Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.

2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement of any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.

3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.

4. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.

5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The intended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user's manual or other Renesas Electronics document.

6. When using Renesas Electronics products, refer to the latest product information (data sheets, user's manuals, application notes, "General Notes for Handling and Using Semiconductor Devices" in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.

7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.

8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that require the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.

9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.

10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.

11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.

12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.
## Contents

1. NetX and NetX Duo Source Module Overview ................................................................. 1
2. Messaging Framework on sf_message Module Overview ................................................... 1
3. I2C Framework on sf_i2c Module Overview ................................................................... 1
   3.1 I2C Framework Module APIs Overview ....................................................................... 1
   3.2 Status Return Values................................................................................................... 2
   3.3 I2C Framework Module Operational Overview ............................................................ 2
   3.4 Bus Locking ............................................................................................................. 2
   3.5 I2C Framework Module Important Operational Notes and Limitations ...................... 3
   3.6 Pin Configuration Settings for the I2C Framework Module ......................................... 3
   3.7 Pin Configuration Settings for the I2C Framework Module ......................................... 4
4. UART Driver on r_sci_uart Module Overview ................................................................. 5
5. USBX Synergy Port Framework Module Overview ........................................................... 6
   5.1 USBX Synergy Port Framework Module Block Diagram .............................................. 6
   5.2 USBX Synergy Port Framework Module Operational Notes .......................................... 6
   5.3 USBX Synergy Port Framework Module Limitations .................................................... 6
6. JPEG_Encode module usage notes ................................................................................. 7
   6.1 JPEG Encode HAL Module Features ........................................................................... 7
   6.2 JPEG Encode HAL Module APIs Overview .................................................................. 7
   6.3 JPEG Encode HAL Module Operational Overview ....................................................... 8
7. Wi-Fi Framework module usage notes ........................................................................... 8
   7.1 NetX/NetX-Duo Driver Function .................................................................................. 8
   7.2 Wi-Fi Framework Module API Use Notes ................................................................... 9
8. NetX Port module usage notes ...................................................................................... 9
9. Cellular Framework Module usage notes ...................................................................... 9
   9.1 Cellular Framework Error Codes ............................................................................... 9
   9.2 Cellular Framework Module Operational Overview .................................................... 10
10. Communications Framework on sf_comms_telnent usage notes ................................... 11
   10.1 Telnet Communications Framework Module Operational Overview ........................ 11
   10.2 Configuration Settings for the NetX/NetX Duo Packet Pool Instance ......................... 11
11. Cellular Framework .................................................................................................... 12
   11.1 Cellular Framework Module Features ....................................................................... 12
   11.2 Cellular Framework Error Codes ............................................................................ 12
   11.3 Cellular Framework Module Limitations ................................................................... 12
   11.4 Using the Cellular Framework Module in an Application ........................................ 13
12. SPI Framework .......................................................................................................... 14
   12.1 SPI Framework Module API Summary .................................................................... 14
12.2 Status Return Values............................................................................................................ 15
12.3 SPI Framework Module Operational Notes .............................................................................. 15
12.4 Implementation Steps for Two Slave Devices on a Single Shared Bus .............................................. 15
12.5 Implementation Steps for Two Slave Devices on Two Shared Buses ........................................... 16
12.6 Adding Another Shared Bus .................................................................................................. 16

13. SPI HAL Layer API Reference .............................................................................................17
13.1 rspi_spcmd_assert_ssl_t Values ............................................................................................ 17
13.2 rspi_spcmd_br_div_t Values .................................................................................................. 18

Revision History.........................................................................................................................20
This document lists the differences between the SSP v1.5.0 and SSP v1.5.3. Take the following changes into account when using SSP v1.5.3.

