User's Manual

Renesas Synergy™ Software Package (SSP) v1.5.2

User's Manual

Renesas Synergy™ Platform

Renesas Electronics
www.renesas.com

Rev.1.11 Nov 2018

Contents

Synergy Software Package

Changes to SSP v1.5.0 User's Manual for SSP v1.5.2

This section lists the differences between the SSP v1.5.0 and SPP v1.5.2. Take the following changes into account when using SSP v1.5.2.

The updated portion of the text is underlined. Changes made for SSP v1.5.1 are shown in blue, changes made for SSP v1.5.2 are green.

1. NetX and NetX Duo Source Module Overview

Refer to: 5.3.8.5 Configuring the NetX and NetX Duo Source Module, ARP Cache size in Bytes – default value 512

ARP cache storage units – default in Bytes – Defines the ARP storage type in Bytes. User can also select the option in Entries to define the number of ARP entries in table which will be aligned to the size of ARP.

ARP Cache size (in storage units) – default value 520 – Defines the size of the ARP cache in bytes. If user selects the storage units in Entries, then the value should be an integer. If user selects the storage units in Bytes, then the value should be a multiple of size of ARP (52 Bytes). This table holds all the ARP entries. If a table is full, no more ARP entries can be added till existing entries 'age' (see explanation of ARP Expiration Rate in this section) and are removed. There are some configurable options listed with the NetX/Duo Source element that can affect the number of ARP entries added to the table, such as ARP Auto ARP Entry and ARP Expiration Rate defined in this section.

2. Messaging Framework on sf_message Module Overview

Refer to: 5.1.26.3 Messaging Framework Module Operational Overview, Message Queue Size and Depth Setting

Message Queue Size and Depth Setting

The Messaging Framework module needs a 4-byte memory block on the message queue as it delivers the pointer to the buffer which contains a message payload. For this reason, the size of the message queue is fixed to 4 bytes.

3. I2C Framework on sf_i2c Module Overview

3.1 I2C Framework Module APIs Overview

Refer to: 5.1.23.2 I2C Framework Module APIs Overview, I2C Framework Module API Summary

Function Name	Example API Call and Description
open	`g_sf_i2c_device.p_api->open(g_sf_i2c_device.p_ctrl, g_sf_i2c_device.p_cfg);` Opens a designated I2C device on the bus.
close	`g_sf_i2c_device.p_api->close (g_sf_i2c_device.p_ctrl);` Disables I2C device designated by control handle. Closes the RTOS services used by the bus if no devices are connected to the bus.
read	`g_sf_i2c_device.p_api->read (g_sf_i2c_device.p_ctrl, &destination, no_of_bytes_to_read, restart,timeout);` Receives data from I2C device.
write	`g_sf_i2c_device.p_api->write (g_sf_i2c_device.p_ctrl, &source, no_of_bytes_to_write , restart, timeout);` Transmits data to I2C device.

lock	`g_sf_i2c_device.p_api->lock (g_sf_i2c_device.p_ctrl);` Locks the bus for a device. Locking reserves the bus until unlocking and allows several reads and writes without interrupt.
unlock	`g_sf_i2c_device.p_api->unlock (g_sf_i2c_device.p_ctrl);` Unlocks the bus from a particular device and makes it available for other devices.
reset	`g_sf_i2c_device.p_api->reset (g_sf_i2c_device.p_ctrl, timeout);` Aborts any in-progress transfer and forces the I2C peripheral into ready state.
versionGet	`g_sf_i2c_device.p_api->versionGet(version);` Retrieves the version information using the version pointer.
lockWait	`g_sf_i2c_device.p_api->lockWait (g_sf_i2c_device.p_ctrl, timeout);` Locks the bus for a device. Locking reserves the bus until unlocking and allows several reads and writes without intervention from other devices on the same I2C bus. Timeout value is user configurable.

3.2 Status Return Values

Refer to: 5.1.23.2 I2C Framework Module APIs Overview, Status Return Values

Name	Description
SSP_SUCCESS	I2C function performed successfully
SSP_ERR_INVALID_MODE	Illegal I2C mode is specified
SSP_ERR_IP_CHANNEL_NOT_PRESENT	Omitted I2C channel is specified
SSP_ERR_IN_USE	I2C channel has already been opened
SSP_ERR_INVALID_ARGUMENT	Argument is not one of the predefined values
SSP_ERR_INTERNAL	Internal error has occurred
SSP_ERR_ASSERTION	A critical assertion has failed or Null pointer(s) is (are) given
SSP_ERR_NOT_OPEN	Device instance not opened
SSP_ERR_TRANSFER_ABORTED	The data transfer was aborted
SSP_ERR_INVALID_RATE	The requested rate cannot be set
SSP_ERR_TIMEOUT	Timeout error occurs.

