Renesas Synergy™ Platform

DK-S124 Out-of-Box Programming Guide

Introduction

The Out-of-Box (OoB) program shows a multithreaded application design that reads the analog values present on the potentiometer, light sensor, and temperature sensor, and displays those values in the included PMOD LCD screen. The three LEDs on the board either flash or chase to the value read from either the light sensor or the potentiometer. Switch 1 and switch 2 are used to select which sensor is used, and if the LEDs flash or chase.

This application note describes the OoB program of the DK-S124 MCU Group in more detail than the *Quick Start Guide* included in the development kit.

Prerequisites

This application note assumes that you have some experience with the IAR Embedded Workbench® for Renesas Synergy™ and Synergy Software Package (SSP). Before you perform the procedure in this application note, follow the procedure in the *Quick Start Guide* for your board to build and run the Blinky project. Doing so enables you to become familiar with the Renesas Synergy™ e^2 studio and the SSP, and also ensures that the debug connection to your board functions properly.

Required Resources

The example application targets Renesas Synergy S124 devices. To build and run the application, you need:

- Renesas Synergy™ DK-S124 MCU Group (v3.0) with included PMOD LCD
- PC running Microsoft® Windows® 7 or above with the following Renesas Synergy™ software installed:
  - e^2 studio ISDE v7.3.0 or later
  - IAR EW for Synergy v8.23.3 or later
  - Synergy Standalone Configurator v7.3.0 or later
  - Synergy Software Package (SSP) v1.6.0 or later.

You can download the required software from the Renesas Synergy™ Gallery at: www.renesas.com/synergy/software.

Refer to the included *Renesas Synergy Project Import Guide* (r11an0023eu0121-synergy-ssp-import-guide.pdf) for instructions on importing the project into e^2 studio, and then building and running the project.

Time Required and Steps

You should be able to install, build, and run the example application in under 30 minutes following these basic steps:

1. Configure the DK-S124 MCU Group board for the OoB application
2. Import, build, and debug the project
3. Observe the analog sensor values change on the LCD screen
4. Change the mode of operation using the two momentary pushbutton switches
5. Vary the frequency of the LED flashing using either the potentiometer or the light sensor

Contents

1. Overview

This application note describes how the capabilities of the DK-S124 Synergy MCU Group and the challenges faced when trying to design a robust application using a low-cost SPI LCD screen.

The DK-S124 Synergy MCU Group is designed to familiarize developers with the lowest cost processors available in the Renesas Synergy™ Platform line of products. The S1 series processor is a 32-MHz Arm® Cortex-M0+ CPU with 128 KB of code flash.

The Renesas Synergy™ Platform also provides a high-level tool, known as GUIX Studio, for rapidly developing graphical user interfaces in a point-and-click environment on these types of higher-end processors. A variety of application notes detail the development of complex graphical interfaces on these higher end boards, and these processors are also available on the Renesas Synergy™ Gallery website: https://synergygallery.renesas.com/about/gallery.

The DK-S124 Synergy MCU is ideally suited for applications requiring a simple human machine interface (HMI) to convey critical information to the user. The development kit ships with a 128x128-pixel PMOD LCD. The DK-S124 Synergy MCU has a single PMOD interface to accommodate this LCD. The DK-S124 Synergy MCU communicates with this LCD over a SPI interface. A principle design consideration for this application note is to provide a small graphical library in order to display characters on the PMOD LCD in a timely fashion.

2. Synergy Platform Capabilities

The library driving the PMOD LCD, the Renesas Synergy™ Platform, contains all the remaining resources necessary for quickly prototyping the type of application demonstrated in the DK-S124 Synergy MCU Group OoB program. Figure 1 highlights the DK-S124 MCU Group peripherals used by the OoB application.

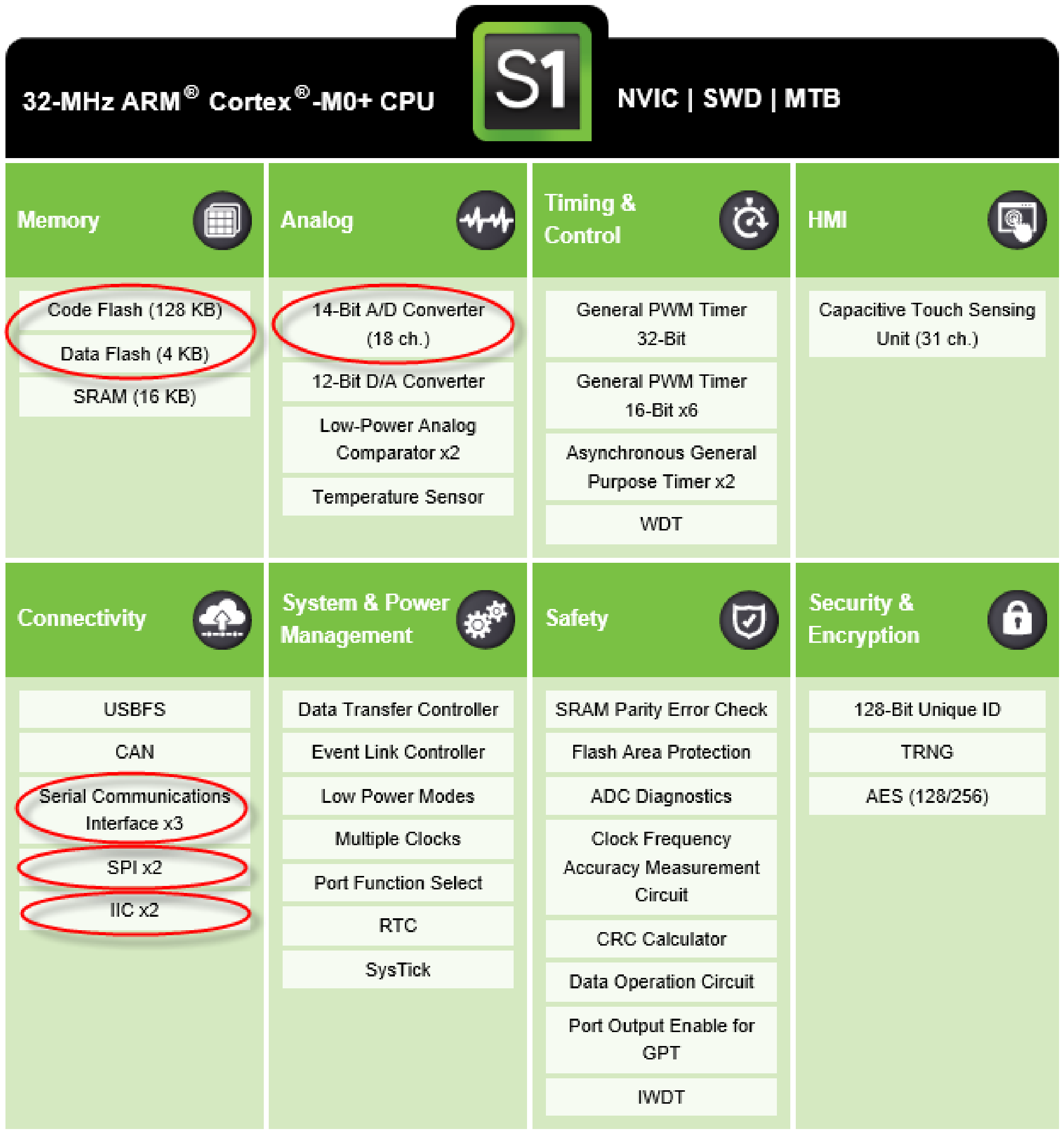


Figure 1. S1 peripherals used by the OoB

Table 1 defines the processor pin connections used to connect these peripherals to the various external components on the DK-S124 Synergy MCU Group board. For additional pin references, see the DK-S124 Synergy MCU schematics.

