creating the Synergy Application

1. In the existing e2 Studio workspace, create a new Synergy project.



2. Give the project a name. This example will use **LAB\_RTOS\_1**

You will notice that as you are creating a project within an existing workspace, as you have previously specified the license file location, this time you do not need to specify it.

e2 studio - Project Configuration (Synergy C Project)

**e2 studio - Project Configuration (Synergy C Project)**  
Specify the new project details.

**Project**

Project name: LAB\_RTOS\_1

☒ Use default location

Location: C:\synergy\workspace\LAB\_RTOS\_1 Browse...

**Toolchains**

GCC ARM Embedded

**License**

License file: [Change 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

[Visit the Apps Gallery for license file and Pack file downloads](#)

? < Back **Next >** Finish Cancel

Click Next

3. Specify the SSP version, development board, device and version of toolchain to use.
4. 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)**  
Select the board support that you require.

**Device Selection**

SSP version: 1.3.3  
Board: S5D9 PK  
Device: R7FS5D97E3A01CFC

**Select Tools**

Toolchain: GCC ARM Embedded  
Toolchain version: 4.9.3.20150529  
Debugger: J-Link ARM

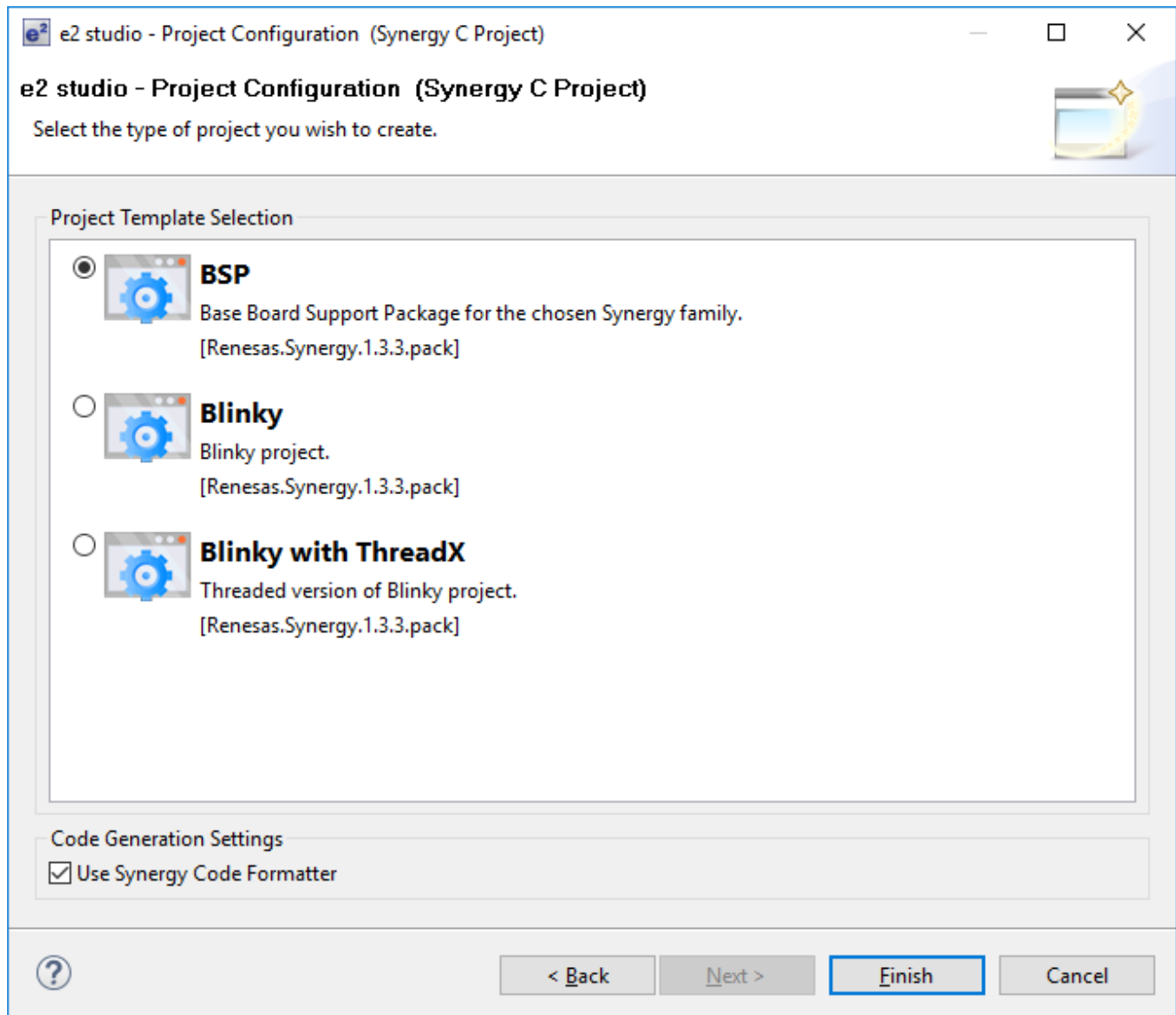
**Available Tools**

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

< Back **Next >** Finish Cancel

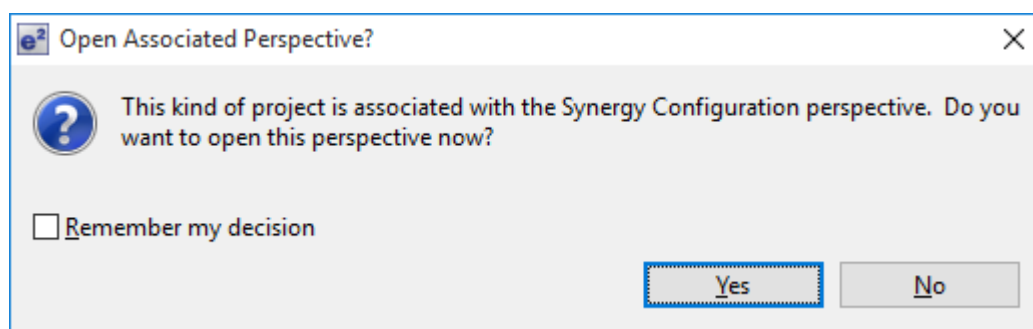
Click **Next**

5. Select the Project Template BSP

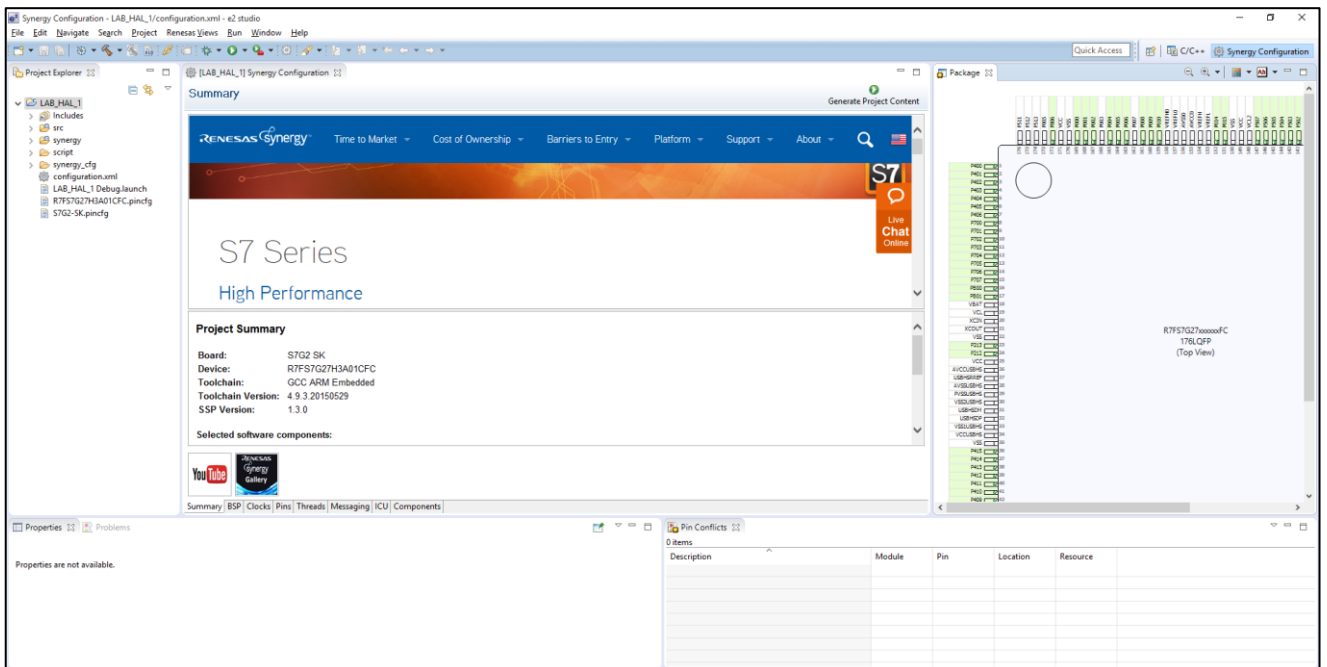
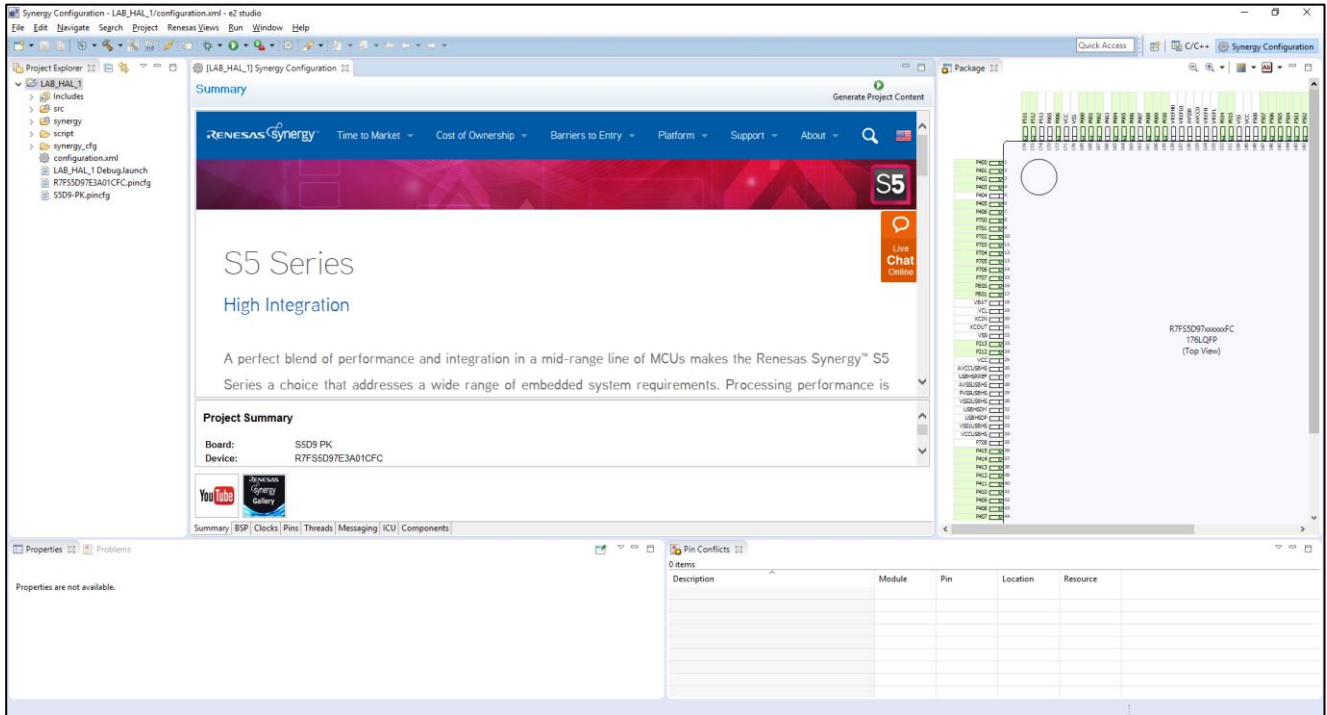