References are to the SSP v1.5.0 User’s Manual, rev 1.10, r11um0091eu0110-synergy-ssp-v150.pdf. 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 shown in green
- Changes made for SSP v1.5.3 are shown in purple

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

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Example API Call and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>open</td>
<td>g_sf_i2c_device.p_api-&gt;open(g_sf_i2c_device.p_ctrl, g_sf_i2c_device.p_cfg); Opens a designated I2C device on the bus.</td>
</tr>
<tr>
<td>close</td>
<td>g_sf_i2c_device.p_api-&gt;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.</td>
</tr>
<tr>
<td>read</td>
<td>g_sf_i2c_device.p_api-&gt;read (g_sf_i2c_device.p_ctrl, &amp;destination, no_of_bytes_to_read, restart, timeout); Receives data from I2C device.</td>
</tr>
<tr>
<td>write</td>
<td>g_sf_i2c_device.p_api-&gt;write (g_sf_i2c_device.p_ctrl, &amp;source, no_of_bytes_to_write , restart, timeout); Transmits data to I2C device.</td>
</tr>
<tr>
<td>lock</td>
<td>g_sf_i2c_device.p_api-&gt;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.</td>
</tr>
</tbody>
</table>
unlock  
\[
g_{\text{sf} \_\text{i2c}\_\text{device}.p\_api}\rightarrow\text{unlock (}}g_{\text{sf} \_\text{i2c}\_\text{device}.p\_ctrl)\text{);}
\]
Unlocks the bus from a particular device and makes it available for other devices.

reset  
\[
g_{\text{sf} \_\text{i2c}\_\text{device}.p\_api}\rightarrow\text{reset (}}g_{\text{sf} \_\text{i2c}\_\text{device}.p\_ctrl, \text{timeout})\text{);}
\]
Aborts any in-progress transfer and forces the I2C peripheral into ready state.

versionGet  
\[
g_{\text{sf} \_\text{i2c}\_\text{device}.p\_api}\rightarrow\text{versionGet(version);}
\]
Retrieves the version information using the version pointer.

lockWait  
\[
g_{\text{sf} \_\text{i2c}\_\text{device}.p\_api}\rightarrow\text{lockWait (}}g_{\text{sf} \_\text{i2c}\_\text{device}.p\_ctrl, \text{timeout})\text{;}
\]
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

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

### 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-Mutex to 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.
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:
  o I2C Framework device module
  o Configure the I2C shared bus module for the first device being configured, then use the same bus for the remaining devices.
  o I2C HAL module
  o 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

<table>
<thead>
<tr>
<th>Pin Configuration Property</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation Mode</td>
<td>Disabled, Asynchronous UART, Synchronous UART, Simple I2C, Simple SPI, SmartCard</td>
<td>Select Simple I2C as the Operation Mode for I2C on SCI</td>
</tr>
<tr>
<td></td>
<td>Default: Disabled</td>
<td></td>
</tr>
<tr>
<td>RXD1_SCL1_MISO1</td>
<td>None, P212, P708</td>
<td>SCL Pin</td>
</tr>
<tr>
<td></td>
<td>Default: None</td>
<td></td>
</tr>
<tr>
<td>TXD1_SDA1_MOSI1</td>
<td>None, P213, P709</td>
<td>SDA Pin</td>
</tr>
<tr>
<td></td>
<td>Default: None</td>
<td></td>
</tr>
</tbody>
</table>
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:
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

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

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.

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-subsamples (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

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

### 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 user's 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 NetXIP 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 exchange 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**


### 9. Cellular Framework Module usage notes

#### 9.1 Cellular Framework Module Error Codes

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

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSP_SUCCESS</td>
<td>Call successful</td>
</tr>
<tr>
<td>SSP_ERR_CELLULAR_CONFIG_FAILED</td>
<td>Configuration failed</td>
</tr>
<tr>
<td>SSP_ERR_CELLULAR_INIT_FAILED</td>
<td>Initialization failed</td>
</tr>
</tbody>
</table>
### 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.