Table 1. Port summary of peripheral connections used by the OoB

Schematic part	Description	S1 port connection	Pin function
S1	Momentary switch	Port 2 pin 6	I/O port pin
S2	Momentary switch	Port 0 pin 4	I/O port pin
POT1	10K potentiometer	Port 0 pin 12	Analog channel 7
U11	APDS-9005 light sensor	Port 0 pin 0	Analog channel 0
U14	TMP35 temperature sensor	Port 0 pin 1	Analog channel 1
U13	BMA250 3-axis accelerometer	Port 2 pin 6	IRQ
U13	BMA250 3-axis accelerometer	Port 4 pin 1	SDA0 (I2C clock)
U13	BMA250 3-axis accelerometer	Port 4 pin 0	SCL0 (I2C data)
U5	ISL41387 RS-232/RS485 transceiver	Port 4 pin 10	RXD0 (receive data)
U5	ISL41387 RS-232/RS485 transceiver	Port 4 pin 11	TXD0 (transmit data)

3. OoB Application Design

The SK-S124 Synergy MCU board contains several analog sensors along with a 3-axis accelerometer connected to the I^{2}C bus. The analog sensors include a potentiometer, a temperature sensor, and a light sensor. The OoB application continuously reads the analog values provided by these sensors and displays them on the PMOD LCD screen. It also flashes three on-board LEDs in proportion to the analog voltage produced by the potentiometer or the light sensor. The LEDs may be programmed to flash in unison or to chase each other. The selection of which sensor drives the flashing rate, and whether the LEDs flash or chase, is controlled by the pushbutton switches **S1** and **S2**, found in the lower right corner of the DK-S124 Synergy MCU board.

The application illustrates the use of Synergy peripheral drivers to read the sensor values, while demonstrating how easy it is to setup a multi-threaded application. The application has separate threads to read the sensors, display the values to the LCD and flash the LEDs based on the pushbutton and sensor inputs.

Table 2 summarizes the three user-defined threads and their functions.

Table 2. Application thread names and functions

Thread name	Thread function
LED thread	Controls flashing/chasing of the 3 on-board LEDs
LCD thread	Updates the LCD as new data arrives from the three analog sensors
Inputs thread	Periodically polls the switch inputs and reads the analog sensor voltages

Figure 2 conceptually shows the OoB application. The input thread periodically samples the state of the pushbuttons and the voltage values present on the analog sensors. The input thread makes this data available to the other threads through two access functions, eliminating the need for global variables.

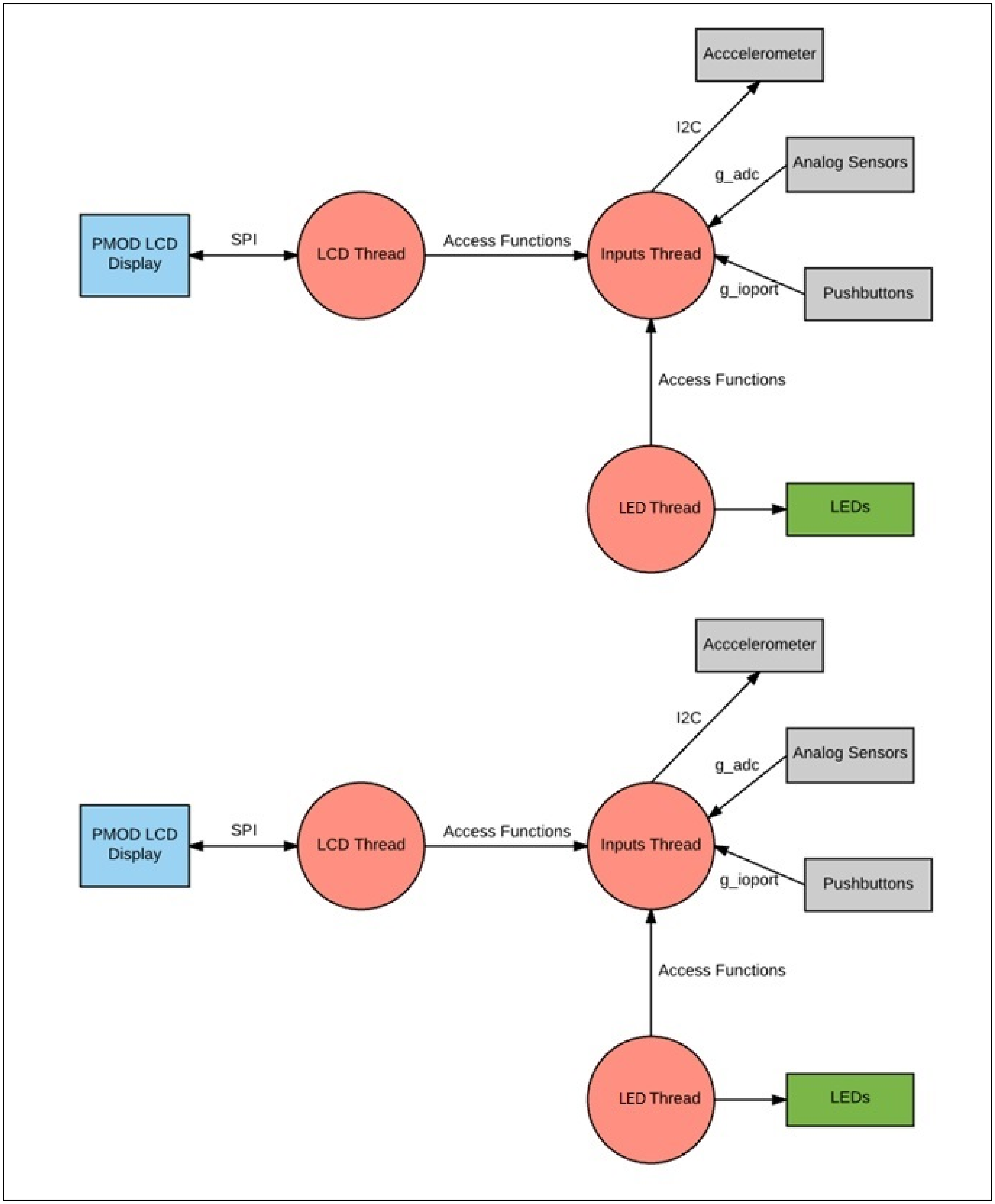


Figure 2. Conceptual diagram of the OoB application

While both pushbuttons are connected to hardware pins with interrupt capability, this application uses a thread to periodically poll the switch states by reading the I/O port pins connected to the switches. This simple technique provides basic switch debouncing by reading the state of the switch at an interval longer than the typical mechanical switch bounce time (50 ms). Other more complex debouncing techniques may be used, such as the excellent switch debouncing algorithms published by Jack Ganssle (http://www.ganssle.com).