3.3 I2C Framework Module Operational Overview

Refer to: 5.1.23.3 I2C Framework Module Operational

The I2C Framework module complies with the layered driver architecture of the SSP. It uses the lower-level I2C HAL modules to communicate with I2C peripherals and controls the I2C-capable peripherals on a Synergy microcontroller (as configured by a user). With the I2C Framework module, one or more I2C buses can be created and multiple I2C peripherals can be connected to each I2C bus. The I2C Framework module APIs use a ThreadX-Mutexto acquire and release the shared bus for I2C Slave devices. Acquire and release are implemented by lock or lockWait and unlock APIs respectively in the I2C Framework module.

3.4 Bus Locking

Refer to: 5.1.23.3 I2C Framework Module Operational Overview, Bus Locking

The I2C bus is locked when lock or lockWait API is called. This API locks the I2C bus by acquiring the mutex for the thread in which the I2C Framework device is used. Once locked, the I2C bus can only be utilized by the associated device. The other I2C Framework devices or same I2C Framework device, from other threads, can't acquire the mutex so they won't be able to access the bus. Once the bus is unlocked by calling unlock API from the sf_i2c device that locked it, the mutex will be released and the bus becomes available for other sf_i2c devices. The lockWait API is similar to the lock API except it provides the user an option to set the timeout value. The lockWait API waits for the specified timeout period if the I2C bus is already locked by another device. In the case of the lock API, the thread waits forever, if the I2C bus is not released by the other device.

3.5 I2C Framework Module Important Operational Notes and Limitations

Refer to: 4.1.23.3 I2C Framework Module Operational Overview I2C Framework Module Operational Notes

- The closest possible baud rate that can be achieved (less than or equal to the requested rate) at the current PCLKB settings is calculated and used. If a valid clock rate could not be calculated, an error is returned.
- The I2C can trigger the start of other peripherals available from the ELC. See the ELC Module Guide for further information.
- The I2C Framework can support multiple I2C devices on the same bus if the clock rate remains the same for all the devices. That means multiple devices can be opened in the same bus if they are of the same clock rate. If devices have different clock rates, only one device can be opened at a time.
- SDA and SCL output pin type should be n-channel open drain when using I2C on SCI.
- In the I2C Framework configuration, the channel number given to this bus overrides the channel number given in the HAL module.
- Shared bus can be used by multiple slave devices with the respective configuration. The framework also handles mutual exclusion in lock and unlock APIs when multiple devices are using the same I2C channel.
- To configure multiple I2C devices on the same bus, add and configure the following modules for each device connecting to the bus:
    - I2C Framework device module
    - Configure the I2C shared bus module for the first device being configured, then use the same bus for the remaining devices.
    - I2C HAL module
    - DTC module(Optional)
- Lock functionality will be effective for devices from different threads. If multiple devices connected to the bus are from the same thread, the I2C bus will be locked for all devices from that thread. In such cases, even if the bus is locked, all devices from the same thread can access the bus.
- In case a device is being used from multiple threads, and the device locks the I2C bus from one thread, the same device cannot access the I2C bus from other threads.

NOTE: Each I2C Framework device must be configured with a unique name in the ISDE configurator.

Provide the same configuration settings for all the devices connected on the same bus (except the slave address and addressing modes.)

I2C Framework Module Operational Notes

The I2C framework module does not currently support the following feature: The use of DMA

Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

3.6 Pin Configuration Settings for the I2C Framework Module

Refer to: 5.1.23.5 Configuring the I2C Framework Module, Pin Configuration Settings for the I2C Framework Module