![ISDE configuration for baud rate](image)

**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 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

<table>
<thead>
<tr>
<th>ISDE Property</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>g_packet_pool0</td>
<td>Module name</td>
</tr>
<tr>
<td>Packet Size in Bytes</td>
<td>1568</td>
<td>Packet size selection</td>
</tr>
<tr>
<td>Number of Packets in Pool</td>
<td>16</td>
<td>Number of packets in pool selection</td>
</tr>
<tr>
<td>Name of generated initialization function</td>
<td>packet_pool_init0</td>
<td>Name of generated initialization function selection</td>
</tr>
<tr>
<td>Auto Initialization</td>
<td>Enable, Disable</td>
<td>Auto initialization selection</td>
</tr>
<tr>
<td></td>
<td>Default: Enable</td>
<td></td>
</tr>
</tbody>
</table>
11. Cellular Framework

11.1 Cellular Framework Module Features
Refer to: 5.1.12.1 Cellular Framework Introduction, Cellular Framework Module Features

- Supports connectivity using:
  - BSD Socket interface for On-Chip stack present on the Cellular Module
  - NetX Stack on Synergy MCU (Host) using NSAL interface
- Supports a common set of APIs to interface to the networking stack and a generic interface for the different Cellular hardware modules.
- Using generic APIs and abstraction, applications developed for the cellular hardware module can be easily migrated to work with another cellular hardware module.
- Supported Cellular modems:
  - NimbeLink CAT3 (NL-SW-LTE-TSVG) Verizon-US
  - NimbeLink CAT3 (NL-SW-LTE-TEUG) India and Europe
  - NimbeLink CAT1 (NL-SW-LTE-GELS3-D) Verizon-US
  - Quectel BG96 (CAT M1, NB-IoT and GPRS) Rev F

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

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSP_SUCCESS</td>
<td>Call successful</td>
</tr>
<tr>
<td>SSP_ERR_CELLULAR_CONFIG_FAILED</td>
<td>Configuration failed</td>
</tr>
<tr>
<td>SSP_ERR_CELLULAR_INIT_FAILED</td>
<td>Initialization failed</td>
</tr>
<tr>
<td>SSP_ERR_CELLULAR_TRANSMIT_FAILED</td>
<td>Transmit failed</td>
</tr>
<tr>
<td>SSP_ERR_CELLULAR_FW_UPTODATE</td>
<td>Up to date</td>
</tr>
<tr>
<td>SSP_ERR_CELLULAR_FW_UPGRADE_FAILED</td>
<td>Upgrade failed</td>
</tr>
<tr>
<td>SSP_ERR_CELLULAR_FAILED</td>
<td>General failure</td>
</tr>
<tr>
<td>SSP_ERRgetCellular_INVALID_STATE</td>
<td>Invalid state</td>
</tr>
<tr>
<td>SSP_ERR_CELLULAR_REGISTRATION_FAILED</td>
<td>Registration Failure</td>
</tr>
</tbody>
</table>

11.3 Cellular Framework Module Limitations
Refer to: 5.1.12.3 Cellular Framework Module Operational Overview, 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-D with firmware version 4.3.3.0c and A-Revision 36343)
  - Quectel BG96 (CAT M1, NB-IoT and GPRS) with firmware version BG96MAR02A07M1G_01.008.01.008
    The firmware for BG96 and instructions for updating it on the modules can be downloaded from the Partner Projects page of the Solutions Gallery:
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

11.4 Updating the Quectel BG96 firmware
Refer to: 5.1.12.4 Including the Cellular Framework Module in an Application

**Updating the Quectel BG96 firmware**

To determine the current version of firmware on the module, issue the following commands:

```
ATI
AT+QGMR
```

To update (flash) the firmware for the Quectel BG96 (CAT M1, NB-IoT and GPRS), follow these instructions:
1. Visit the Quectel BG96 product download page: https://www.quectel.com/ProductDownload/BG96.html. Click the Download button and open the zip file.

2. From the Tool folder, extract the QFLASH tool.

3. From the Driver folder, extract the Quectel LTE Windows USB Driver.

4. Please follow the steps below to install the LTE Driver:
   a. Unzip and install the LTE Driver on your PC, then connect the USB port of the module to the PC with a USB cable. Check if the DM port of USB appears in device manager:

5. Download the firmware from the following URL: https://cnquectel.sharepoint.com/:u:/g/Home/EX9QakMg64xAsy-Tp-\vb5SMBoAhL3JGb9wByYw3HiBQMg?e=lweReP

6. Unzip firmware and QFLASH file and follow these steps:
   a. Click Load FW Files and load the firmware path (shown as 1 in the figure below):

![Load FW Files](image)

   b. Choose DM port (shown as 2 in the figure above).
   c. Choose 460800 baud rate (shown as 3 in the figure above).
   d. Click Start.
   e. Normally, it takes a while for the upgrade to complete and display the success message. Check module firmware with “ATI” command.