Click Finish

e2 Studio will now create the project template, extracting the relevant files from the SSP based upon the selected settings. When prompted, select Yes to open the Synergy Configuration perspective.



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



## Section 2: Creating a new thread – DAC, GPT & 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.

The first diagram is equivalent to the previous example, except that a ThreadX semaphore is used rather than the software flag.

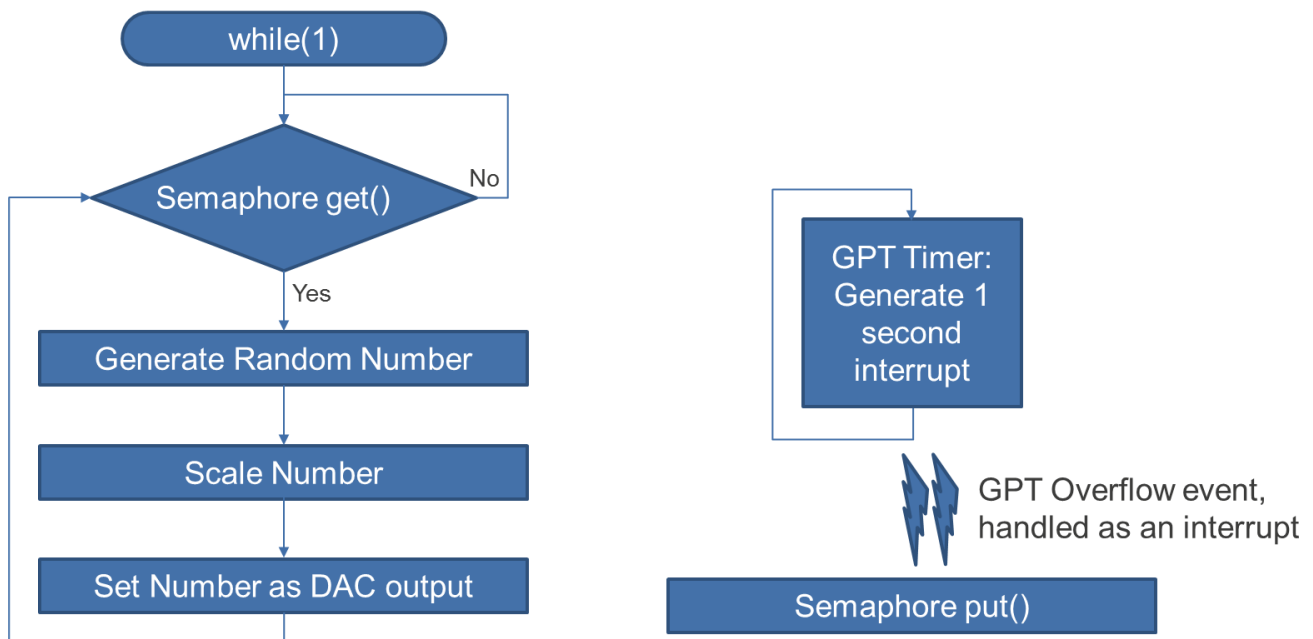


Figure 2.1

The second diagram is functionally the same, but here the internal scheduling of the RTOS is used by suspending the thread via the `tx_thread_sleep(100);` instruction.

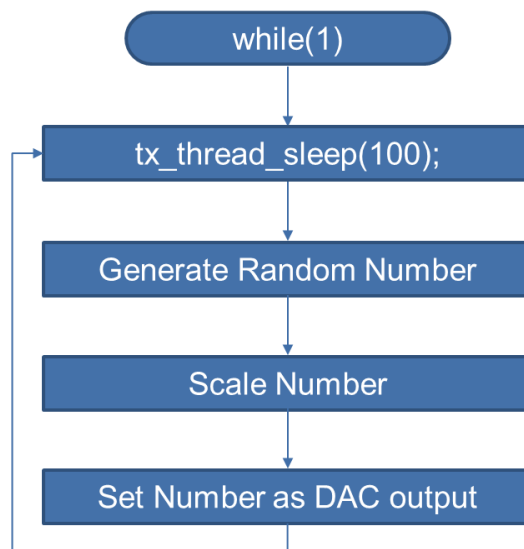


Figure 2.2

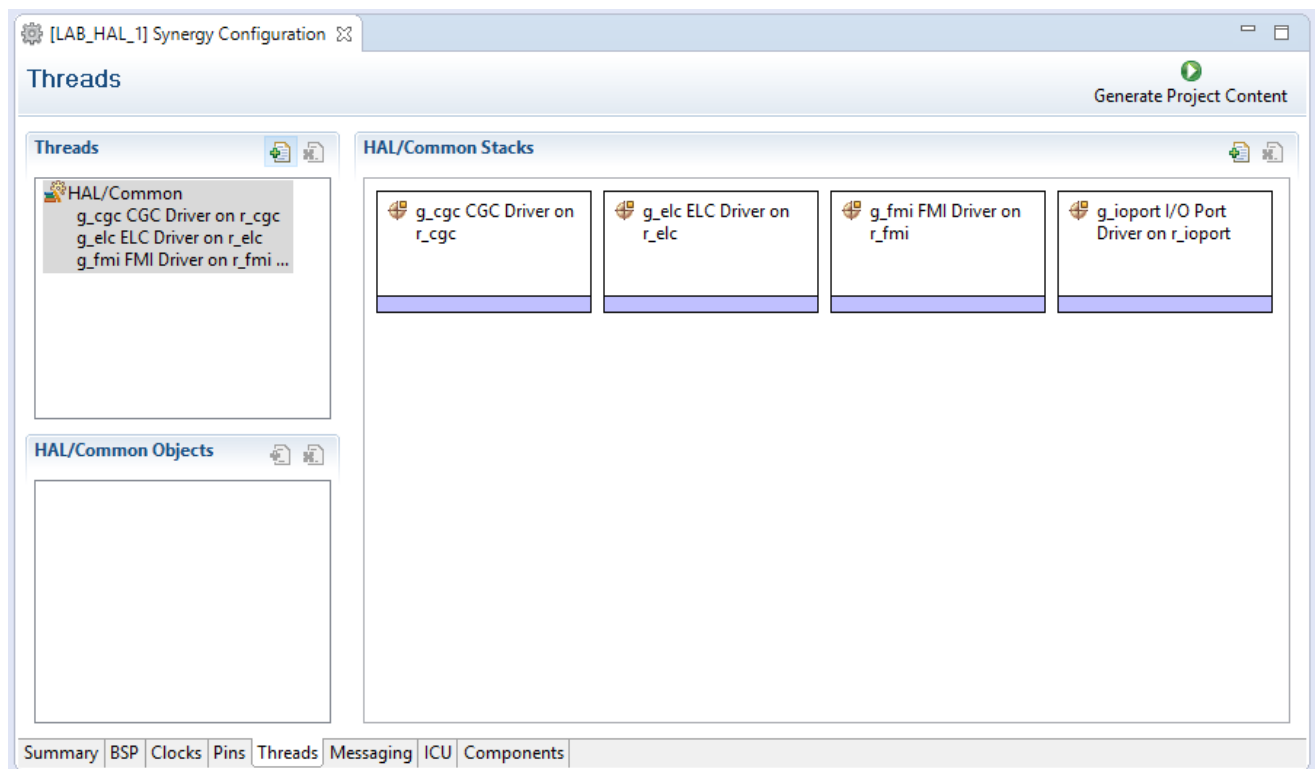


In this second case the DAC value will not be changed exactly every second. The time will be 1 second + execution time for Generate Random Generation + Scaling number + Setting number as DAC output. Whilst very minimal, this drift could be undesirable. However, if the drift is acceptable, then this would free up the GPT resource for other usage if required.

Both solutions will be implemented next.