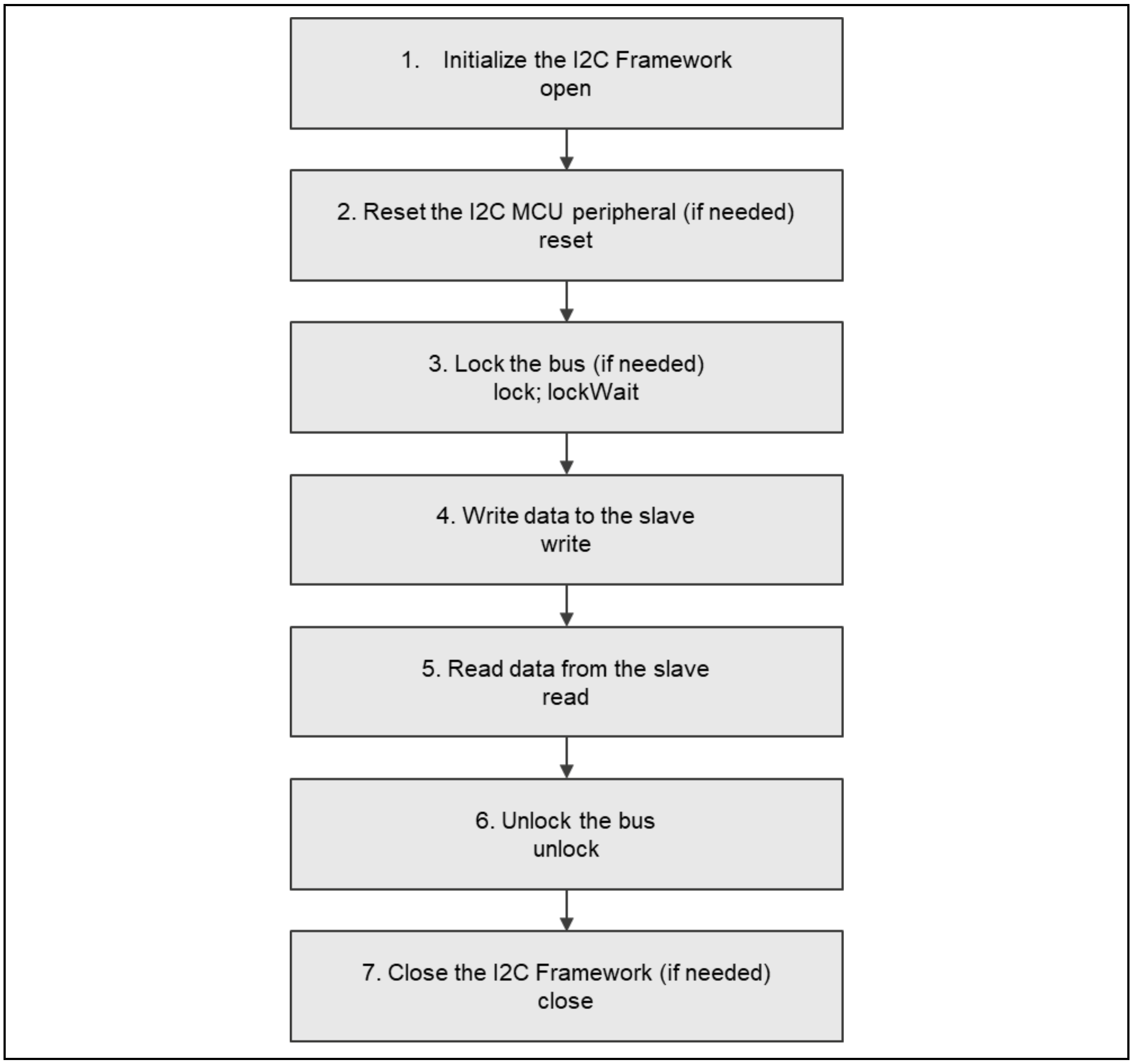
Pin Configuration Property	Value	Description
Operation Mode	Disabled, Asynchronous UART, Synchronous UART, Simple I2C, Simple SPI, SmartCard Default: Disabled	Select Simple I2C as the Operation Mode for I2C on SCI
RXD1_SCL1_MISO1	None, P212, P708 Default: None	SCL Pin
TXD1_SDA1_MOSI1	None, P213, P709 Default: None	SDA Pin

3.7 Pin Configuration Settings for the I2C Framework Module

Refer to: 5.1.23.6 Using the I2C Framework Module in an Application

1. Initialize the I2C Framework module using the open API. Each I2C framework module needs to call open API at least once before performing any operations on the bus.
2. Reset the I2C MCU peripheral using the reset API (if needed)
3. Lock the bus using the lock or lockWait API for a particular framework module. Once the bus is locked by an I2C framework module it cannot be used by any other I2C framework module on the same bus. This ensures that ownership of the bus remains with the I2C framework module until it unlocks it. Any operation from other I2C framework modules on the bus will fail while the bus is locked. It is not mandatory to lock the bus before any read/write operations on the bus. It is optional. (if needed). If thread is not supposed to wait forever when locking the I2C bus, call lockWait API with desired timeout value.
4. Write data to the slave using the write API. The write operation will not be successful if the bus is already locked by any other I2C framework module.
5. Read data from the slave using read API. The read operation will not be successful if the bus is already locked by any other I2C framework module.
6. Unlock the bus using the unlock API if it is already locked by the same I2C framework module. Once the bus is unlocked other I2C framework modules can use it. It is necessary to unlock the locked bus after the protected read or write operations are over. (if needed)
7. Close the I2C framework module using the close API. Each I2C framework module can call the close API after all its read and write operations on the bus are completed. (If needed)