Other application notes, such as the *Simple Audio Example*, highlight the use of the Synergy IRQ Framework to detect pushbutton presses. There are varying schools of thought about the validity of connecting mechanical switches to interrupts. The Synergy system makes it easy to do it either way.

3.1 Inputs Thread Logic

Figure 3 shows the logic implemented in the inputs thread. To be responsive, the inputs thread samples the sensors and switches inputs every 50 ms. One of the advantages of threads is shared memory space. The LCD thread periodically updates the PMOD LCD screen with the current mode, selected by pushbuttons, and the latest sensor values. Subsequent versions of this application note will demonstrate the use of the Synergy Messaging Framework to send messages to the LCD thread as these values change. This allows the LCD thread to be event driven.

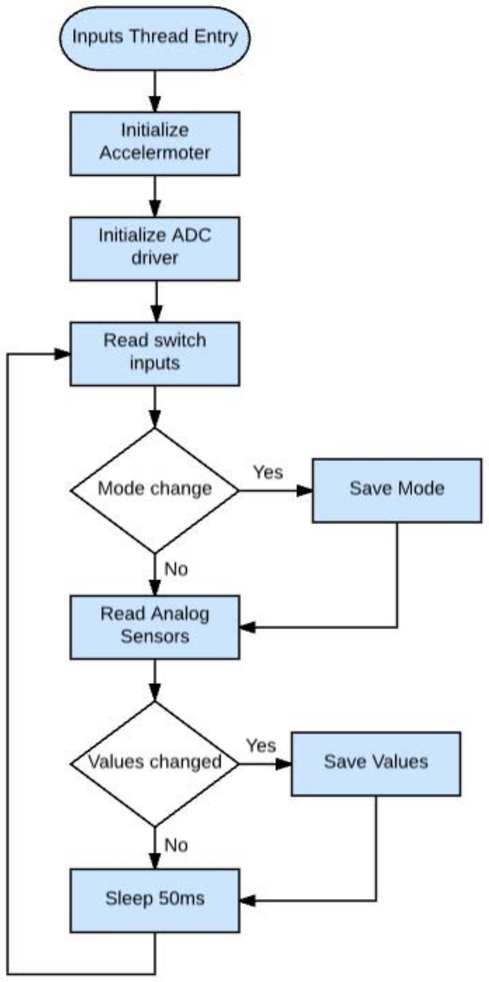


Figure 3. Simplified flowchart of the inputs thread

3.2 Accelerometer Initialization

On the SK-S124 Synergy MCU v3.0 board, the accelerometer IRQ line is shared with **S1**, since the accelerometer IRQ pin defaults to a push/pull configuration and works as an active-high IRQ state. This means that it is always pulling down on the line connected to S1, which prevents the use of this line in reading the state of **S1**. To use **S1** to select between the potentiometer and light sensor, it is necessary to change the interrupt configuration of the accelerometer. To implement this change, the I^{2}C framework is added to the inputs thread and several I^{2}C commands are issued to the accelerometer.

On the DK-S124 Synergy MCU v3.0, the jumper J20 is used to disconnect the accelerometer IRQ pin from S1, and so the accelerometer initialization described above can be skipped.

3.3 LCD Thread Logic

The LCD thread periodically wakes up, reads the current mode and sensor values, and displays these values to the PMOD LCD screen. Figure 4 shows the logic implemented in the LCD thread. The mode and sensor values are obtained by calling the two access functions provided by the inputs thread.

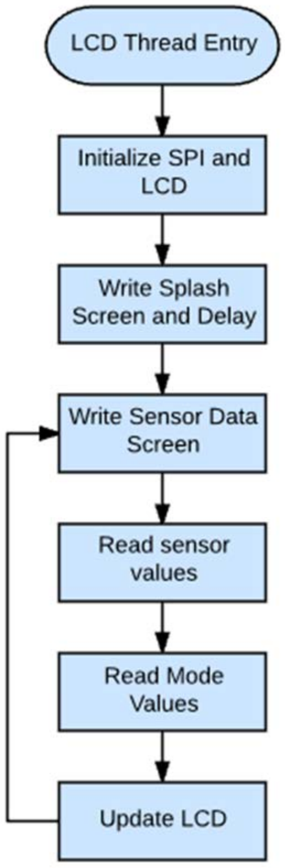


Figure 4. Simplified flowchart of the LCD thread

3.4 LED Thread Logic

The LED thread must refresh the LEDs at a periodic rate driven by the voltage present on either the light sensor or the potentiometer, depending on the current mode selected by the operator. The standard `tx_thread_sleep()` call is used as the periodic delay.

When the LED thread wakes up, it calls two different access functions, `inputs_get_sensor_values()` and `inputs_get_mode()`, to obtain the current sensor values. This illustrates one of the benefits of threads, which is shared memory space. This allows the inputs thread to maintain a local copy of these variables while still providing access of those variables to the LED thread. The access functions eliminate the use of global variables and allow the sensor variables to have local scope inside the inputs thread.

3.5 Threads and Modules

As is the case with all Renesas Synergy™ applications, module drivers are added to threads and their properties are configured using the ISDE configurator. Open the configurator by double clicking on the **configuration.xml** file from inside the project explorer in e^2 studio. This application note does not cover all the driver modules added to the application. As an example, the addition of the SPI driver is discussed; it is necessary for communicating with the PMOD LCD. All other drivers may be inspected through the ISDE in a similar manner. For details on Synergy modules and drivers, refer to the *SSP User's Manual*.

As previously mentioned, the S124 processor communicates with the PMOD LCD through the SPI bus. Therefore, it is necessary to add an SPI driver to the thread that communicates with the display, which in this case, is the LCD thread. Figure 5 shows the ISDE configurator with the LCD thread highlighted. You can see added the `r_sci_spi` driver.