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

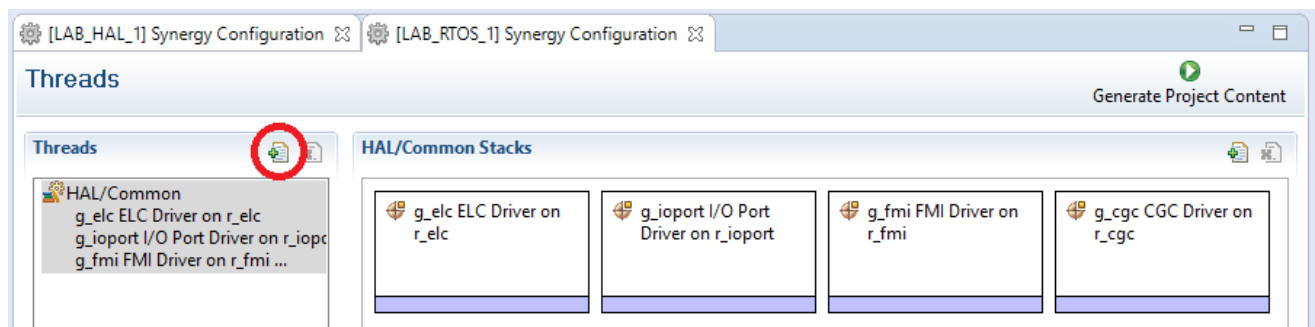
Clicking on the HAL/Common thread will see that the project has the 4 default modules already added to the project. (CGC, ELD, FMI & I/O Port)



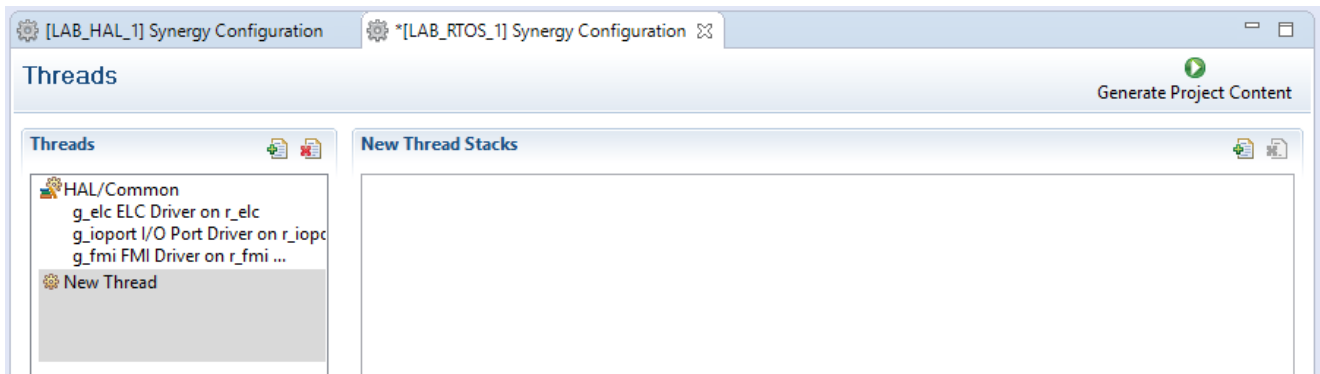
We could add the modules to this HAL/Common thread as before, but we want to create a multi-threaded application and use the ThreadX RTOS.

Therefore, add a new thread.

2. To add a new thread, click the + button in the Threads window



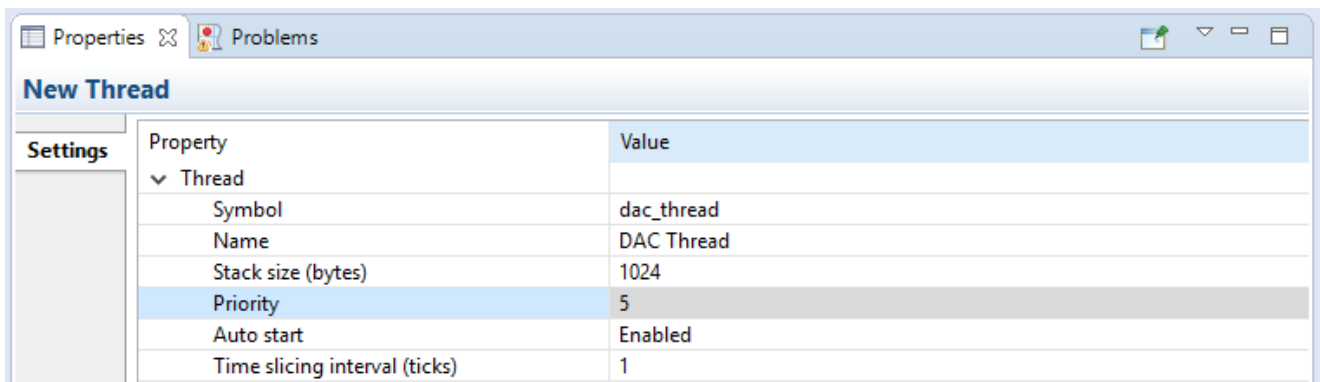
You will now see a “New Thread” created.



***If you were to now click on “Generate Project Content”, all of the ThreadX RTOS files will be automatically added to you project. There is no porting of RTOS BSP files etc, to your chosen target. It is a seamless and automated process!***

3. In the Properties window of the New Thread, give the thread a name. For example, **DAC\_Thread**  
Set the thread Priority to something other than 1.

By default the Synergy configuration tool will create all threads with a thread priority of 1 and a time slicing interval of 1. If you are not aware of this and create many threads, which all at the same priority, then you will get a system that spends its entire time task switching and not actually doing any processing.



4. To this DAC thread, 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

For this application the **r\_dac** default settings do not need to be changed.

For this application the **TRNG** module settings do not need to be changed.

For this application the **r\_gpt** module settings do need to be changed.

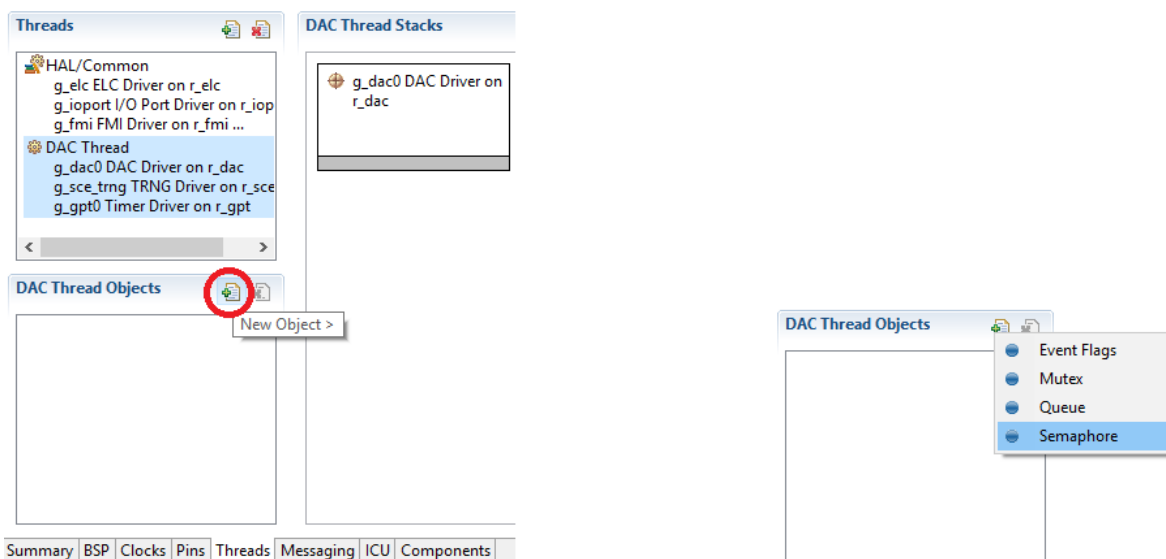
Change:

- 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: **cb\_gpt0**  
(This function is called when the GPT timer interrupt occurs)
- Enable the Interrupt by specifying an Interrupt Priority, for example: **Priority 8**

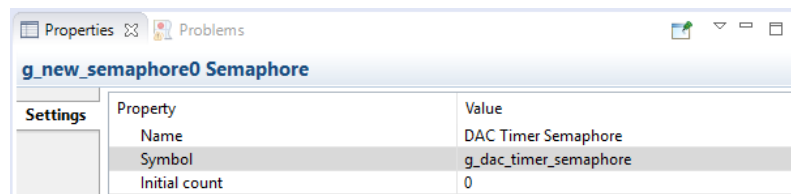
These are the same settings as the earlier lab.

If can be seen in *figure 2.1* (page 8) that the while(1) loop of the thread will wait for a semaphore. This ThreadX object is added now.

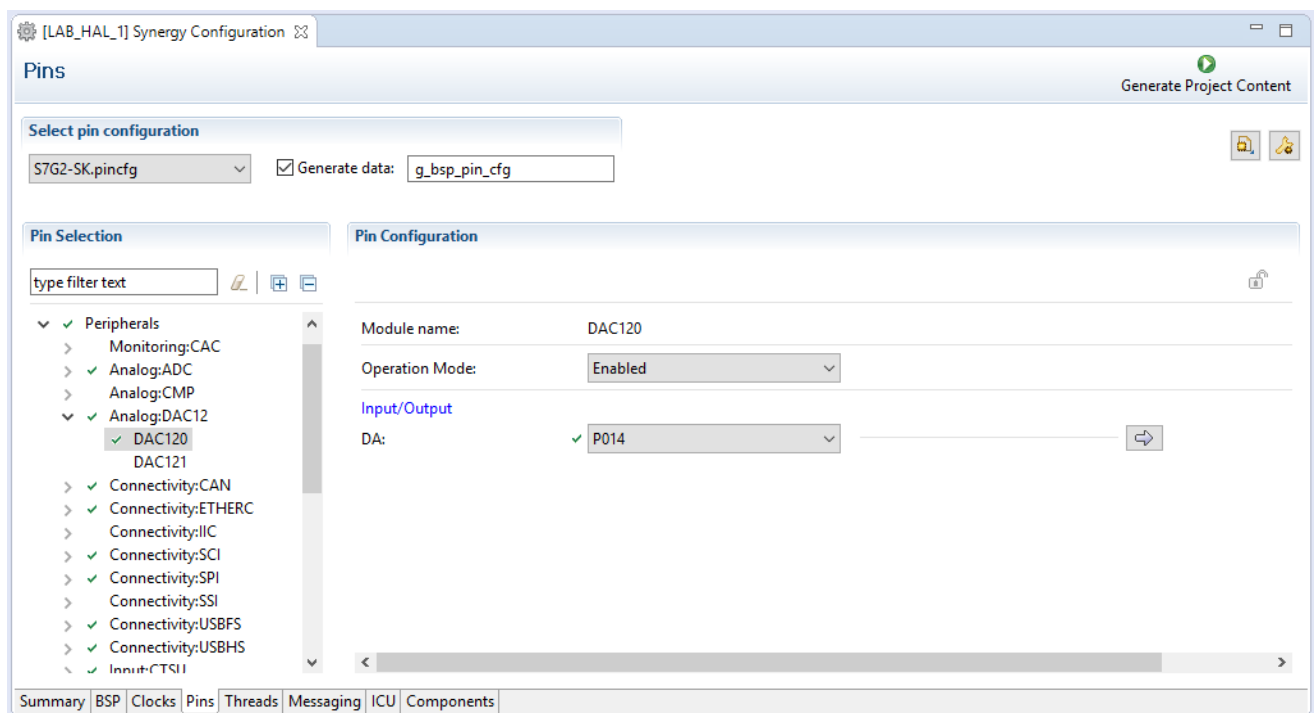
5. Add a ThreadX semaphore object to the DAC\_Thread.