These common steps are illustrated in a typical operational flow in the following figure:


1. Initialize the I2C Framework
open
2. Reset the I2C MCU peripheral (if needed)
reset
3. Lock the bus (if needed)
lock; lockWait
4. Write data to the slave
write
5. Read data from the slave
read
6. Unlock the bus
unlock
7. Close the I2C Framework (if needed)
close

4. UART Driver on r_sci_uart Module Overview

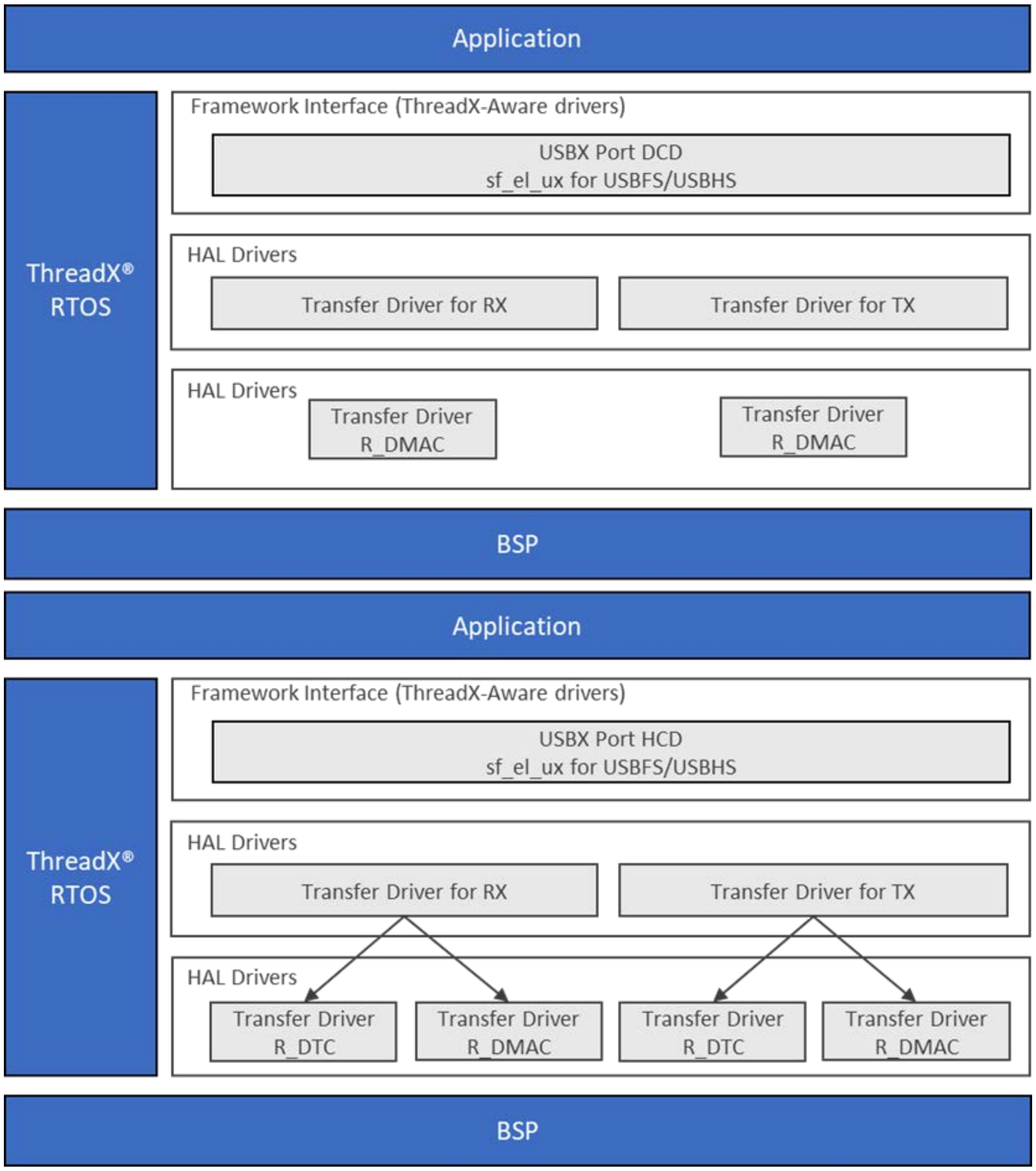
Refer to: 5.2.44.5 Configuring the UART HAL Module, Configuration Settings for the UART HAL Module on r_sci_uart

Receive FIFO Trigger Level	One, Max Default: Max	Receive FIFO trigger level selection: One: an interrupt occurs for every byte received. Max: an interrupt will be triggered if either of the below conditions are met: a) The FIFO is filled to the Max level (15). b) 15bit times have occurred with no data received.