5.1.12.4 Including the Cellular Framework Module in an Application

11.5 Using the Cellular Framework Module in an Application

Refer to: 5.1.12.6 Using the Cellular Framework Module in an Application
The user added code is responsible for the data connections and the ICMP ping. It is responsible for sending the ping request to the user entered Public IP address and verifying the ping response. Callback functions can be implemented for PPP link down, PPP link up and cellular provisioning.

12. SPI Framework

12.1 SPI Framework Module API Summary

Refer to: 5.1.29.2 SPI Framework Module APIs Overview, SPI Framework Module API Summary

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Example API Call and Description</th>
</tr>
</thead>
</table>
| open          | g_sf_spi_device.p_api->open(g_sf_spi_device.p_ctrl, g_sf_spi_device.p_cfg);  
Open a designated SPI device on a bus. |
| read          | g_sf_spi_device.p_api->read(g_sf_spi_device.p_ctrl, &destination, length, SPI_BIT_WIDTH_8_BITS, timeout);  
Receive data from SPI device. |
| write         | g_sf_spi_device.p_api->write(g_sf_spi_device.p_ctrl, &source, length, SPI_BIT_WIDTH_8_BITS, timeout);  
Transmit data to SPI device. |
| writeRead     | g_sf_spi_device.p_api->writeRead(g_sf_spi_device.p_ctrl, &source, &destination, length, SPI_BIT_WIDTH_8_BITS, timeout);  
Simultaneously transmits data to an SPI device while receiving data from an SPI device (full duplex). The writeread API gets a mutex object, handles the SPI data transmission at SPI HAL layer, and receives data from the SPI HAL layer. The API uses the event flag wait to synchronize to completion of data transfer. |
| close         | g_sf_spi_device.p_api->close(g_sf_spi_device.p_ctrl);  
Disable the SPI device designated by the control handle and close the RTOS services used by the bus, if no devices are connected to the bus. This function removes power to the SPI channel designated by the handle and disables the associated interrupts. |
| lock          | g_sf_spi_device.p_api->lock(g_sf_spi_device.p_ctrl);  
Lock the bus for a device. The locking allows devices to reserve a bus to themselves for a given period of time (such as between lock and unlock). This allows devices to complete several reads and writes on the bus without an interrupt. |
| unlock        | g_sf_spi_device.p_api->unlock(g_sf_spi_device.p_ctrl);  
Unlock the bus for a particular device and make the bus usable for other devices. |
| versionGet    | g_sf_spi_device.p_api->versionGet(&version);  
Retrieve the API version with the version pointer. |
| lockWait      | g_sf_spi_device.p_api->lockWait(g_sf_spi_device.p_ctrl, timeout);  
Lock the bus for a device. The locking allows devices to reserve a bus to themselves for a given period of time (that is, between lock and unlock). This allows devices to complete several reads and writes on the bus without interrupt. The wait option allows thread to wait for the specified timeout when acquiring the bus mutex. |
12.2 Status Return Values
Refer to: 5.1.29.2 SPI Framework Module APIs Overview, Status Return Values

