RENESAS
BIG IDEAS FOR EVERY SPACE

BASICS OF THE RENESAS SYNERGY™ PLATFORM

Richard Oed

2020.04

CHAPTER 10
SENDING DATA THROUGH USB USING A QUEUE

CONTENTS

10 SENDING DATA THROUGH USB USING A QUEUE ____ 03
10.1 Setting Up a USB Port in Synergy Platform ____ 03
10.2 Sending Messages ____ 05
10.3 Setting Up a Receiver On the Host Side ____ 06
Disclaimer ____ 08

10 SENDING DATA THROUGH USB USING A QUEUE

What you will learn in this chapter:

- How to setup an USB-transfer using one of the Renesas Synergy™ Application Frameworks and how to receive the data on a host workstation.

In this chapter, we will use the communication framework inside the Application Frameworks to send the state of LED1 through an USB-port to the Windows® workstation. For this, you will to add a new thread and a queue to your project from the previous chapter 9 and modify the existing LED thread to send the state as a string to the queue. The new communications thread will then send the string to a terminal program on the host using USB. While programming this exercise, you will experience once again the simplicity the Synergy Software Package (SSP) provides to users, even when setting up complex communications like USB.

If you haven't done the exercise from chapter 9, you can download the project from the book's website (**www.renesas.com/synergy-book**). Import it according to the instructions given in chapter 5.1.3 and you are all set. Of course, you can download a full project for this chapter as well, if you just want to see how the goal is achieved. If you are using IAR Embedded Workbench® for Renesas Synergy™ (IAR EW for Synergy), you can also download a complete project, if you do not want to follow the instructions below and adjust them as necessary.

10.1 Setting Up a USB Port in Synergy Platform

If you closed e² studio after the last exercise, re-open it and make sure that your project *MyRtosProject* is active. If not, just click on it and it will be displayed in bold. The first step for you is to switch to the *Synergy Configuration* perspective and to go to the *Synergy Configuration [MyRtosProject]* view. If this view was closed before, you can re-open it by double clicking on the *configuration.xml* file in the *Project Explorer* view, or by clicking on the *Gear* icon on the main menu bar. As you are a Synergy expert by now, I will not describe every step in detail, as it was already detailed in a previous exercise.

Select the *Threads* tab and add a new thread with the symbol *comms_thread* and the name *Comms Thread*. Also change the priority from the default *1* to *2*. Having all threads running at the same priority is something to be avoided. With the newly added thread selected, add a communications framework instance to it. Click on the *New Stack* icon in the *Comms Thread Stack* pane and select *New Stack → Framework → Connectivity → Communications Framework on sf_el_ux_comms_v2*.

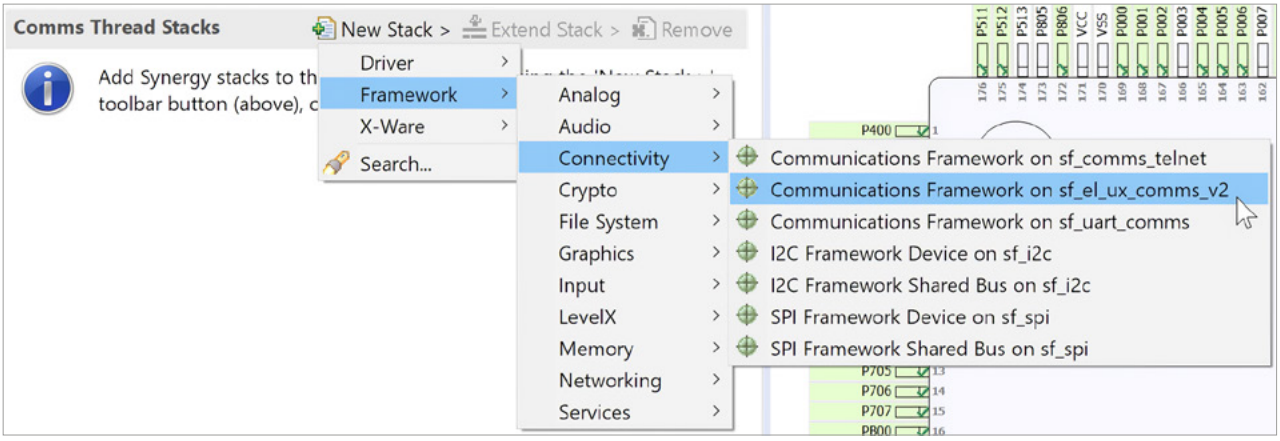

Figure 10-1: First step is to add the communications framework

This will add the complete stack of the *g_sf_comms0 Communications Framework on sf_el_ux_comms_v2* to the system, down to alevel where a user intervention is necessary. You might wonder what the meaning of different colour bars of the thread modules is. Itis quite simple: Regular instances are marked in grey, common instances in blue (there is just one global per project) and pink marks options. And the small triangles in the colour bars let you expand or collapse the module trees.