5. USBX Synergy Port Framework Module Overview

5.1 USBX Synergy Port Framework Module Block Diagram

Refer to: 5.3.37.1 USBX Synergy Port Framework Module Features, Figure 491: USBX Synergy Port Framework Module Block Diagram



5.2 USBX Synergy Port Framework Module Operational Notes

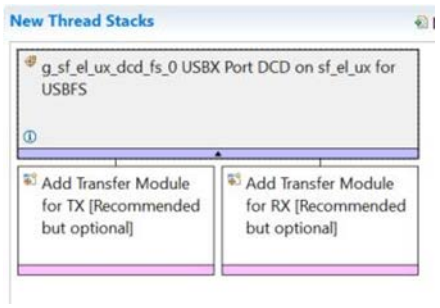
Refer to: 4.3.37.3 USBX Synergy Port Framework Module Operational Overview, USBX Synergy Port Framework Module Operational Notes

Users have an option to select the Transfer Module for the USBX Synergy Port framework module to improve data throughput. Transfer Modules can be selected to use DMA on Synergy S3, S5 and S7 MCU series devices. Transfer Modules should be removed on S1 MCU series devices so that CPU transfer (which is automatically implemented when the optional Transfer Modules are not added) are used. The Synergy Configuration tool auto-generates the driver setup code to enable DMAC transfer or CPU transfer in common_data.c.

5.3 USBX Synergy Port Framework Module Limitations

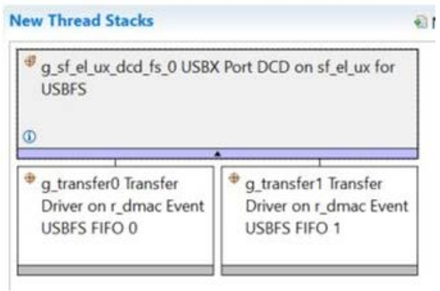
Refer to: 5.3.37.3 USBX Synergy Port Framework Module Operational Overview, USBX Synergy Port Framework Module Limitations

- Synergy USB controllers (USBHS and USBFS) have a limited number of PIPEs you can use for the isochronous transfer type (PIPE1 and PIPE2). This will limit the number of UVC devices (two devices) you can connect to the synergy board configured as UVC HOST.
- The device side driver (sf_el_ux DCD driver) does not support DTC as the transfer interface. A compile error can result if it is selected.
  - On Synergy S1 MCU series, the DTC transfer modules should be removed, and the CPU will be used as the transfer agent. The resulting thread is illustrated below.


New Thread Stacks
g_sf_el_ux_dcd_fs_0 USBX Port DCD on sf_el_ux for USBFS
Add Transfer Module for TX [Recommended but optional]
Add Transfer Module for RX [Recommended but optional]

Transfer Module Stack- No DTC

  - On Synergy S3, S5 and S7 MCU series, the DTC transfer modules should be changed to use DMA, as shown in the below diagram.


New Thread Stacks
g_sf_el_ux_dcd_fs_0 USBX Port DCD on sf_el_ux for USBFS
g_transfer0 Transfer Driver on r_dmac Event USBFS FIFO 0
g_transfer1 Transfer Driver on r_dmac Event USBFS FIFO 1

Transfer Module Stack- DMA

6. JPEG_Encode module usage notes

6.1 JPEG Encode HAL Module Features

Refer to: 5.2.29.1 JPEG Encode HAL Module Introduction, JPEG Encode HAL Module Features

- Supports JPEG Compression.
- Supports polling mode that allows an application to wait for JPEG Encoder to complete.
- Supports interrupt mode with user-supplied callback functions.
- Configures parameters such as horizontal and vertical resolution, horizontal stride, and Quality factor.
- Supports putting raw image data in an input buffer and an output buffer to store the encoded/compressed jpeg image.
- Supports streaming raw image data into JPEG Encoder module. This feature allows an application to read coded raw image from a capture device or camera or from network without buffering the entire image.

- Only supports the YCbCr422 color space with Luma (Y) unsigned, and Chroma-subsample (Cr and Cb) as 8-bit signed values.
- Supports DRI Maker for RTP streaming application.
- Supports quality factor configuration: The quality factor value determines the quality of output image.