- Give the semaphore a name, for example g\_dac\_timer\_semaphore



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.

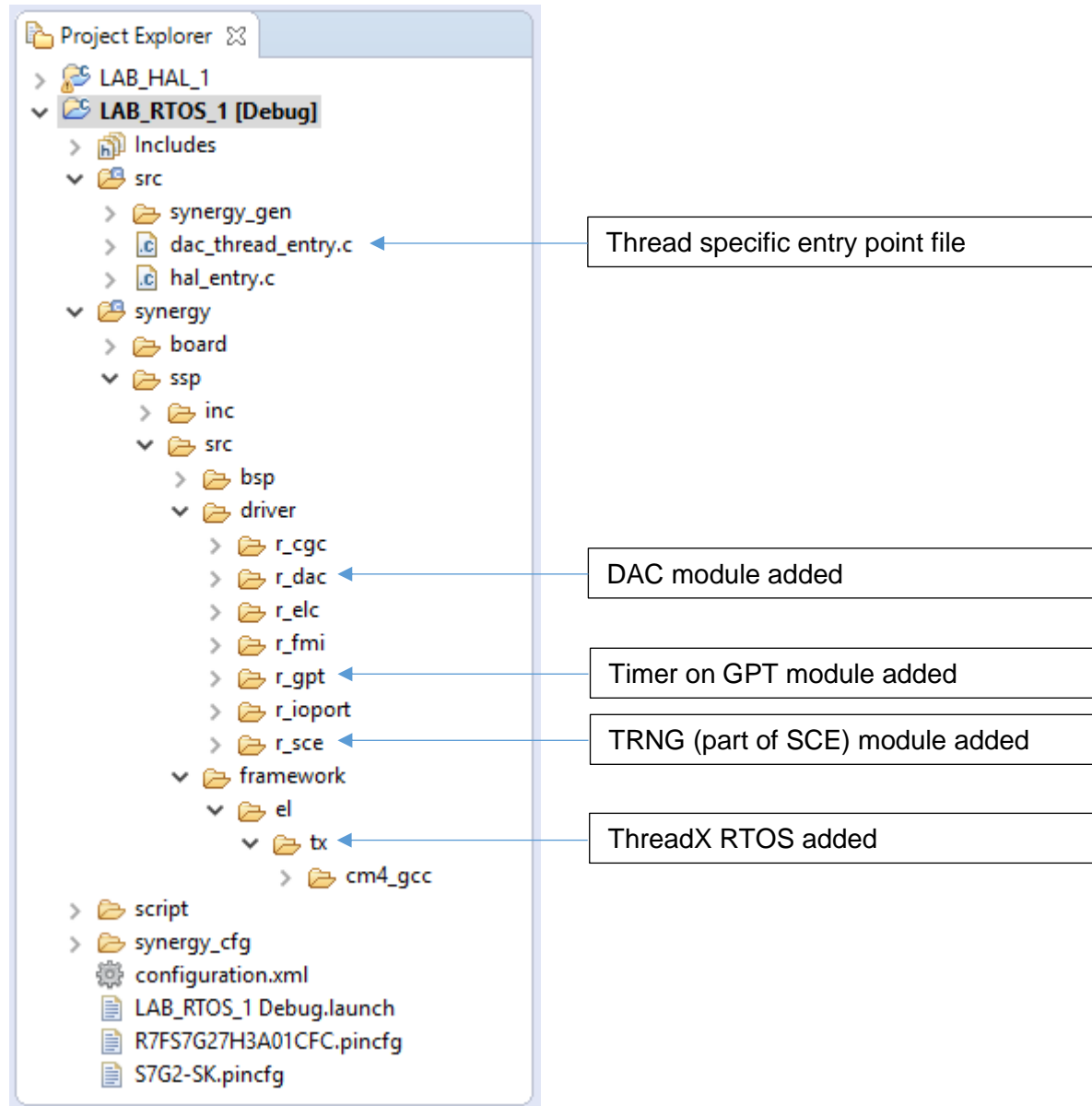


- With the changes made, generate the project content:



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 by looking at the Project Explorer. It will be seen that DAC, GPT and TRNG files have been added to the project and in addition, the Express Logic ThreadX files.

In addition to the hal\_entry.c file, a thread specific entry .c file is also created.



## Section 3.1: 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 by you changing a “step” number in a provided header file “step\_select.h”***

***Therefore:***

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

Note: At this point you will be overwriting the existing `dac_thread_entry.c` file. If prompted, confirm that you wish to overwrite it.

2. Open the header file “`step_select.h`” file and set the MACRO number to 1 and save the file.

*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.*

3. Where indicated, add the code to put the semaphore (in the timer ISR callback) and to get the semaphore (in the thread body while(1) loop)

```
tx_semaphore_put(&g_dac_timer_semaphore);
```

```
tx_semaphore_get(&g_dac_timer_semaphore, TX_WAIT_FOREVER);
```

In this lab you will not have to enter API calls to open the modules, as you have learnt about that in the previous lab.

4. Build your application

## Section 3.2: Debugging the application

1. Debug your application

You will have to select a new debug configuration for this project as before.

2. Set a breakpoint on the line of code `tx_semaphore_get()` ;
3. Run the code.

Open the Variables window and you should see the `g_trng_result` changing every time you hit the breakpoint.

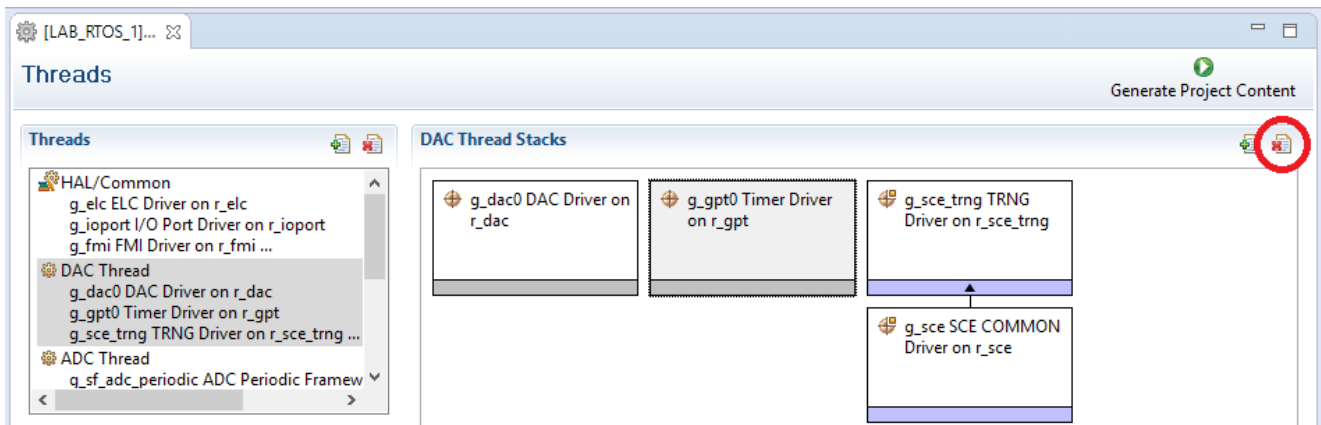
## Section 3.3: Removing the Timer Module

Now that DAC operation has been verified the project will be changed so that the 1 second delay is generated by the ThreadX RTOS, and not the hardware GPT.

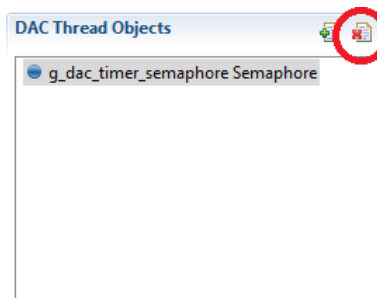
1. Navigate back to the DAC\_thread and select the `g_gpt0` Timer Driver on `r_gpt`.

*NOTE: Switch from the Debug perspective to the Synergy Configuration perspective.*

2. With it selected remove the module by clicking on the Red cross.



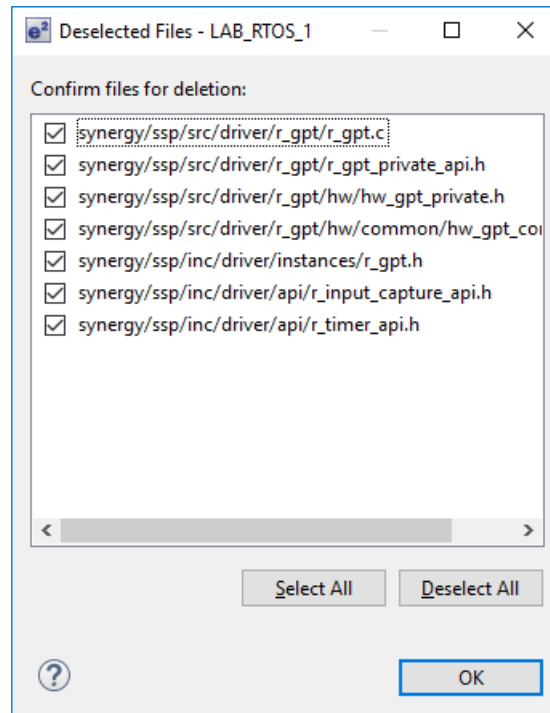
3. Remove the DAC Timer Semaphore



Generate the project content



When removing modules, SSP files from your project will be removed. You will get a notification confirming the deletion.



With the module & semaphore removed we will modify the `dac_thread_entry.c` file.

4. Remove the API which opens the gpt and starts the timer.  
This can be done by changing the `STEP_NUMBER` to 2 and saving the file
5. Where indicated, add the code so that the thread suspends for 1 second.

```
tx_thread_sleep(100);
```

where 100 equates to 100 RTOS ticks, which by default are 10ms.