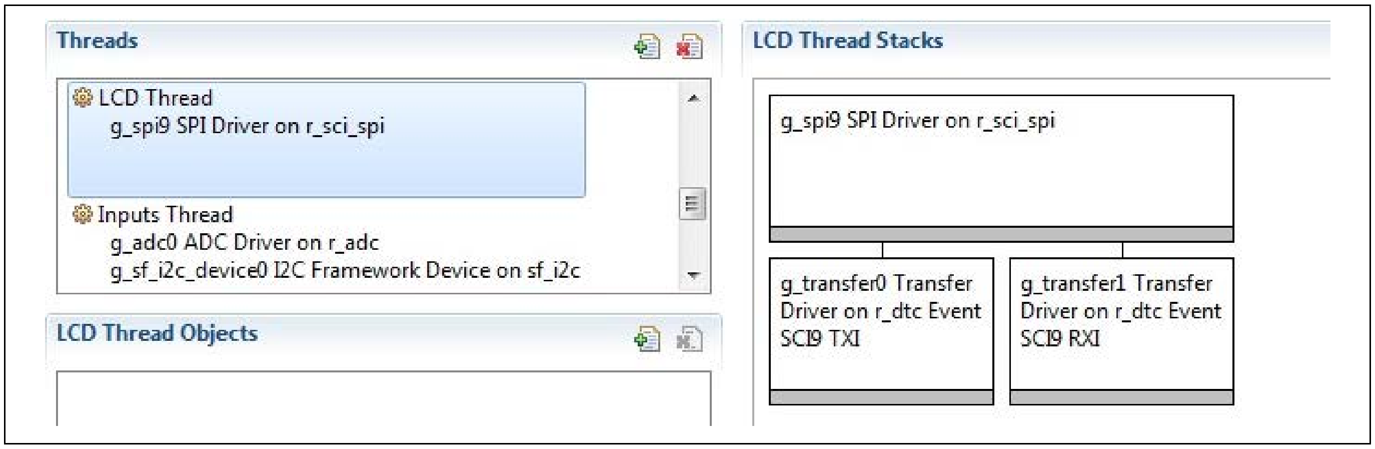

Threads
LCD Thread
g_spi9 SPI Driver on r_sci_spi
Inputs Thread
g_adc0 ADC Driver on r_adc
g_sf_i2c_device0 I2C Framework Device on sf_i2c
LCD Thread Objects
LCD Thread Stacks
g_spi9 SPI Driver on r_sci_spi
g_transfer0 Transfer Driver on r_dtc Event SCI9 TXI
g_transfer1 Transfer Driver on r_dtc Event SCI9 RXI

Figure 5. Threads tab display for the OoB program

Synergy processors contain two different peripherals for SPI communication — a dedicated SPI peripheral, and the Serial Communication Interface (SCI), that may act either as a simple I^{2}C, simple SPI, or UART (synchronous or asynchronous). The SSP supports both peripherals. The same scenario is true for the I^{2}C peripheral.

Choosing the peripheral to use is often based on the complexities of your system. You may not need all the power available on the dedicated SPI or I^{2}C peripherals. You may need to use the pins assigned to the dedicated I^{2}C peripheral for another use. The decision regarding which peripheral to use must typically be made during the hardware design phase, since it affects which physical processor pins are connected to the peripheral.

One consideration when designing the system are the numbers of each type of peripheral. While the S124 processor only has two dedicated SPI and I^{2}C peripherals, it has nine SCI peripherals which can be configured as either simple SPI or I^{2}C.

The OoB application provides an example for each scenario. The SPI peripheral used to communicate with the PMOD LCD is connected to one of the nine SCI channels, while the accelerometer is connected to one of the two dedicated I^{2}C buses.

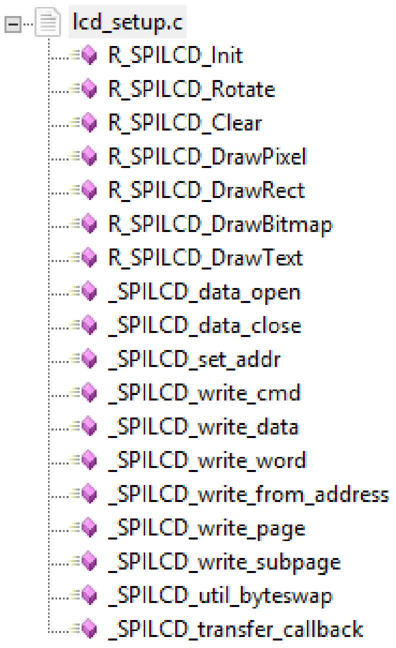
3.6 PMOD LCD Library

The PMOD LCD is a 128x128-pixel LCD that uses an industry standard of the ST7735 graphics controller. Several open source libraries are available from various vendors to communicate with similar screens using the same graphics controller. One of the limitations of using an SPI-based LCD is the slow performance due to addressing each pixel with a separate SPI write. Most of the systems have some latency between SPI writes, and some latency when writing to the other control lines necessary for talking to the display. When writing all the bytes necessary to fill an entire screen, or even writing to output a line of text, these latencies add up to a noticeable delay.

The PMOD LCD library included with the DK-S124 OoB Synergy MCU has been optimized to improve performance by buffering the writes and bursting them out one-at-a-time, eliminating much of the latency that would result from individual SPI write operations. One of the tradeoffs when doing this sort of buffering is that you must allocate buffer space to do it. Currently, the PMOD LCD library allocates a two-dimensional array consuming only 128 bytes for character buffering.

```
static uint16_t          lcd_pmod_buffer[2][64];
```

Figure 6 shows the API calls for the PMOD LCD library at the time this application note was written. The only API calls used in this application are `R_SPILCD Clear` and `R_SPILCD_DrawText`. The PMOD LCD library undergoes continuous development, so check the Renesas website for updates.



lcd_setup.c
- R_SPILCD_Init
- R_SPILCD_Rotate
- R_SPILCD_Clear
- R_SPILCD_DrawPixel
- R_SPILCD_DrawRect
- R_SPILCD_DrawBitmap
- R_SPILCD_DrawText
- _SPILCD_data_open
- _SPILCD_data_close
- _SPILCD_set_addr
- _SPILCD_write_cmd
- _SPILCD_write_data
- _SPILCD_write_word
- _SPILCD_write_from_address
- _SPILCD_write_page
- _SPILCD_write_subpage
- _SPILCD_util_byteswap
- _SPILCD_transfer_callback

Figure 6. LCD library API calls

Using the PMOD LCD library is easy. Use the following code to display the simple flash screen displayed on powering up.

```
    /** Initialize the PMOD LCD **/
 R_SPILCD_Init((spi_instance_t * const) &g_spi9, LCD_ROTATION_90);
 R_SPILCD_Clear(LCD_COL_BLACK);

 R_SPILCD_DrawText("DK-S124", -1, TEXT_COLOR_FG, TEXT_COLOR_BG, CURSOR_X(7), CURSOR_Y(5));
 R_SPILCD_DrawText("V 1.0", -1, TEXT_COLOR_FG, TEXT_COLOR_BG, CURSOR_X(7), CURSOR_Y(8));
```