For our project, one USBX port is needed. To add it, click on the optional *Add USBX Port DCD* module and select *New* and *USBX Port DCD on sf_el_ux for USBFS*. In the Properties for this module, change the *Full Speed Interrupt Priority* property to e.g. *Priority 8* (see Figure 10-2).

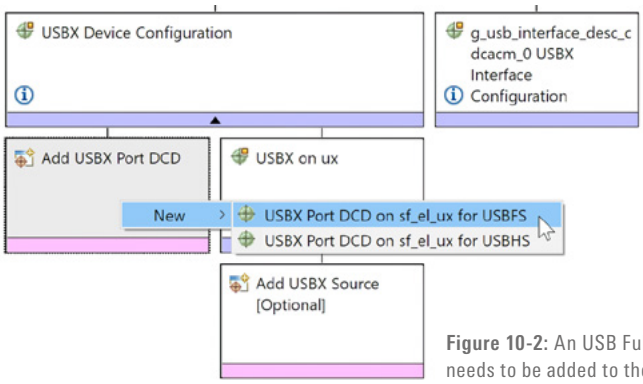

Figure 10-2: An USB Full-Speed port needs to be added to the system

You might also want to have a look at the *USBX Pool Memory Size* in the properties of the *USBX on ux* module. It default to 18 kBytes, which is sufficient for most tasks. If you want to use different transfer classes in the future, you have to adjust the pool size accordingly. The USBX documentation, which can be downloaded from the SSP-page on the Solutions Gallery has all the details for that.

With the *Comms Thread* still selected, add the queue we need for sending the data in the *Comms Thread Objects* pane. Name it *CDC Queue* and assign it the symbol `g_cdc_queue` in the Properties view. Also set the *Message Size* to 3, as we want to transmit 3 words with 4 bytes each in each transfer and the *Queue Size* to 24 bytes, meaning that the queue has space for two (12-byte) messages.

With this, the configuration of the SSP is complete. Save the configuration and click on the *Generate Project Content* button. Note that the file *comms_thread_entry.c* has been added to the *src*-folder of your project. You will add code to this file during the next steps.

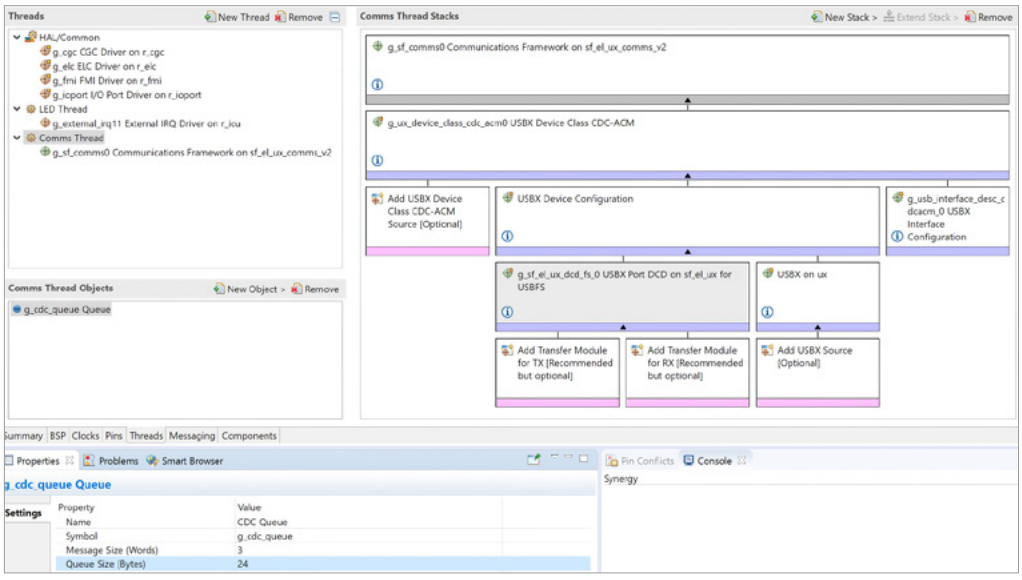

Figure 10-3: With all additions and modifications made, the Threads tab should look like this

10.2 Sending Messages

But before you add the code needed to transfer the state of the LED to the host work-station, you will have to add code to the LED Thread to actually copy the message into the queue. For this, switch back to the *C/C++ Perspective* and open the file *led_thread_entry.c*. At the top, add an include directive for the file *comms_thread.h* to share elements, like the `g_cdc_queue`, from the Comms Thread with the code in the LED Thread. As second step, add a global array of the type `char` named `send_str` with 12 characters. And finally, add the following lines just after the ioport write statement inside the `while(1)` loop in this file:

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

if(led_level == IOPORT_LEVEL_HIGH)
{
    strcpy(send_str, "LED off\n\r");
    led_level = IOPORT_LEVEL_LOW;
}
else
{
    strcpy(send_str, "LED on\n\r");
    led_level = IOPORT_LEVEL_HIGH;
}