Therefore, the task will suspend to  $100 * 10\text{ms} = 1\text{s}$ .

6. Build and debug your application
7. Set a breakpoint on the line of code `tx_thread_sleep();`
8. Run the code.

Open the Expressions window and you should see the `g_trng_result` changing every time you hit the breakpoint.

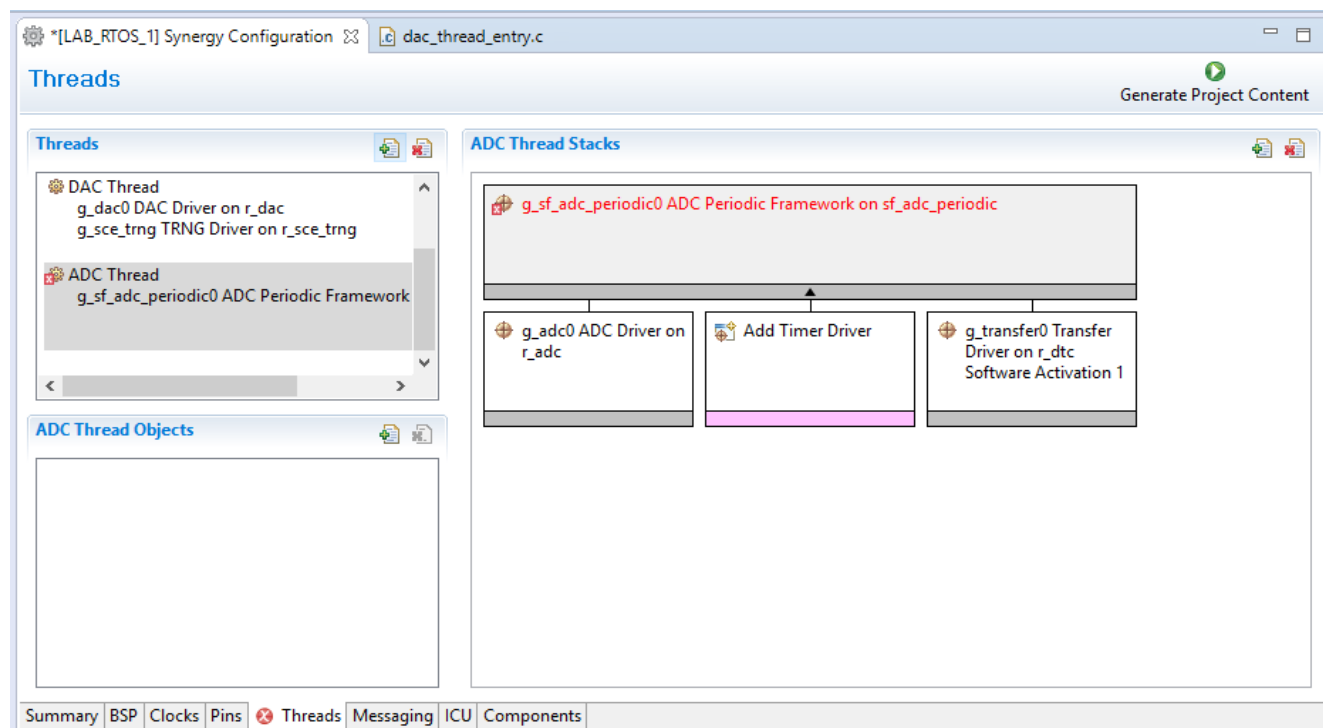
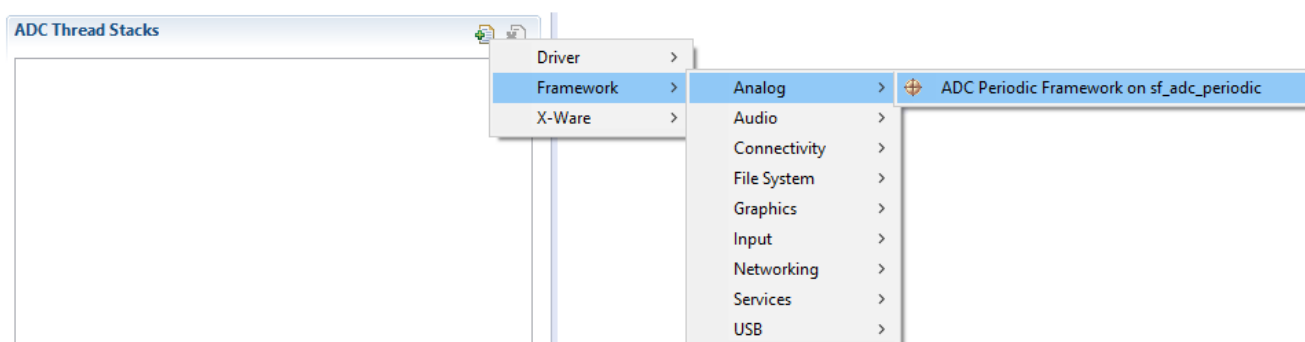
*NOTE: If `g_trng_result` is not displayed, then add this to the Expressions window.  
Remember, you can set real time refresh on `g_trng_result` and remove the breakpoint.*



## Section 4: Adding the ADC Periodic Framework

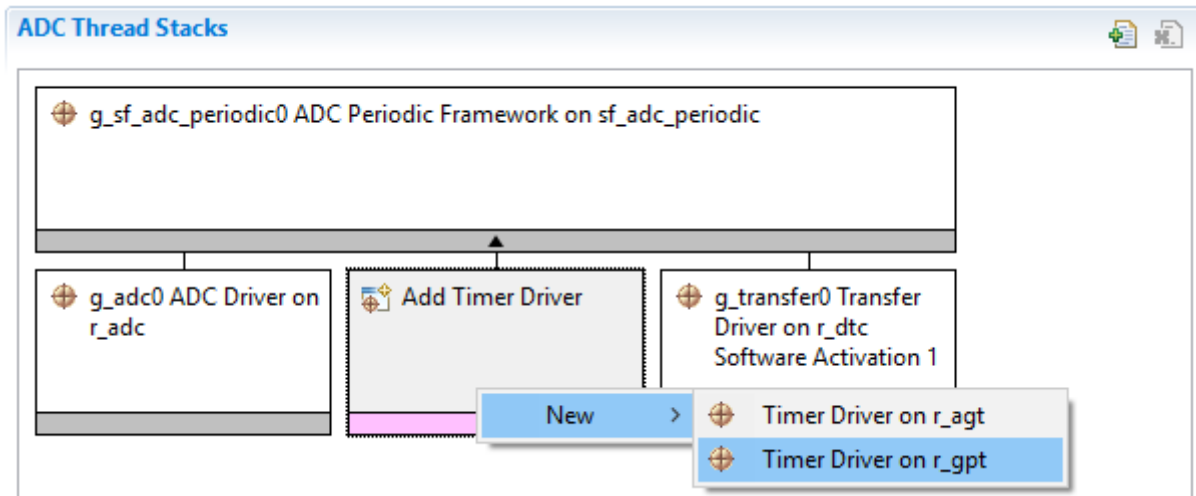
In this section we will recreate the ADC – Timer- Transfer functionality of the previous project. But rather than use individual modules we will use one of the Synergy Frameworks.

1. Switch from the Debug perspective to the Synergy Configuration perspective.
2. Open the Synergy Configuration tab.
3. As you did in section 2, create a new thread can call it `adc_thread` and give it a priority of 3.
4. To this `adc_thread` add the ADC Periodic Framework.



The Framework will be red, indicating that some user configuration is required. In this case “Requires Timer Driver”

5. Add a Timer Driver on r\_gpt to the framework



6. Make the following changes to the ADC Periodic Framework, ADC, Timer and Transfer modules.

<b>g_sf_adc_periodic ADC Periodic Framework on sf_adc_periodic Modifications</b>	
Length of the data-buffer	64
Number of sampling iterations	32
Callback	g_adc_framework_user_callback

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

<b>g_timer1 Timer Driver on r_gpt Modifications</b>	
Name	g_gpt1
Channel	1
Periodic Value	31250
Periodic Unit	Microseconds

The following is an extract from the SSP User's Manual, explaining what the ADC Periodic Framework is and does:

*The ADC Periodic Framework can be configured to use the ADC to sample any of the available channels at a configurable rate and buffer the data for a configurable number of sampling iterations before notifying the application. The ADC Periodic Framework uses the ADC, GPT, and DTC peripherals on Synergy MCUs.*

*The ADC Periodic Framework samples and buffers ADC data, and notifies the application once the configured number of samples are buffered.*

More information on the framework can be found in the SSP Hardware User's Manual.

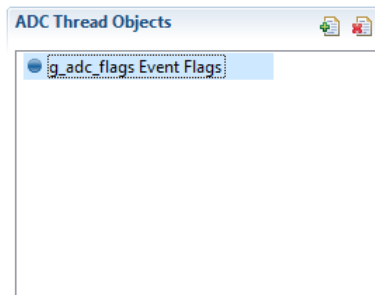
When in operation, the ADC Periodic Framework uses 2 buffers to store the ADC results. When 1 buffer is full, the framework notifies the application via the callback function and the argument event is set as [\*SF\\_ADC\\_PERIODIC\\_EVENT\\_NEW\\_DATA\*](#). In addition, the callback argument `buffer_index` indicates which of the 2 buffers is full.

The framework is configured to perform 32 sampling iterations, at a frequency of 31.250 ms. This will result in a buffer being filled every 1 second (  $32 \times 31.250\text{ms} = 1\text{s}$  )

The `adc_thread` main execution loop will wait for the ADC framework to finish a scan. It will then read the data in the buffer.

Event flags can be used to indicate when data is ready and which buffer has data.

7. Add an Event Flags Thread Object called `g_adc_flags`.



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



## Section 5: Adding APIs for the ADC Periodic Framework.

1. Copy the file `adc_thread_entry.c` provided on the USB memory stick, to the `src` folder.
2. Open the header file "`step_select.h`" file and set the MACRO number to 3 and save the file.
3. Where indicated, enter the following SSP API calls

The first is to start ADC Periodic Framework.

```
ssp_err = g_sf_adc_periodic0.p_api->start(g_sf_adc_periodic0.p_ctrl);
```

The second is in the thread while(1) loop, which is waiting to get an Event flag

```
tx_event_flags_get (&g_adc_flags, ALL_FLAGS, TX_OR_CLEAR, &actual_flags,
TX_WAIT_FOREVER);
```

You may have noticed that to use the framework it is not required to call an open API, as has been required with the other driver modules. Even though the open is still required, this code is automatically generated for you. This can be seen in the file `\src\synergy_gen\adc_thread.c`, in the function

```
void sf_adc_periodic_init0(void)
```

All frameworks will be opened automatically.

