Develop an RTOS Based Application in Less Than 30 Minutes

Device: ARIS EDGE

Description:

The purpose of this lab is to use the SSP (Synergy Software Package) to create a ThreadX RTOS based application utilizing a thread and a semaphore to signal a push button event and toggle an LED, all within 30 minutes or less.

Lab Objectives

1. Create a thread.
2. Use a semaphore to signal a push button event.

Lab Materials

Please verify that you have the following materials at your lab station:

1. e2studio ISDE v.5, SSP v.1.3.0
2. ARIS EDGE Board
3. Mini USB cable for J-Link
4. TraceX 5.2.0 Renesas Synergy

Skill Level

Programming in C

Time to Complete Lab

30 minutes

Lab Sections:

1 Setup LED Toggle Thread

Overview:

In this section of the lab we will create a new Synergy project using the ISDE and add a thread and a semaphore to the project.

When using an RTOS to create an application the application is broken down into semi-independent program segments called threads. Each thread typically controls one aspect of an application. For example in the application we are creating the first thread is associated with toggling an LED. A thread has its own stack space and a priority with respect to the other threads in the application.

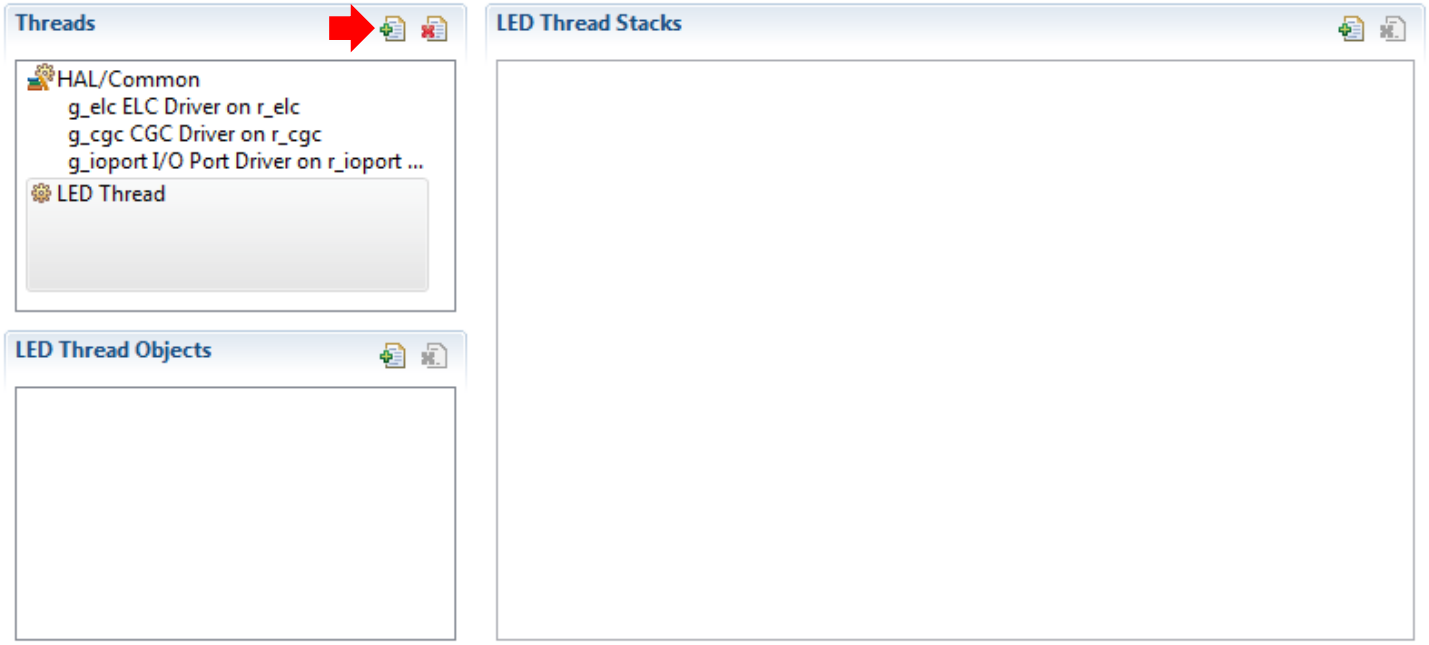
A semaphore is an RTOS resource which can be used for event signalling and thread synchronization. In this application the semaphore will be used to signal to a thread that a hardware switch on the board has been pressed. Using a semaphore in this way means a thread can be suspended until the event occurs and the semaphore is posted. In a non-RTOS system it would be necessary for constant polling of a flag variable or placing code responding to the interrupt in the ISR. Using a semaphore allows the ISR to exit quickly and the LED code execution to be deferred to the thread.