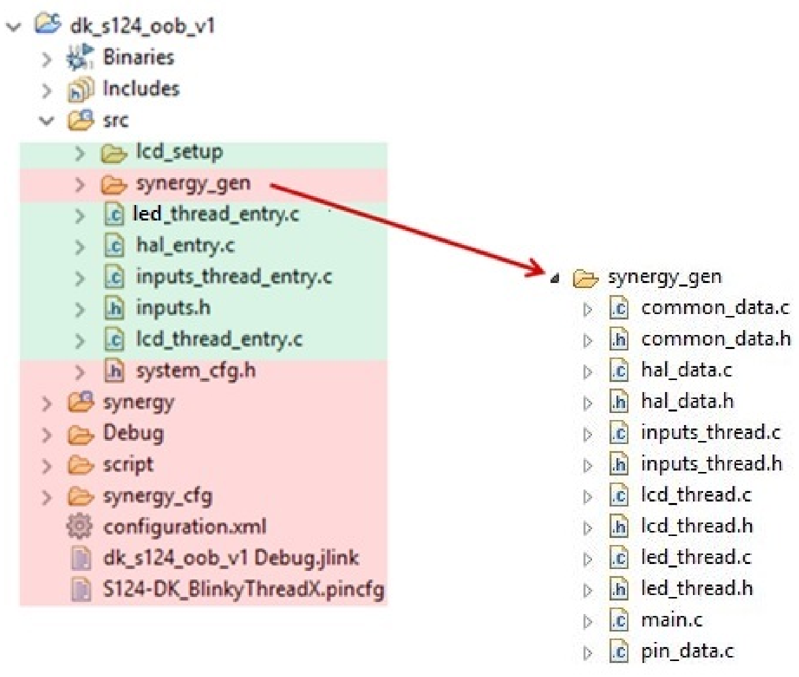
In most applications, you would use the Synergy SPI driver `open` call before accessing the SPI driver. This call would normally be done just inside the thread entry call, prior to entering the thread loop. However, in this case, the PMOD LCD library does this for you when you call the `R_SPILCD_Init()` routine to initialize the LCD. As you can see in the code snipped above, you simply pass a pointer to the SPI driver instance to the `R_SPILCD_Init()` call. This instance pointer is automatically created for you when you add the SPI driver to the LCD thread.

3.7 Source Code Layout

This section gives a brief overview of the source code layout you expect to see when you build the project with e^2 studio.

If you are already experienced with writing code for Renesas Synergy applications, you are probably familiar with how e^2 studio organizes a standard project. In this case, you can skip to the next section.

After you import the DK-S124 Synergy MCU project into e^2 studio and build the project, you should see a directory structure like the one shown in Figure 7. The first time you import the project, you may notice that the `synergy_gen`, `synergy`, `debug`, `synergy`, and `synergy_cfg` folders do not exist. These folders are built automatically by the framework when you compile the project. To reduce the size, these subfolders are normally not included when programs are distributed. The only files that you typically need to work within the project are files highlighted in green.



```
dk_s124_oob_v1
  Binaries
  Includes
  src
    lcd_setup
    synergy_gen
    led_thread_entry.c
    hal_entry.c
    inputs_thread_entry.c
    inputs.h
    lcd_thread_entry.c
    system_cfg.h
  synergy
  Debug
  script
  synergy_cfg
  configuration.xml
  dk_s124_oob_v1 Debug.jlink
  S124-DK_BlinkyThreadX.pincfg

synergy_gen
  common_data.c
  common_data.h
  hal_data.c
  hal_data.h
  inputs_thread.c
  inputs_thread.h
  lcd_thread.c
  lcd_thread.h
  led_thread.c
  led_thread.h
  main.c
  pin_data.c
```

Figure 7. Source code layout for DK-S124 OoB Synergy MCU v1.0 program

3.8 Design Example

The DK-S124 Synergy MCU comes preloaded with the out-of-box application software. Prior to running the program or programming the board from e^2 studio ISDE, you must first configure the board.

3.9 Configure the DK-S124 Synergy MCU for the Out-of-box Program

The steps to configure the DK-S124 Synergy MCU are:

1. Verify that the J3 header near the battery is configured with two jumpers to make connections 1-3 and 2-4, as shown in Figure 8.
2. Connect the J-Link-OB on DK-S124 connector J18 to the PC using a micro USB cable (Figure 8).

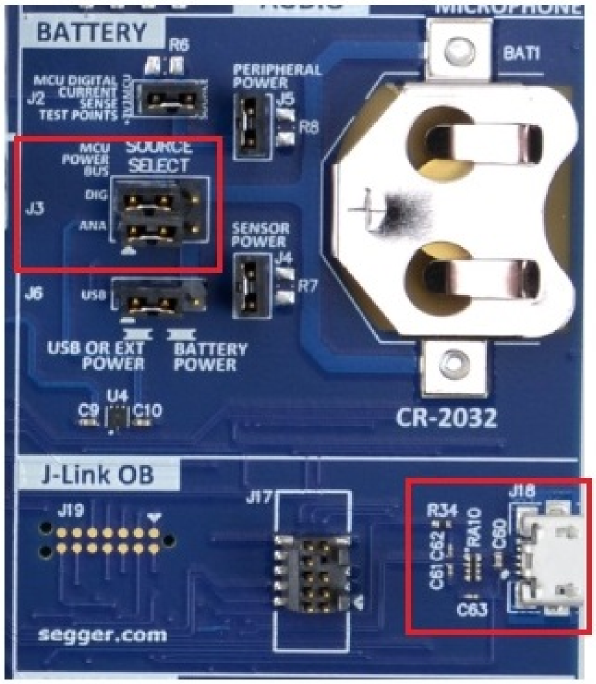


Figure 8. Power setup (top) and SEGGER J-Link® OB connection (bottom)

Once the DK-S124 is plugged in, it powers up and immediately starts flashing the three LEDs: LED1, LED2, and LED3. It displays a simple splash screen on the PMOD LCD for two seconds. This splash screen shows the board number and the version of the out-of-box software that shipped with the board (Figure 9).

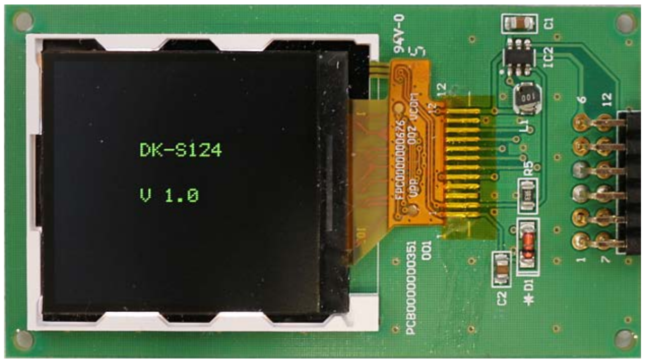


Figure 9. OoB splash screen

After 2 seconds, the screen changes to the analog measurement screen (Figure 10). The SSP Out-Of-Box application uses the A/D converter to read the voltage values present on the potentiometer (POT1), the light sensor U11 (APDS-9005), and the temperature sensor U14 (TMP 35). The program displays the raw hex values read from the light sensor and the potentiometer, and it converts the analog value read from the temperature sensor to the equivalent Fahrenheit temperature.