/* Send the message in the queue. Wait forever for space */
/* to be available in the queue for the message.         */
tx_queue_send(&g_cdc_queue, send_str, TX_WAIT_FOREVER);

tx_semaphore_get(&g_sw4_semaphore, TX_WAIT_FOREVER);
```

The final step is now to add code to the *comms_thread_entry.c* file. Open it from the *Project Explorer*.

Edit the *comms_thread_entry.c* file and add a global array of 12 elements of the type `uint8_t` named `rx_msg`. After that, replace the line `tx_thread_sleep(1);` inside the `while(1)` loop with the following:

```
tx_queue_receive(&g_cdc_queue, rx_msg, TX_WAIT_FOREVER);

g_sf_comms0.p_api->write(g_sf_comms0.p_ctrl,
                         rx_msg,
                         strlen(rx_msg),
                         TX_WAIT_FOREVER);
```

This concludes all the coding necessary. You may ask now: "Wait a minute: Two lines of code is all what's needed to transfer a string received by a queue over USB? Isn't that too easy?" And the answer is: No, it is so easy. The SSP and the Application Frameworks will take care about everything else. Isn't that really simple. Remember the last time you coded a transfer on another platform? I would guess, there was a huge difference in the effort needed for that.

Now, all what is left, is to build the project. The first time you do that, it will take some time, as the code for the communications framework will need to be compiled as well. Once the project is built with zero errors, connect the S5D9 Promotion Kit and start the debug session. With the *Debug* perspective open, click on *Resume* twice to start the program. As a first test, press SW4 once to see if LED1 still toggles.

10.3 Setting Up a Receiver On the Host Side

With the program running, connect a second USB type A to micro B cable to the USB port labeled J5 at the bottom of the Promotion Kit. Insert the other end into your Windows® workstation and wait a couple of seconds until Windows® recognizes the board and installs the drivers for it.

Start a terminal emulator program. During the development of this exercise, we used Tera Term Pro, which can be downloaded from **https://ttssh2.osdn.jp/** and found it quite useful. In Tera Term, you will see the CDC serial port listed. In Figure 10-4 it is COM6, but it is likely to be different on other workstations. If you are not sure, use the Device Manager of Windows® to find out the port the board is connected to.

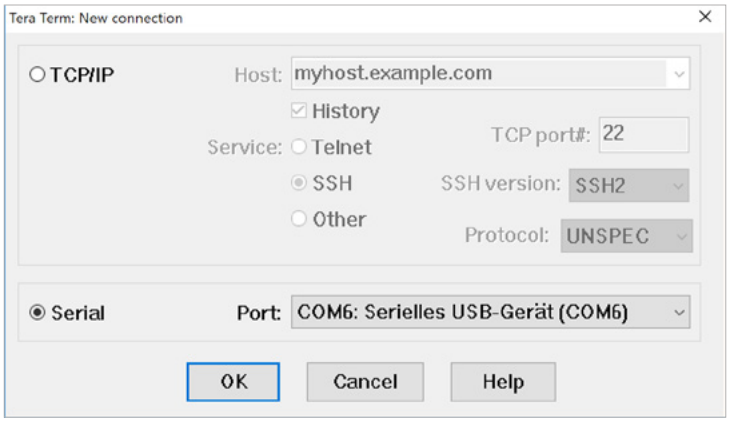

Figure 10-4: If Windows® recognized the board correctly, it will be listed in Tera Term as serial connection

In case, the board is not listed at all or the Device Manager indicates an error, there might be a problem with the driver. Please refer to the latest support entry for this topic in the Renesas Synergy™ Knowledge Base to resolve this: **https://en-support.renesas.com/knowledgeBase/18959077**.

With the connection made and Tera Term running, press SW4 a couple of times and you should see the LED1 toggling and the state of it output to the terminal as shown in Figure 10-5.

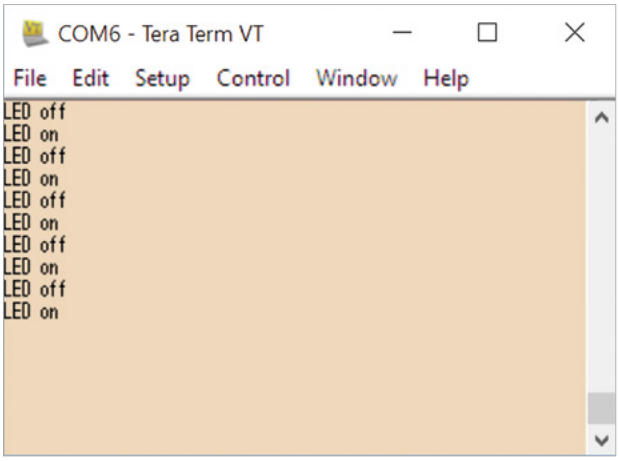

Figure 10-5: With the transfer running, the terminal program will display the state of LED1 each time SW4 is pressed

CONGRATULATIONS!

You just finished this exercise. And what do you think now: Was this easy or not?

Points to take away from this chapter:

- Adding an USB transfer is easy if the Application Frameworks are used.
- Only very few lines are needed for the implementation.