6.2 JPEG Encode HAL Module APIs Overview

Refer to: 5.2.29.2 JPEG Encode HAL Module APIs Overview, JPEG Encode HAL Module API Summary

Function Name	Example API Call and Description
open	`g_jpeg_encode0.p_api->open(g_jpeg_encode0.p_ctrl, g_jpeg_encode0.p_cfg);` Initial configuration.
imageParameterSet	`g_jpeg_encode0.p_api->imageParameterSet(g_jpeg_encode0.p_ctrl, p_image_parameters);` Set image parameters to JPEG Codec.
outputBufferSet	`g_jpeg_encode0.p_api->outputBufferSet(g_jpeg_encode0.p_ctrl, p_buffer);` Assign output buffer to JPEG Codec for storing output data.
inputBufferSet	`g_jpeg_encode0.p_api->inputBufferSet(g_jpeg_encode0.p_ctrl, p_buffer, buffer_size);` Assign input data buffer to JPEG Codec.
statusGet	`g_jpeg_encode0.p_api->statusGet(g_jpeg_encode0.p_ctrl, p_status);` Retrieve current status of the JPEG Codec module.
close	`g_jpeg_encode0.p_api->close(g_jpeg_encode0.p_ctrl);` Cancel an outstanding operation.
versionGet	`g_jpeg_encode0.p_api->versionGet(&version);` Get version and store it in the provided pointer version.

6.3 JPEG Encode HAL Module Operational Overview

Refer to: 5.2.29.3 JPEG Encode HAL Module Operational Overview, Normal Operational Description

Normal Operational Description

In this mode raw image data arrives from the network, file or capturing device as a complete frame. The HAL-layer driver handles this mode and can compress the entire raw image frame.

Note: JPEG Codec requires RAW image in YCbCr422 (Y 8-bit unsigned: Cr, Cb 8-bit signed integer). It is the users responsibility to provide a proper image to the Codec. An invalid image or out of range sub-sample values of Cr and Cb may result in an undesired color jpeg image.

7. Wi-Fi Framework module usage notes

7.1 NetX/NetX-Duo Driver Function

Refer to: 5.1.34.2 WiFi Framework Module APIs Overview, NetX/NetX-Duo Driver Function

NetX/NetX-Duo Driver Function

The NetX/NetX-Duo driver function takes the NetX IP instance, Wi-Fi framework instance and NSAL configuration as arguments. The NSAL configuration controls the behavior of transmit and receive callback functions. The NSAL configuration includes flags which indicates zero-copy support is enabled or disabled in transmit and receive path. The NetX/NetX-Duo driver functions implement various IP driver commands used by NetX/NetX-Duo. The interface attach command calls the Wi-Fi framework open API to initialize the Wi-Fi module. The initialize command calls the Wi-Fi framework macAddressGet API to read MAC address from the Wi-Fi module. The multicast-join command calls the Wi-Fi framework multicastListAdd API to add the given MAC address to multicast list. The multicast-leave command calls the Wi-Fi framework multicastListDelete API to delete the given MAC address from the multicast list. The Send/Broadcast command calls the Wi-Fi framework transmit API to transmit a packet.

7.2 Wi-Fi Framework Module API Use Notes

Refer to: 5.1.34.3 WiFi Framework Module Operational Overview, Wi-Fi Framework Module API Use Notes, Close

Close:

For successful de-initialization of the module, the application should call the Wi-Fi framework module close API explicitly from an application thread. When using the Wi-Fi framework module with NSAL i.e. with NetX, the application should use the following de-initialization sequence,

- Call the Wi-Fi framework module close API
- Call the nx_ip_delete() API

8. NetX Port module usage notes

Refer to: 5.3.7.3 NetX Port Ether Module Operational Overview, NetX Port Ether Module Important Operational Notes and Limitations

NetX Port Ether Module Operational Notes

Refer to the NetX User Guide for the Renesas Synergy™ Platform and NetX Duo User Guide for the Renesas Synergy™ Platform for more details on each of these topics.

NetX Source Properties

NOTE: There are two ways to modify NetX source properties- using the source code property directly in the source element or defining the source code symbol directly. For example, to change the number of physical network interfaces, one can either set the Maximum Physical Interfaces property in the NetX Source or NetX Duo Source element, or one can define the source code symbol NX_MAX_PHYSICAL_INTERFACES directly. In either case, it is still necessary to include the NetX and NetX Duo Source component, generate the project files and to rebuild the NetX library.