Creating a new Project

1. Launch the Renesas Synergy ISDE
   (if needed select the workspace location somewhere without spaces in the path and folder name)
2. In the ISDE select File → New → **Synergy C Project**
3. Enter a name for the project **arisedge_rtos_lab**.
   (if the license file isn't specified select the license file)
4. Click **Next**
5. Select the SSP version 1.3.0, board **aris_edge1** (the device will be selected automatically for the board), toolchain version 4.9.3.20150529 and J-Link ARM debugger.
6. Click **Next**
7. Select **BSP**. Selecting the BSP version of the project for the Aris board will include configuration of the clocks, IO pins etc.
8. Click **Finish**
9. The Synergy configurator will open on the summary screen. Switch to the **Threads** tab.

Adding LED Thread to the Configuration File

1. Create a new thread named “LED Thread” with symbol name “led_thread”



Property	Value
Thread	
Symbol	led_thread
Name	LED Thread
Stack size (bytes)	1024
Priority	1
Auto start	Enabled
Time slicing interval (ticks)	1

Adding External Interrupt Driver to LED Thread

The application will toggle an LED in response to pushing SW1 on the ARIS EDGE board. SW1 on the board is connected to the external interrupt IRQ6. The external interrupt can be configured and used via the SSP module External IRQ.

1. Add a new IRQ Driver to LCD Thread. **New Stack → Driver → Input → External IRQ Driver on r_icu**.

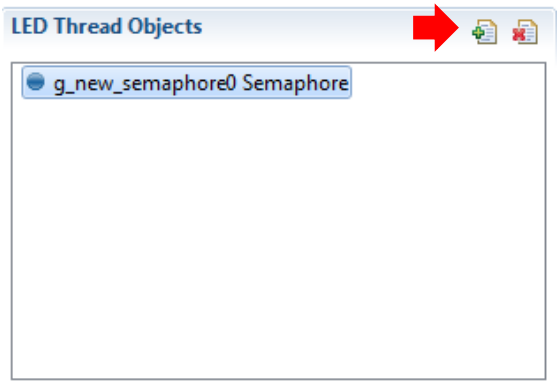
2. The properties of the IRQ Driver are configured as following:

Property	Value
◢ Common	
Parameter Checking	Default (BSP)
◢ Module g_external_irq0 External IRQ Driver on r_icu	
Name	g_external_irq0
Channel	6
Trigger	Falling
Digital Filtering	Enabled
Digital Filtering Sample Clock (Only valid when D	PCLK / 64
Interrupt enabled after initialization	True
Callback	irq6_callback
Interrupt Priority	Priority 6 (CM4: valid, CM0+: invalid)

The step will create a function (**irq6_callback**) which is called when SW1 is pressed. We will add code to this callback later.