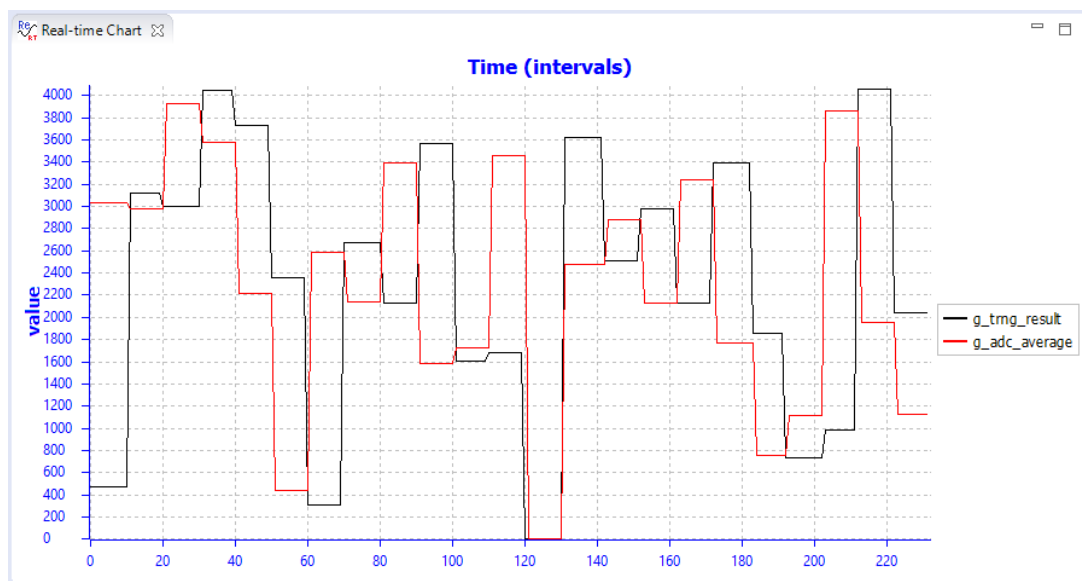
4. Build and debug your application.

Ensure that the DAC and ADC pins are connected - pin P0\_00 (AN000) to P0\_14 (DAC0)

5. Navigate to the Real-time Chart window and add 2 series (right click in the window to open menu)

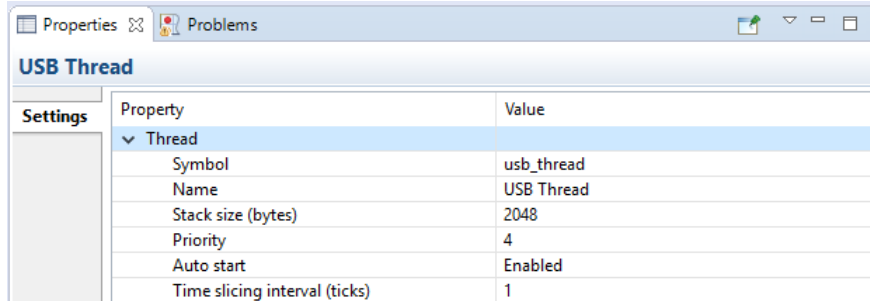
Add the series `g_trng_result` and `g_adc_average`

When the application runs you will hopefully see the `g_adc_average` following the `g_trng_result`, but with a delay of 1 second.



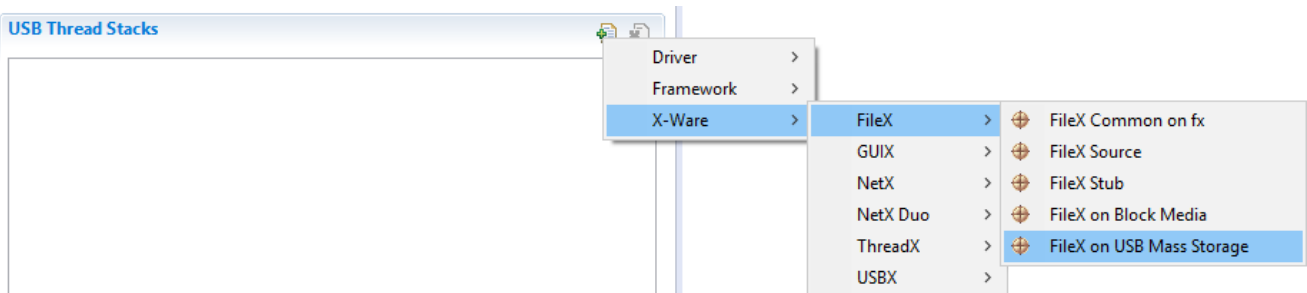
## Section 6: Add USB Host functionality

1. Create a new thread called usb\_thread.
2. Set its properties as shown

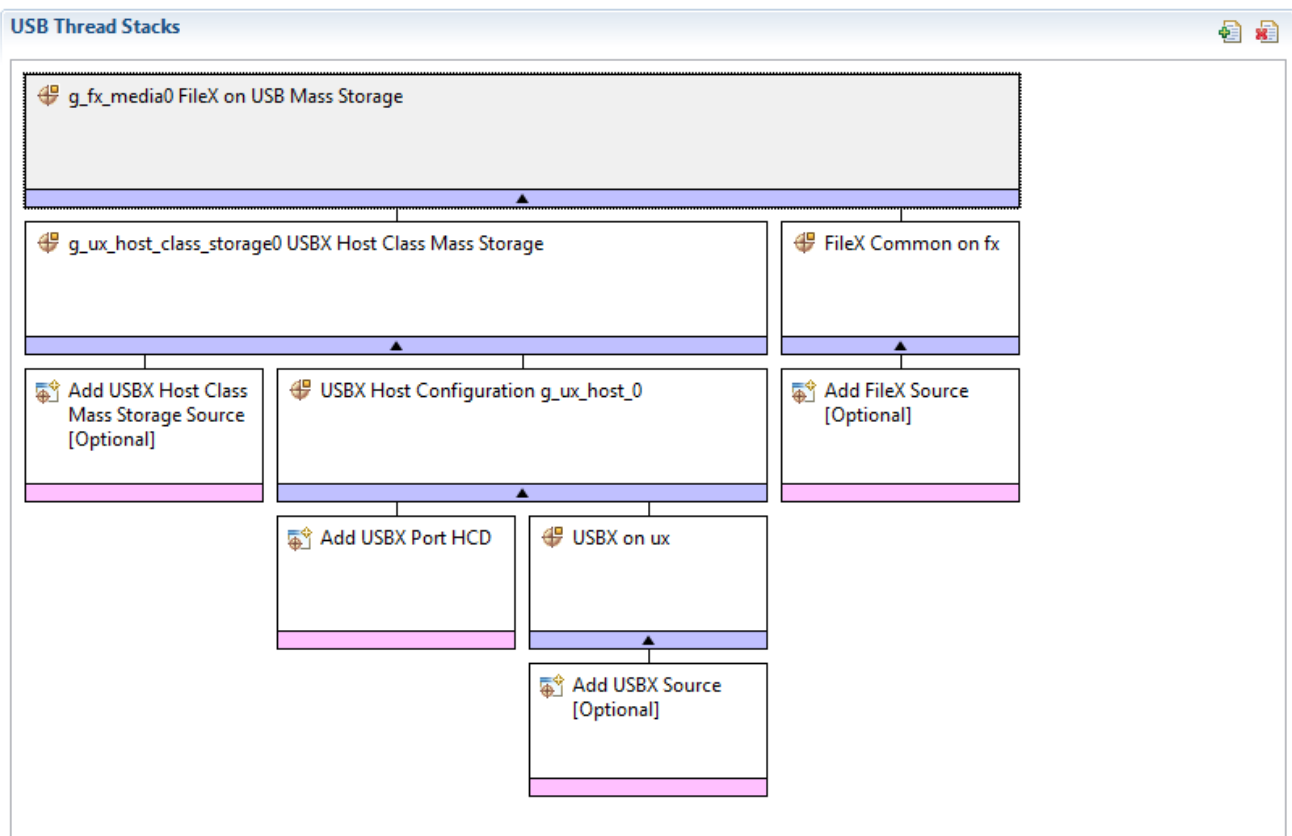


Property	Value
Symbol	usb_thread
Name	USB Thread
Stack size (bytes)	2048
Priority	4
Auto start	Enabled
Time slicing interval (ticks)	1

3. To this thread add the X-Ware component FileX on USB Mass Storage



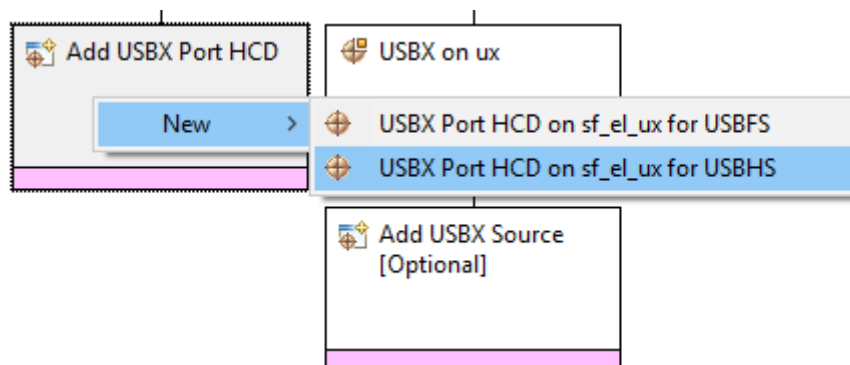
The stack view will look as such:



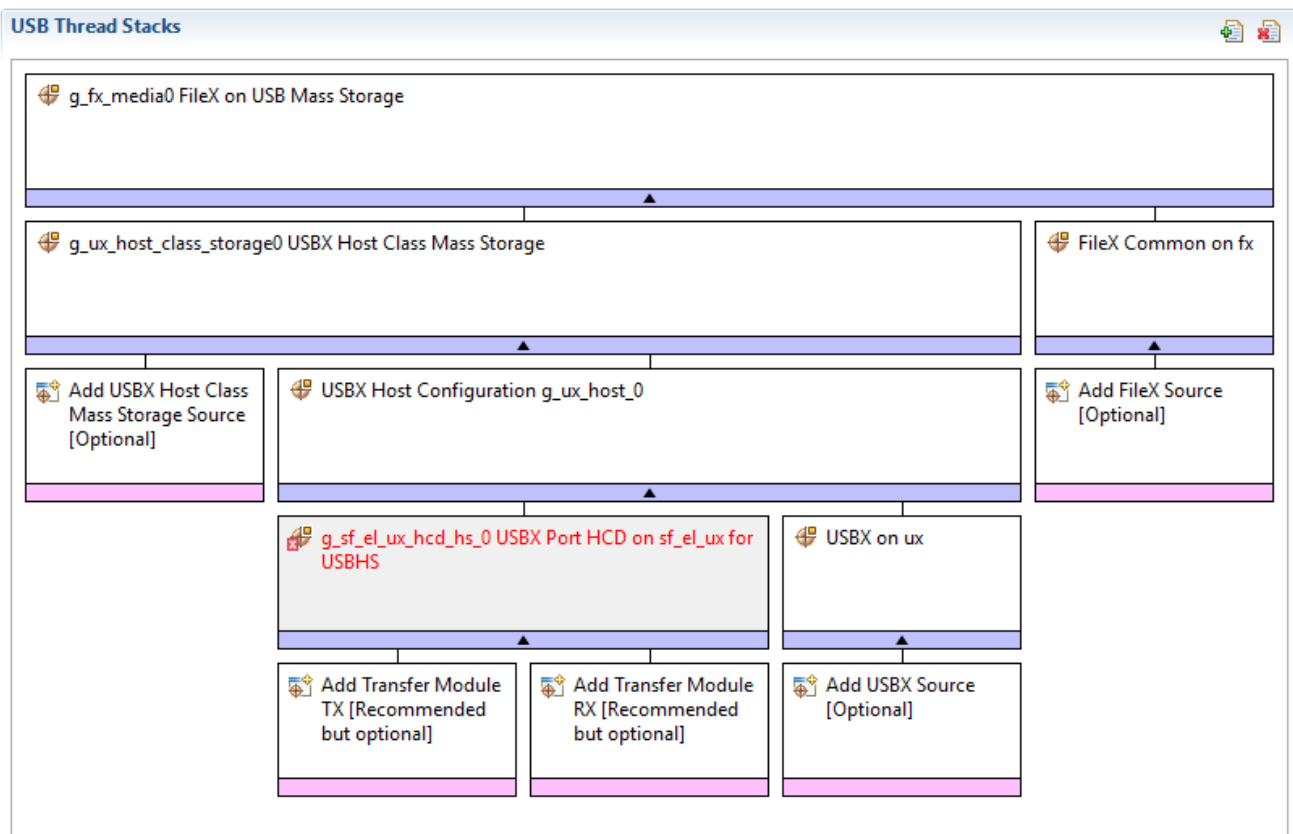
It can be seen that this X-ware component has many more levels than the previous modules that have been used, but it is configured in the same way.

- First add a USBX Port HCD. The Synergy S5D9 has 2 USB peripherals capable of supported USB Host, a High Speed and Full Speed peripheral. The S5D9-PK board is configured so that the USB Host port is connected to the High Speed peripheral.

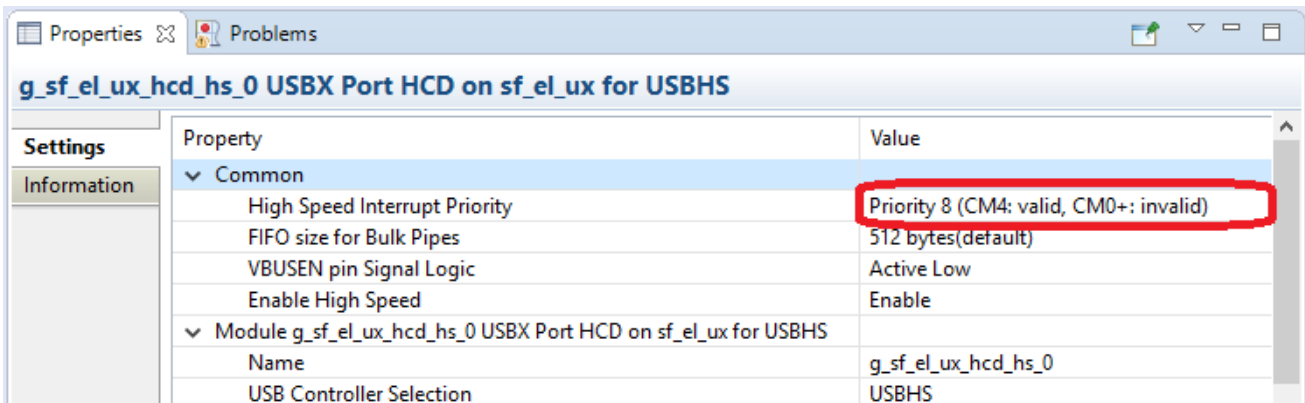
Click on the USBX Port HCD and select New > USBX Port HCD on sf\_el\_ux for USBHS



Once selected, the stack view will look as such:



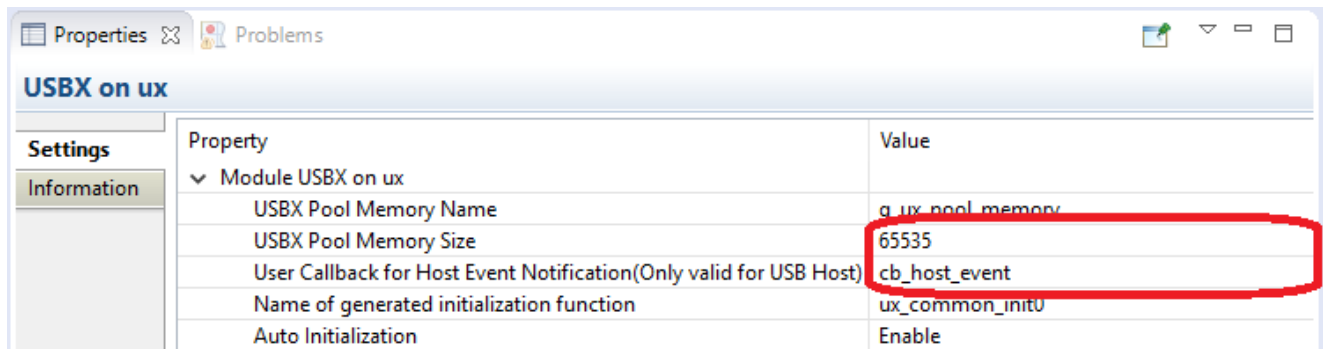
5. For this application, only a few property changes need to be made:



Property	Value
<b>Common</b>	
High Speed Interrupt Priority	Priority 8 (CM4: valid, CM0+: invalid)
FIFO size for Bulk Pipes	512 bytes(default)
VBUSEN pin Signal Logic	Active Low
Enable High Speed	Enable
<b>Module g_sf_el_ux_hcd_hs_0 USBX Port HCD on sf_el_ux for USBHS</b>	
Name	g_sf_el_ux_hcd_hs_0
USB Controller Selection	USBHS

Enable the USB High Speed interrupt.

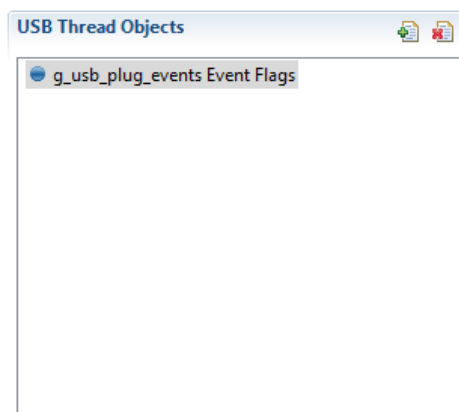




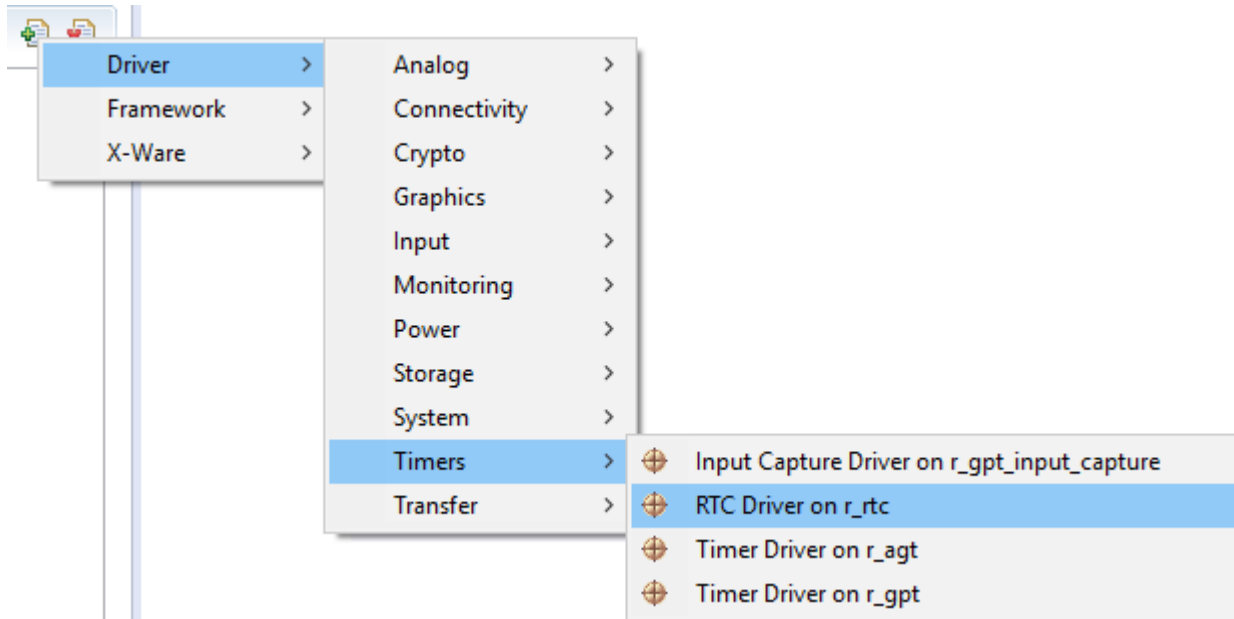
By default, the USB thread is configured with the USBX Pool Memory size for USB Function. We have to increase it for USB Host functionality.