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSP_SUCCESS</td>
<td>Function completed successfully</td>
</tr>
<tr>
<td>SSP_ERR_INVALID_MODE</td>
<td>Invalid mode</td>
</tr>
<tr>
<td>SSP_ERR_INVALID_CHANNEL</td>
<td>Invalid channel</td>
</tr>
<tr>
<td>SSP_ERR_IN_USE</td>
<td>In-use error</td>
</tr>
<tr>
<td>SSP_ERR_INVALID_ARGUMENT</td>
<td>Invalid argument</td>
</tr>
<tr>
<td>SSP_ERR_QUEUE_UNAVAILABLE</td>
<td>Queue unavailable</td>
</tr>
<tr>
<td>SSP_ERR_INVALID_POINTER</td>
<td>Invalid pointer</td>
</tr>
<tr>
<td>SSP_ERR_INTERNAL</td>
<td>Internal error</td>
</tr>
<tr>
<td>SSP_ERR_TRANSFER_ABORTED</td>
<td>Transfer aborted</td>
</tr>
<tr>
<td>SSP_ERR_MODE_FAULT</td>
<td>Mode fault</td>
</tr>
<tr>
<td>SSP_ERR_READ_OVF</td>
<td>Read overflow</td>
</tr>
<tr>
<td>SSP_ERR_PARITY</td>
<td>Parity error</td>
</tr>
<tr>
<td>SSP_ERR_OVERRUN</td>
<td>Overrun error</td>
</tr>
<tr>
<td>SSP_ERR_UNDEF</td>
<td>Unknown error</td>
</tr>
<tr>
<td>SSP_ERR_TIMEOUT</td>
<td>Timeout error</td>
</tr>
<tr>
<td>SSP_ERR_NOT_OPEN</td>
<td>Device not opened</td>
</tr>
<tr>
<td>SSP_ERR_ALREADY_OPEN</td>
<td>Requested channel is already open in a different configuration</td>
</tr>
</tbody>
</table>

12.3 SPI Framework Module Operational Notes
Refer to: 5.1.29.3 SPI Framework Module Operational Overview, SPI Framework Module Operational Notes

- Multiple SPI devices can be configured to share a common bus. Once the SPI Framework bus module is configured, different SPI peripherals (devices) can be connected to that bus.
- For each SPI device connected to the bus, one SPI HAL module (new or shared) and one SPI Framework device module must be added.
- User defined Callback is not required as it has been internally taken care by framework.
- Setting the interrupts to different priority levels could result in improper operation.
- In the SPI 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 SPI channel.
- Lock functionality will be effective for devices from different threads. If multiple devices connected to the bus are from the same thread, the SPI 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 used from multiple threads, and the device locks the SPI bus from one thread, the same device cannot access the SPI bus from other threads.

12.4 Implementation Steps for Two Slave Devices on a Single Shared Bus
Refer to: 5.1.29.6 Using the SPI Framework Module in an Application, Implementation Steps for Two Slave Devices on a Single Shared Bus

Implementation Steps for Two Slave Devices on a Single Shared Bus

When using the SPI framework module to create a single bus with multiple slave devices create two thread stacks each with an SPI framework instance. These instances will use the same shared bus instance. Follow the steps below to see how this is done within the SSP Configurator.

Note: The following steps assume some familiarity with the use of the SSP development environment. If any of the following steps are confusing, read over the first few chapters of the SSP User's Manual to become familiar with the SSP development environment.
1) Add the first SPI framework device module to a new or existing thread. This creates the SPI master stack. A shared bus on sf_spi is added along with the SPI driver. The SPI driver can be selected for implementation on r_rspior r_sci_spi. The DTC transfer driver is also added by default. This can be removed if the CPU transfer mode is needed instead.

12.5 Implementation Steps for Two Slave Devices on Two Shared Buses
Refer to: 5.1.29.6 Using the SPI Framework Module in an Application, Implementation Steps for Two Slave Devices on Two Shared Buses

When using the SPI framework module to create a single bus with multiple slave devices create two thread stacks each with an SPI framework instance. These instances will use the same shared bus instance. Follow the steps below to see how this is done within the SSP Configurator.

12.6 Adding Another Shared Bus
Refer to: 5.1.29.6 Using the SPI Framework Module in an Application, Adding Another Shared Bus

The typical steps in using the SPI Framework module in an application are:

1. Initialize the SPI Framework device module using the sf_spi_api_t::open API function. Each SPI framework device module needs to call the spi_api_t::open API function at least once before performing any operations on the bus.

2. Lock the bus for continuous transfer using the sf_spi_api_t::lock or sf_spi_api_t::lockWait API function for a particular SPI Framework device module. Once Bus is locked by a particular SPI Framework device module it cannot be used by any other SPI Framework device module on the bus. This ensures that ownership of the bus remains with the locked module until it explicitly unlocks it. Any kind of operation from other SPI Framework device modules on the bus will return a fail status during this period. It is not mandatory to lock the bus before any read/write operations on the bus. It is optional.