Adding Semaphore to LED Thread

1. Click the “New” button on the right of the “LED Thread Objects” pane and select **New → Semaphore**



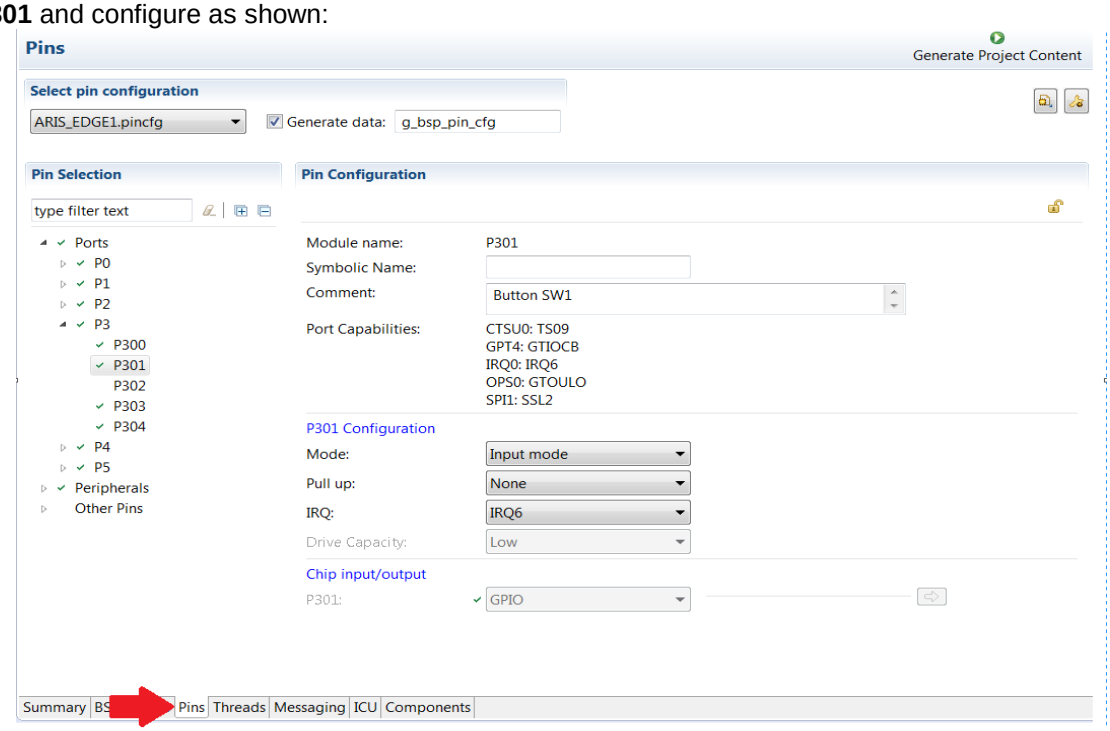
2. Set the properties of this semaphore as shown below:

Property	Value
Name	LED Semaphore
Symbol	g_new_semaphore0
Initial count	0

This semaphore will be used to signal SW2 being pressed so its initial count value should be left as zero.

Configuring SW2 IRQ pin as Input

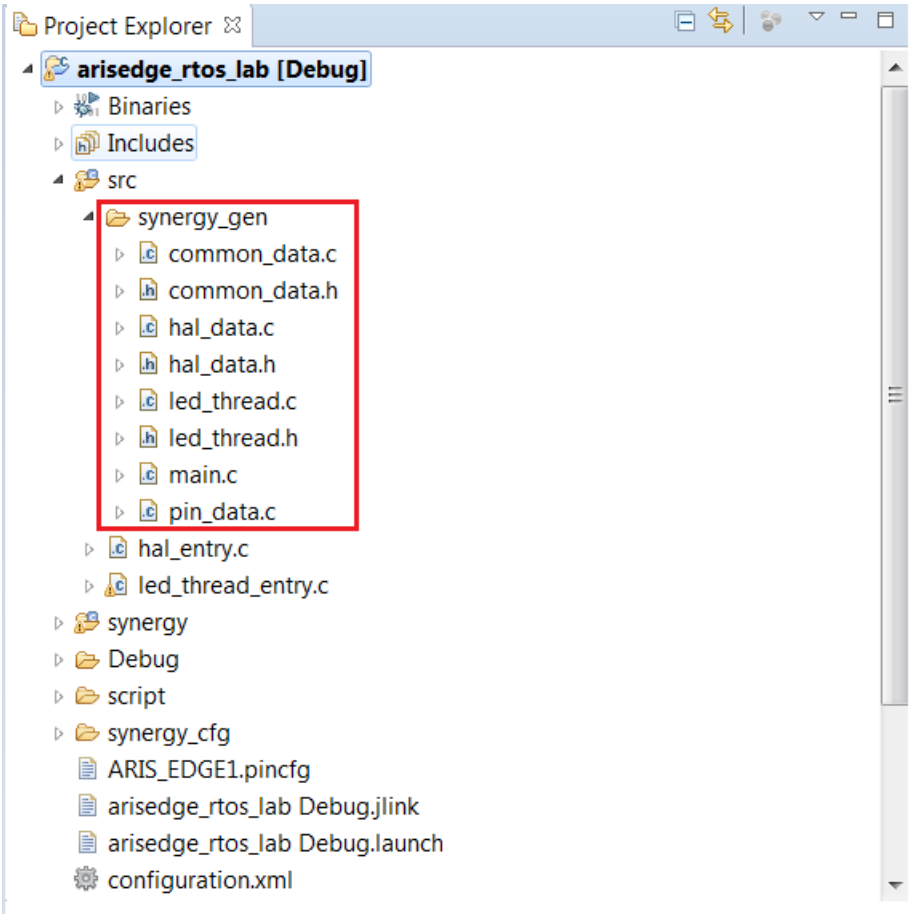
1. The IO pin connected to SW1 must be configured as the IRQ6 input. Select the **Pins** tab and expand **Ports → P3 → P301** and configure as shown:



Adding application code to LED Thread

SSP configuration is now complete. Press the “Generate Project Content” button: 

The following file and folder structure will be created:



The files in the “synergy_gen” folder shown in the red box are rewritten each time “Generate Program Content” is pressed. Therefore, do not edit these files as any changes will be overwritten. **User code should be added to the files “hal_entry.c” and “led_thread_entry.c”, which are not overwritten when generating project content.**

1. **Edit the source file “led_thread_entry.c” so it contains the code found at the next page of this manual.**

Prototypes for any callback functions created by the ISDE and SSP can be found in the HAL/Thread code in the “synergy_gen” folder. For example there is no need to create the callback “irq6_callback()” from scratch as it can be copied from led_thread.h.

2. Build the project via either the menu Build → Build Project or by using the **Hammer button** 

```
text     data          bss          dec          hex      filename
21260     124        10068        31452         7adc      aris_rtos_lab.elf
'Finished building: aris_rtos_lab.srec'
'Finished building: aris_rtos_lab.siz'
```

3. If the build is successful, program and run the project using the **Debug button** 

Congratulations! You have created an RTOS based application! Press button SW2 on the Aris Board to toggle blue LED.

4. Resume execution using F8 or the **Resume button** 
5. Terminate the debugging session before continuing with the tutorial using the **Stop button** 

led_thread_entry.c

```
#include "led_thread.h"

/* LEDs struct provided by BSP */
bsp_leds_t Leds;

void led_thread_entry(void)
{
    /* Storage of LED0 on the board's output level */
    ioport_level_t led_0_level = IOPORT_LEVEL_HIGH;

    /* Populate the Leds structure array to simplify the use of the LEDs on the board. */
    /* No need to reach for the schematic. */
    R_BSP_LedsGet(&Leds);

    /* Open and configure the external IRQ pin connected to SW1 on the board. */
    g_external_irq0.p_api->open(g_external_irq0.p_ctrl,  g_external_irq0.p_cfg);

    while (1)
    {
        /* Output the current output level to the LED0 connected pin. */
        g_ioport.p_api->pinWrite(Leds.p_leds[0], led_0_level);

        /* Toggle the pin level */
        if (led_0_level == IOPORT_LEVEL_HIGH) {
            led_0_level = IOPORT_LEVEL_LOW;
        } else {
            led_0_level = IOPORT_LEVEL_HIGH;
        }

        /* Wait forever for the semaphore from the IRQ6 ISR callback to be posted. */
        /* RTOS will suspend this task until this event occurs. No need for polling. */
        tx_semaphore_get(&g_new_semaphore0, TX_WAIT_FOREVER);
    }
}

/* Callback function called by the external IRQ6 ISR. */
/* Code within ISR context should be kept as quick as possible. */
void irq6_callback(external_irq_callback_args_t  * p_args)
{
    /* Post to the semaphore to indicate SW2 has been pressed. */
    tx_semaphore_put(&g_new_semaphore0);
}
```