TCP Options Field Parameters

Maximum Segment Size

The Maximum Segment Size (MSS) is the maximum amount of bytes a TCP host can receive without being fragmented by the underlying IP layer. During TCP connection establishment phase, both ends exchanges its own TCP MSS value, so that the sender does not send a TCP data segment that is larger than the receiver’s MSS. NetX TCP module will optionally validate its peer’s advertised MSS value before establishing a connection. By default NetX does not enable such a check.

The nx_tcp_socket_mss_set() API sets a specified socket's Maximum Segment Size (MSS). The MSS value must be within the network interface IP Maximum Transfer Unit (MTU), allowing room for IP and TCP headers. This service should be used before a TCP socket starts the connection process. If the service is used after a TCP connection is established, the new value has no effect on the connection. To retrieve the MSS value use the nx_tcp_socket_mss_get() API after the TCP connection is established.

Additional Information

Refer to the NetX User Guide for the Renesas Synergy™ Platform and NetX Duo User Guide for the Renesas Synergy™ Platform for additional operational details.

9. Cellular Framework Module usage notes

9.1 Cellular Framework Error Codes

Refer to: 5.1.12.2 Cellular Framework Module APIs Overview, Cellular Framework Error Codes

Error Codes	Description
SSP_SUCCESS	Call successful
SSP_ERR_CELLULAR_CONFIG_FAILED	Configuration failed
SSP_ERR_CELLULAR_INIT_FAILED	Initialization failed
SSP_ERR_CELLULAR_TRANSMIT_FAILED	Transmit failed
SSP_ERR_CELLULAR_FW_UPTODATE	Up to date
SSP_ERR_CELLULAR_FW_UPGRADE_FAILED	Upgrade failed
SSP_ERR_CELLULAR_FAILED	General failure
SSP_ERR_CELLULAR_INVALID_STATE	Invalid state
SSP_ERR_CELLULAR_REGISTRATION_FAILED	Registration Failure

9.2 Cellular Framework Module Operational Overview

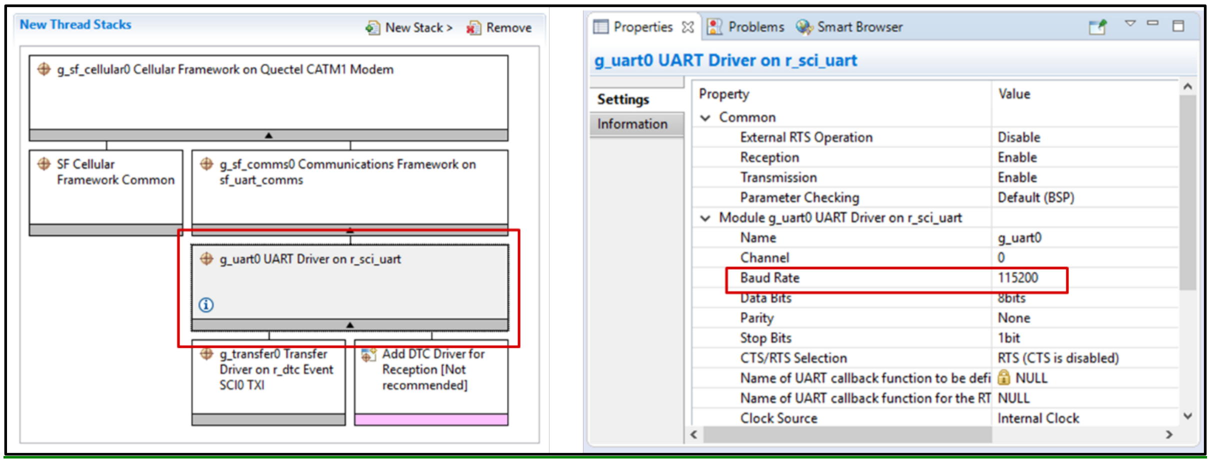
Refer to: 5.1.12.2 Cellular Framework Module APIs Overview, Cellular Framework Error Codes

Cellular Module baud rate

The Cellular Framework baud rate update feature works in the following stages:

- Check if the modem is operating at the baud rate set by the user in ISDE. If so, proceed with modem initialization.
- If the modem is not responding over the user specified baud rate, auto detect the baud rate at which the modem is currently operating.
- Switch the modem to the baud rate configured by the user in the ISDE. (This baud rate is then saved on the module).

The developer can configure the Baud rate of the UART under SF_CELLULAR in ISDE configuration as shown in the below image. This is the baud rate at which user wants the Modem to operate.