3. Read data using the sf_spi_api_t::read API function. The read operation will not be successful if the bus is already locked by any other SPI Framework device module.

4. Write data using the sf_spi_api_t::write API function. The write operation will not be successful if the bus is already locked by any other SPI Framework device module.

5. Write and read data simultaneously using the sf_spi_api_t::writeRead API function. The simultaneous read and write operation will not be successful if the bus is already locked by any other SPI Framework device module.

6. Unlock the bus from continuous transfer using the sf_spi_api_t::unlock API function if it is already locked by the same device. Once the bus is unlocked other SPI Framework device modules can use it. It is necessary to unlock the locked bus after the intended read/write operation is completed.

7. Close the SPI Framework device module using the sf_spi_api_t::close API function. Each SPI Framework device module can call the sf_spi_api_t::close API function after all read/write operations on the bus are over.

These common steps are illustrated in a typical operational flow in the following figure:
13. SPI HAL Layer API Reference

13.1 rspi_spcmd_assert_ssl_t Values

Refer to: 6.1.5.46 SPI, rspi_spcmd_assert_ssl_t

- rspi_spcmd_assert_ssl_t

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSPI_SPCMD_ASSERT_SSL0</td>
<td>Select SSL0</td>
</tr>
<tr>
<td>RSPI_SPCMD_ASSERT_SSL1</td>
<td>Select SSL1</td>
</tr>
<tr>
<td>RSPI_SPCMD_ASSERT_SSL2</td>
<td>Select SSL2</td>
</tr>
<tr>
<td>RSPI_SPCMD_ASSERT_SSL3</td>
<td>Select SSL3</td>
</tr>
</tbody>
</table>

Slave select to be asserted during transfer operation.
13.2 rspi_spcmd_br_div_t Values

Refer to: 6.1.5.46 SPI, rspi_ssl_level_keep_t

◆ rspi_spcmd_br_div_t

enum rspi_spcmd_br_div_t

Clock base rate division

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSPI_SPCMD_BR_DIV_1</td>
<td>Select the base bit rate</td>
</tr>
<tr>
<td>RSPI_SPCMD_BR_DIV_2</td>
<td>Select the base bit rate divided by 2</td>
</tr>
<tr>
<td>RSPI_SPCMD_BR_DIV_4</td>
<td>Select the base bit rate divided by 4</td>
</tr>
<tr>
<td>RSPI_SPCMD_BR_DIV_8</td>
<td>Select the base bit rate divided by 8</td>
</tr>
</tbody>
</table>

◆ rspi_ssl_level_keep_t
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
- renesassynergy.com/software
- renesassynergy.com/ssp
- renesassynergy.com/addons
- renesassynergy.com/softwareglossary
- renesassynergy.com/tools

Synergy Hardware
- renesassynergy.com/hardware
- renesassynergy.com/mcus
- renesassynergy.com/mcuglossary
- renesassynergy.com/parametric
- renesassynergy.com/kits

Synergy Solutions Gallery
- renesassynergy.com/solutionsgallery

Partner projects
- renesassynergy.com/partnerprojects

Application projects
- renesassynergy.com/applicationprojects

Self-service support resources:
- renesassynergy.com/docs
- renesassynergy.com/knowledgebase
- renesassynergy.com/forum
- renesassynergy.com/training
- renesassynergy.com/videos
- renesassynergy.com/support
## Revision History

<table>
<thead>
<tr>
<th>Rev.</th>
<th>Date</th>
<th>Description</th>
<th>Page</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>Oct 10, 2018</td>
<td>Addendum to SSP User's Manual for SSP v1.5.1</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>1.11</td>
<td>Nov 15, 2018</td>
<td>Addendum to SSP User's Manual for SSP v1.5.2</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>1.25</td>
<td>Dec 12, 2018</td>
<td>Addendum to SSP User's Manual for SSP v1.5.3</td>
<td>—</td>
<td></td>
</tr>
</tbody>
</table>

All trademarks and registered trademarks are the property of their respective owners.