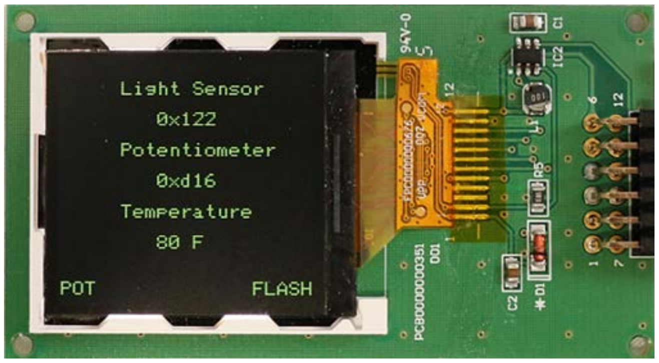


Figure 10. OoB analog sensor screen

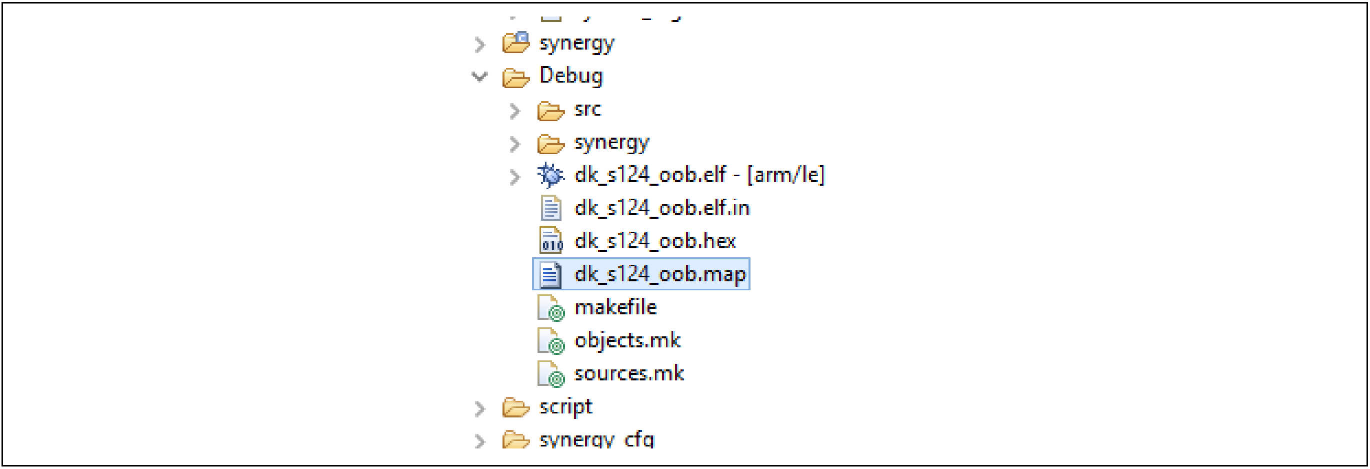
You can interact with the Out-of-Box program in the following ways:

1. The three LEDs (LED1, LED2, and LED3), all flash at the same time, or chase each other. Pushing the momentary pushbutton switch **S2** causes the LEDs to alternate between flashing and chasing. The bottom right corner of the screen indicates which mode the LEDs are operating in (Flash/ Chase).
2. The flashing rate of the LEDs is determined by the value read from the light sensor or the potentiometer. You toggle between these two by pushing the momentary pushbutton switch **S1**. The bottom left corner of the screen indicates which sensor is driving the LED flashing rate (POT/Light).
3. With the POT mode selected (push **S1**), rotate POT1 clockwise and counterclockwise. LED flashes increase or decrease appropriately. The screen displays the raw value reported by the A/D converter.
4. With the Light mode selected (push **S1**), move a light source (such as a flashlight) closer and farther away from the light sensor U11. Observe that the flashing rate of the LEDs changes. The raw hex value displayed to the screen also changes.

3.10 e^2 studio Tricks

When adding threads to an application, they default to 1 KB for stack space. Reducing unused stack space makes more memory available for the application. It is not easy to determine how much stack space is enough.

ThreadX® initializes all thread stacks with a known value, 0xEF, which makes inspecting each thread stack easier using the memory monitor included with e^2 studio. First, you must determine where the stacks are located in the physical memory. Locating stacks can be done one of two ways. The first is by viewing the map file that is produced when you compile the application, as the following figure shows.



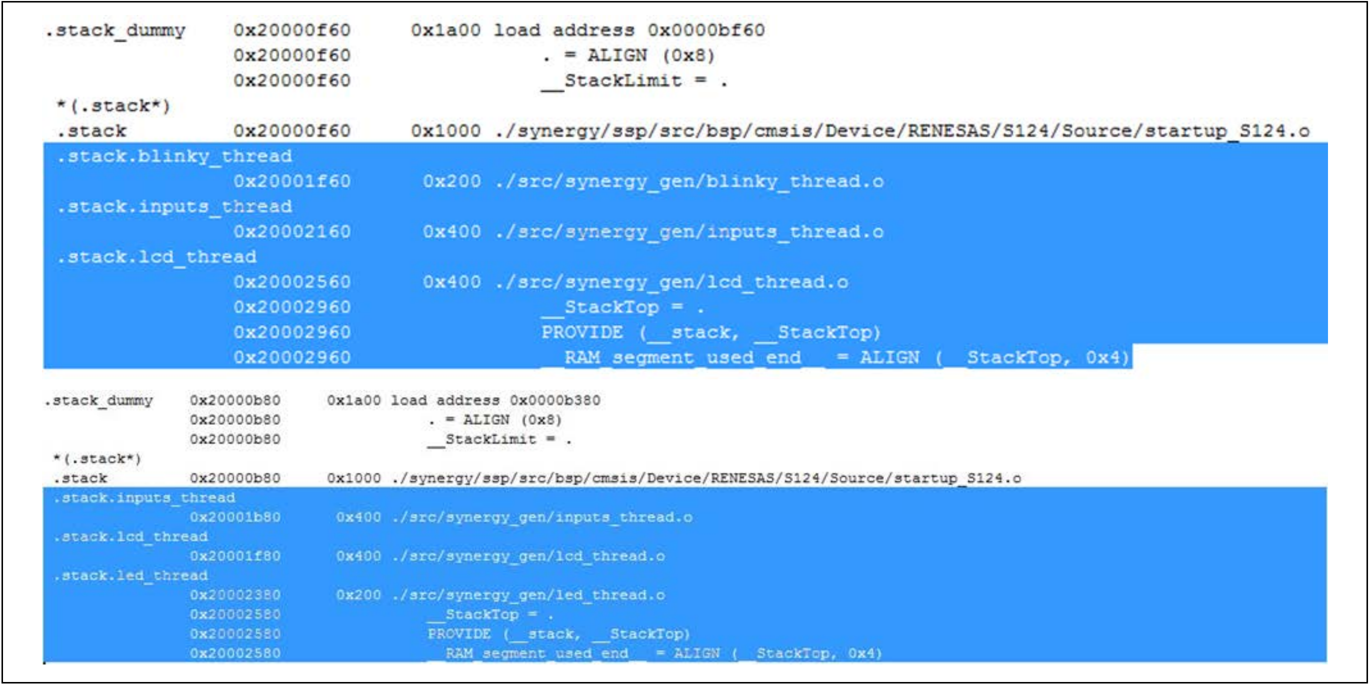
```
> synergy
v Debug
    > src
    > synergy
    > dk_s124_oob.elf - [arm/le]
      dk_s124_oob.elf.in
      dk_s124_oob.hex
      dk_s124_oob.map
      makefile
      objects.mk
      sources.mk
> script
> synergy_cfg
```

Figure 11. Location of .map file

Depending on the system configuration, e^2 studio may default to opening the file in some other editor. For example, systems with Visual Studio installed may open this file in Visual Studio when you double click on the file. If this happens, you can always right click on the file and select **Open With**>**Text Editor** to view the file inside the e^2 studio.