Also, the application will use a User Callback function for detecting the insertion / removal of a USB device.

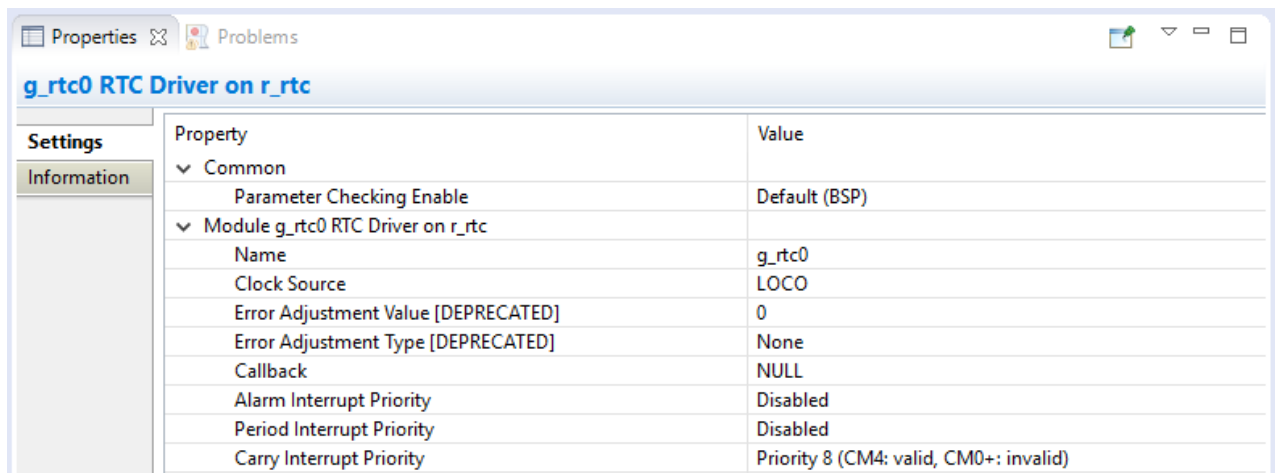
- The User Callback function will use Event Flags to signal to the USB thread that the USB has been inserted or removed. Therefore, add an event flag: `g_usb_plug_events`



7. Finally, add the Real Time Clock Driver to the USB Thread.



8. Enable the Carry Interrupt of the RTC Driver.



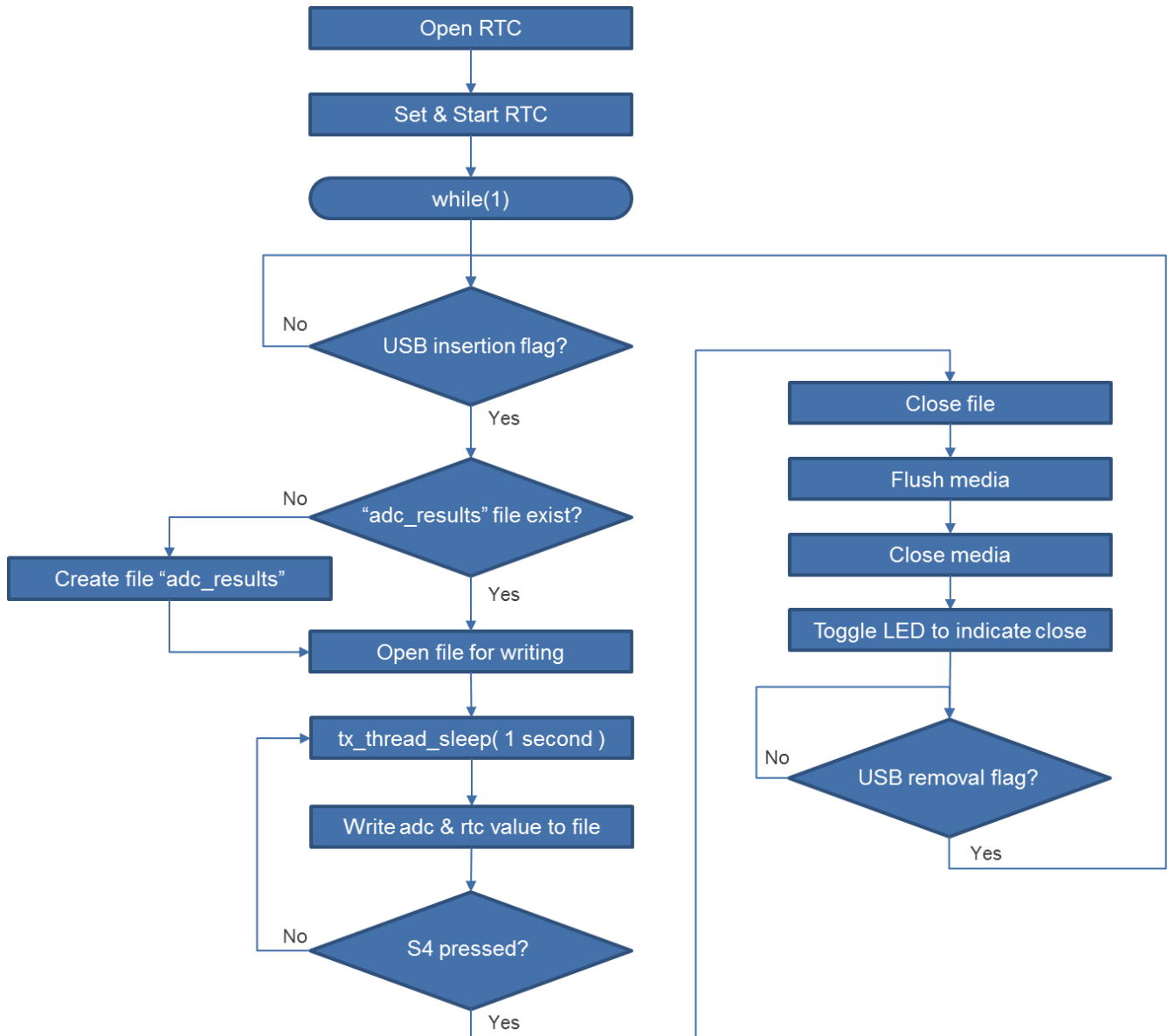
9. Generate the project content



## Section 7: Add USB Host Code

1. Copy the file `usb_thread_entry.c` provided on the USB memory stick, to the `src` folder.
2. Open the header file "`step_select.h`" file and set the MACRO number to 4 and save the file.

The code that you have added will perform the following!



3. Build and debug your application

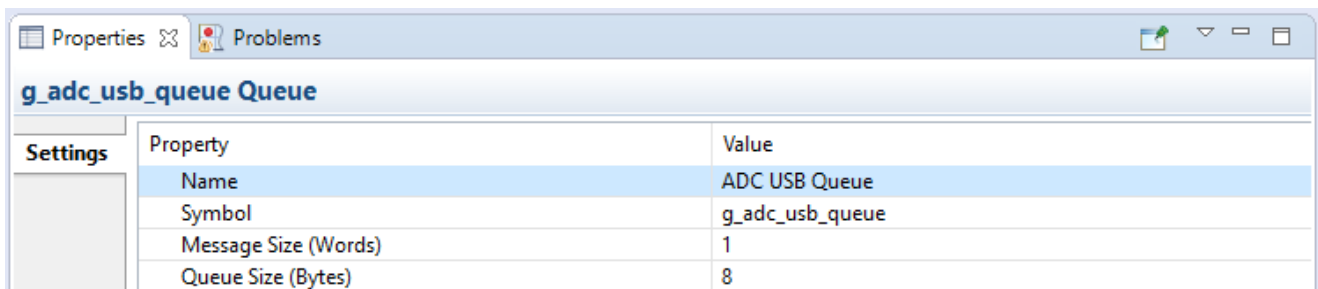
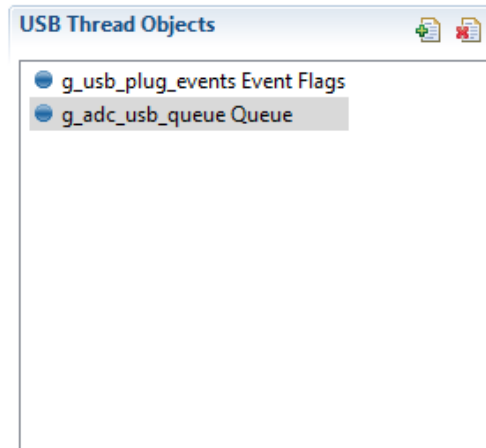
Run the application and plug in a USB drive to the board. After a few seconds press and hold S4 to "unmount" the USB device and view the text file on your PC. LED2 (Red LED) will toggle to show that USB device can be removed.

You should see a file with an ADC value of 0000, and a time stamp.

## Section 8: Inter-thread communication with a Queue

The final part of the lab is to transfer the `adc_average` value created in the ADC thread to the USB thread via a queue.

1. To the USB Thread, add a Queue Thread Object.



2. Generate the project.



3. Open the header file "**step\_select.h**" file and set the MACRO number to 5 and save the file.
4. In `usb_thread_entry.c`, in the do – while loop that writes data to the file, where indicated add code to read from the queue

```
tx_queue_receive(&g_adc_usb_queue, &rec_data, TX_WAIT_FOREVER);
```

5. Add code to the `adc_thread` to send to the queue.

```
status = tx_queue_send(&g_adc_usb_queue, &g_adc_average, 50);
```

Here, we do not want the thread to wait indefinitely to send to the queue, as data will only be read from the queue when the USB device is inserted. If the wait was TX\_WAIT\_FOREVER and a USB device was not inserted the ADC thread would suspend. To handle this case, we can flush the queue if it is full.

Build and debug your application.

Run the application and plug in a USB drive to the board. After a few seconds press S4 to “unmount” the USB device and view the text file on your PC.

You should see a file with variable ADC values, and a time stamp.

***Congratulations. You have finished the labs.***