New Thread Stacks
New Stack >
Remove
g_sf_cellular0 Cellular Framework on Quectel CATM1 Modem
SF Cellular Framework Common
g_sf_comms0 Communications Framework on sf_uart_comms
g_uart0 UART Driver on r_sci_uart
g_transfer0 Transfer Driver on r_dtc Event SCI0 TXI
Add DTC Driver for Reception [Not recommended]
Properties
Problems
Smart Browser
Settings
Information

g_uart0 UART Driver on r_sci_uart

Property	Value
Common	
External RTS Operation	Disable
Reception	Enable
Transmission	Enable
Parameter Checking	Default (BSP)
Module g_uart0 UART Driver on r_sci_uart	
Name	g_uart0
Channel	0
Baud Rate	115200
Data Bits	8bits
Parity	None
Stop Bits	1bit
CTS/RTS Selection	RTS (CTS is disabled)
Name of UART callback function to be defi	NULL
Name of UART callback function for the RT	NULL
Clock Source	Internal Clock

Figure 155: ISDE configuration for baud rate

Operational Steps

1. The Cellular Framework will try to communicate over the baud rate specified in the ISDE. If the module responds over that Baud rate, then the baud rate change feature will not be initiated.
2. If the Modem does not respond to the ISDE configured baud rate, then the Cellular Framework will detect the baud rate at which the Modem is operating currently. Once the baud rate is detected, the Cellular Framework will change the baud rate of the Modem to the user configured baud rate. The Framework will then save the baud rate configuration in the Modem.

Note: The Cellular Framework will detect the baud rate of the Modem from the following list of baud rates {115200, 9600, 921600, 4800, 14400, 19200, 38400, 57600, 230400, 460800}. The baud rate configured in the ISDE will be skipped from the above list.

3. Once the Baud rate of the Modem is configured, the Cellular Framework will proceed with Modem initialization.

Framework close sequence

Calling nx_ip_delete() function will not de-initialize the Cellular module. For successful de-initialization of module, application should call cellular framework's close() API explicitly from application thread. Typical de-initialization sequence by the application should be

- networkDisconnect()
- close()
- nx_ip_delete()
- nx_ppp_delete()

Cellular Framework Module Limitations

- The current framework supports the following Cellular modules:
  - NimbeLink CAT3 (NL-SW-LTE-TSVG) Verizon-US
  - NimbeLink CAT3 (NL-SW-LTE-TEUG) India and Europe
  - NimbeLink CAT1 (NL-SW-LTE-GELS3-B and NL-SW-LTE-GELS3-C)
  - Quectel BG96 (CAT M1, NB-IoT and GPRS)
    - Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

10. Communications Framework on sf_comms_telnet usage notes

10.1 Telnet Communications Framework Module Operational Overview

Refer to: 5.1.14.3 Telnet Communications Framework Module Operational Overview

Operations supported by the framework include initializing the module using the `open` API, and closing the module using the `close` API. A communications read is implemented by the `read` API and a communications write by the `write` API. The `read` and `write` lock the module only till called the API is in action.

During a read operation, when using the TX_WAIT_FOREVER timeout, if the Ethernet link goes down, the `read` API will continue to wait for the data from the read queue. Once the Ethernet link is back up, the read operation resumes and exits. To exit the read operation during a link down event, the user must explicitly abort the read operation by calling the `close` API in the link status change callback function.

The `lock` API locks the module till the `unlock` API is called on the same module instance. This helps insure processing is completed before moving to the next API function call.

10.2 Configuration Settings for the NetX/NetX Duo Packet Pool Instance

Refer to: 5.1.14.5 Configuring the Telnet Communications Framework Module, Configuration Settings for the NetX/NetX Duo Packet Pool Instance

ISDE Property	Value	Description
Name	g_packet_pool0	Module name
Packet Size in Bytes	1568	Packet size selection
Number of Packets in Pool	16	Number of packets in pool selection
Name of generated initialization function	packet_pool_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Renesas Synergy^{TM} SSP

User's Manual v.1.5.2 Addendum

Publication Date: Rev.01.11 Nov 2018

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics Corporation
TOYOSU FORESIA, 3-2-24 Toyosu, Koto-ku, Tokyo 135-0061, Japan

Renesas Electronics America Inc.
1001 Murphy Ranch Road, Milpitas, CA 95035, U.S.A.
Tel: +1-408-432-8888, Fax: +1-408-434-5351

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-651-700

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709 Quantum Plaza, No.27 ZhichunLu, Haidian District, Beijing, 100191 P. R. China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, 200333 P. R. China
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HAL 2nd Stage, Indiranagar, Bangalore 560 038, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
17F, KAMCO Yangjae Tower, 262, Gangnam-daero, Gangnam-gu, Seoul, 06265 Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5338

Renesas Synergy™ SSP v1.5.2
User's Manual

Renesas Electronics Corporation