Once you have the file open, the fastest way to find the information you are looking for is to search for `stack_dummy`, which brings you to the section in the map file where the thread stack addresses are defined.

Figure 12 shows an example where the address of the inputs thread stack is 0x20002160, with a size of 0x400 (1024 bytes). To examine the stack memory, you can open the memory monitor. Locate the memory tab along the bottom of e^2 studio. If the tab is not currently visible, open it by going to **Window**>**Show View**> **Memory**, using the top tool bar. Click the small green + symbol to add a new memory monitor and type this address into the dialog box.



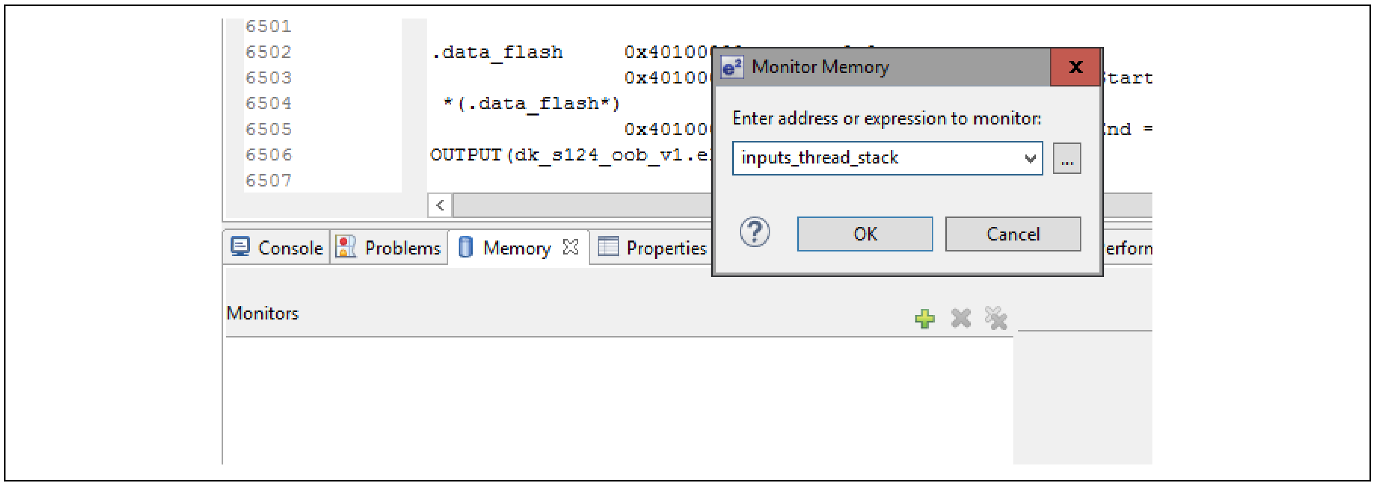
```
.stack_dummy    0x20000f60     0x1a00 load address 0x0000bf60
                0x20000f60              . = ALIGN (0x8)
                0x20000f60              __StackLimit = .
 *(.stack*)
 .stack         0x20000f60     0x1000 ./synergy/ssp/src/bsp/cmsis/Device/RENESAS/S124/Source/startup_S124.o
 .stack.blinky_thread
                0x20001f60      0x200 ./src/synergy_gen/blinky_thread.o
 .stack.inputs_thread
                0x20002160      0x400 ./src/synergy_gen/inputs_thread.o
 .stack.lcd_thread
                0x20002560      0x400 ./src/synergy_gen/lcd_thread.o
                0x20002960              __StackTop = .
                0x20002960              PROVIDE (__stack, __StackTop)
                0x20002960                __RAM_segment_used_end__ = ALIGN (__StackTop, 0x4)

.stack_dummy    0x20000b80     0x1a00 load address 0x0000b380
                0x20000b80              . = ALIGN (0x8)
                0x20000b80              __StackLimit = .
 *(.stack*)
 .stack         0x20000b80     0x1000 ./synergy/ssp/src/bsp/cmsis/Device/RENESAS/S124/Source/startup_S124.o
 .stack.inputs_thread
                0x20001b80      0x400 ./src/synergy_gen/inputs_thread.o
 .stack.lcd_thread
                0x20001f80      0x400 ./src/synergy_gen/lcd_thread.o
 .stack.led_thread
                0x20002380      0x200 ./src/synergy_gen/led_thread.o
                0x20002580              __StackTop = .
                0x20002580              PROVIDE (__stack, __StackTop)
                0x20002580                __RAM_segment_used_end__ = ALIGN (__StackTop, 0x4)
```

Figure 12. Locating thread stack addresses using the .map file

While this process works to locate the address of any variable, you must be careful when using the addresses in the memory monitor. As you make the changes during development, the program addresses may change, so it is always better to use symbols if possible.

A shortcut to this process is applying `appending _stack` to the current symbol name you gave to your thread. In this case, the thread name is `inputs_thread`, and so `appending _stack` to this name gives the name `inputs_thread_stack`. You can enter this symbol name into the Memory Monitor dialog box shown in Figure 13. The advantage to using the symbol is that it stays the same as you add/subtract code from your project, unless you explicitly change the thread name.



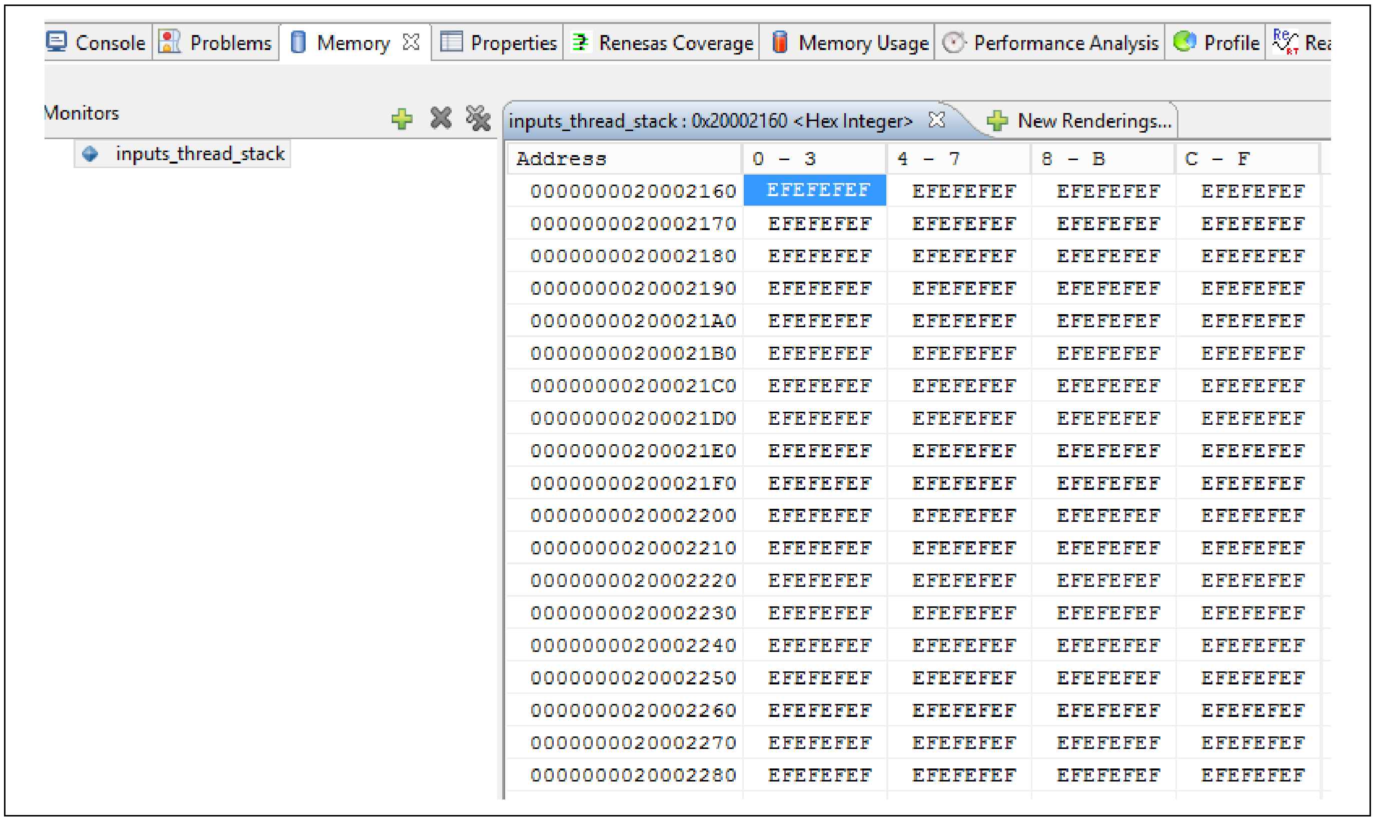
```
.data_flash      0x40100
                 0x40100
 *(.data_flash*)
                 0x40100
OUTPUT(dk_s124_oob_v1.e
```

Enter address or expression to monitor: inputs_thread_stack

Figure 13. Setting up a memory monitor

Figure 13 shows that there is an amount of unused memory in the accelerometer thread stack. Each row represents 16 bytes. Take the number of rows displaying the EFEFEFEF data and multiply that number by 16 to see how many unused bytes exist in the stack. You can reduce the stack size by half that amount without any problem. You can always monitor the stack usage after the program has run for a while to monitor the free stack space.

Saving a few hundred bytes of stack space might not seem like a lot, but when multiplied by the number of threads in your program, the difference might allow one more feature to be added the application.



```
Address            0 - 3     4 - 7     8 - B     C - F
0000000020002160   EFEFEFEF  EFEFEFEF  EFEFEFEF  EFEFEFEF
0000000020002170   EFEFEFEF  EFEFEFEF  EFEFEFEF  EFEFEFEF
0000000020002180   EFEFEFEF  EFEFEFEF  EFEFEFEF  EFEFEFEF
0000000020002190   EFEFEFEF  EFEFEFEF  EFEFEFEF  EFEFEFEF
00000000200021A0   EFEFEFEF  EFEFEFEF  EFEFEFEF  EFEFEFEF
00000000200021B0   EFEFEFEF  EFEFEFEF  EFEFEFEF  EFEFEFEF
00000000200021C0   EFEFEFEF  EFEFEFEF  EFEFEFEF  EFEFEFEF
00000000200021D0   EFEFEFEF  EFEFEFEF  EFEFEFEF  EFEFEFEF
00000000200021E0   EFEFEFEF  EFEFEFEF  EFEFEFEF  EFEFEFEF
00000000200021F0   EFEFEFEF  EFEFEFEF  EFEFEFEF  EFEFEFEF
0000000020002200   EFEFEFEF  EFEFEFEF  EFEFEFEF  EFEFEFEF
0000000020002210   EFEFEFEF  EFEFEFEF  EFEFEFEF  EFEFEFEF
0000000020002220   EFEFEFEF  EFEFEFEF  EFEFEFEF  EFEFEFEF
0000000020002230   EFEFEFEF  EFEFEFEF  EFEFEFEF  EFEFEFEF
0000000020002240   EFEFEFEF  EFEFEFEF  EFEFEFEF  EFEFEFEF
0000000020002250   EFEFEFEF  EFEFEFEF  EFEFEFEF  EFEFEFEF
0000000020002260   EFEFEFEF  EFEFEFEF  EFEFEFEF  EFEFEFEF
0000000020002270   EFEFEFEF  EFEFEFEF  EFEFEFEF  EFEFEFEF
0000000020002280   EFEFEFEF  EFEFEFEF  EFEFEFEF  EFEFEFEF
```

Figure 14. Displaying stack memory with the memory monitor

3.11 Importing and Building the Project

Follow the procedure in the included *Renesas Synergy Project Import Guide* to import the project into the e^2 studio ISDE, and then build and debug the project. For the debugging configuration, select **dk_s124_oob Debug** (under **Renesas GDB Hardware Debugging**).

4. Next Steps

After you run the example application, you can learn more about how the application works and the API calls involved by examining the application source code.

You can download additional Synergy example applications from the following URL: www.renesas.com/products/embedded_systems_platform/synergy/Application_Notes.jsp

5. References

Renesas Synergy Project Import Guide (r11an0023eu0121-synergy-ssp-import-guide.pdf).

Website and Support

Visit the following vanity URLs to learn about key elements of the Synergy Platform, download components and related documentation, and get support.

Synergy Software	www.renesas.com/synergy/software
Synergy Software Package	www.renesas.com/synergy/ssp
Software add-ons	www.renesas.com/synergy/addons
Software glossary	www.renesas.com/synergy/softwareglossary
Development tools	www.renesas.com/synergy/tools
Synergy Hardware	www.renesas.com/synergy/hardware
Microcontrollers	www.renesas.com/synergy/mcus
MCU glossary	www.renesas.com/synergy/mcuglossary
Parametric search	www.renesas.com/synergy/parametric
Kits	www.renesas.com/synergy/kits
Synergy Solutions Gallery	www.renesas.com/synergy/solutionsgallery
Partner projects	www.renesas.com/synergy/partnerprojects
Application projects	www.renesas.com/synergy/applicationprojects
Self-service support resources:	
Documentation	www.renesas.com/synergy/docs
Knowledgebase	www.renesas.com/synergy/knowledgebase
Forums	www.renesas.com/synergy/forum
Training	www.renesas.com/synergy/training
Videos	www.renesas.com/synergy/videos
Chat and web ticket	www.renesas.com/synergy/resourcelibrary

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Sep.23.16	-	Initial version
1.10	Oct.20.16	-	Migrated to SSP 1.2.0-b.1
1.11	Nov.30.16	-	Added support for IAR EW
1.12	Feb.21.17	-	Added support for SSP v1.2.0
1.13	Jul.10.17	-	Added support for SSP v1.3.0
1.14	Aug.04.17	-	Initial release
1.15	Feb.09.18	-	Updated for SSP v1.4.0
1.16	Mar.07.19	-	Updated for SSP v1.6.0