ATWINC15X0
Wi-Fi Add-on Component
for RA Platform

User's Manual

Outline

Revisions

REVISION	DATE	DESCRIPTION	STATUS	AUTHOR	REVISER
0.1	11/11/2019	Document created.	release	C. Tagliaferri	
0.2	13/11/2019	Added FreeRTOS+TCP example.	release	C. Tagliaferri	
0.3	15/11/2019	Added AWS WiFi Library API support.	release	C. Tagliaferri	
1.0	06/04/2020	Projects and documentation updated to FSP 1.0.0	release	C. Tagliaferri	
1.1	09/12/2021	Projects and documentation updated to FSP 3.5.0	release	C. Tagliaferri	

Disclaimer

RELOC s.r.l.
Strada Langhirano, 264/3A
43124 – Parma (Italy)

info@reloc.it – www.reloc.it
fb www.facebook.com/relocsrl
Land +39-0521-649116

References

[1] Renesas Electronics, "Renesas Flexible Software Package (FSP) 3.5.0 – User's Manual", Rev. 2.50, Dec 3, 2021. Document Number: R11UM0155EU0250.

[2] RELOC s.r.l., RA FSP Wi-Fi Driver webpage https://www.reloc.it/products/pmod-wi-fi-atwinc1500/.

[3] Renesas Electronics, Renesas RA Family webpage https://www.renesas.com/eu/en/products/microcontrollers-microprocessors/ra-cortex-m-mcus.

[4] Amazon FreeRTOS WiFi Library https://docs.aws.amazon.com/freertos/latest/userguide/freertos-wifi.html

1. Overview

The ATWINC15X0 Wi-Fi Add-on can be used in two different modes: with the FreeRTOS+TCP stack running on the MCU or using the on-chip stack.

With the FreeRTOS+TCP stack the WINC15X0 module enable a pass-through mode and all the network stack is handle by the FreeRTOS+TCP middleware.

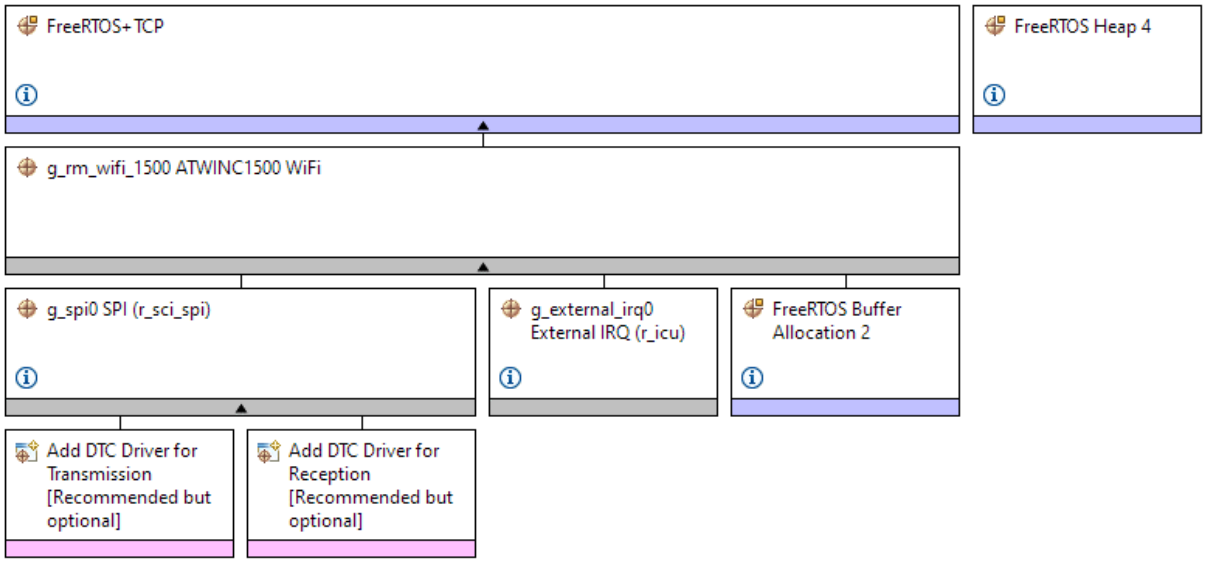

Figure 1: ATWINC15X0 Add-on, FreeRTOS+TCP stack view.

Oppositely, the on-chip stack version relies on the TCP stack running on the WINC15X0 module and provides a socket API layer to the application.

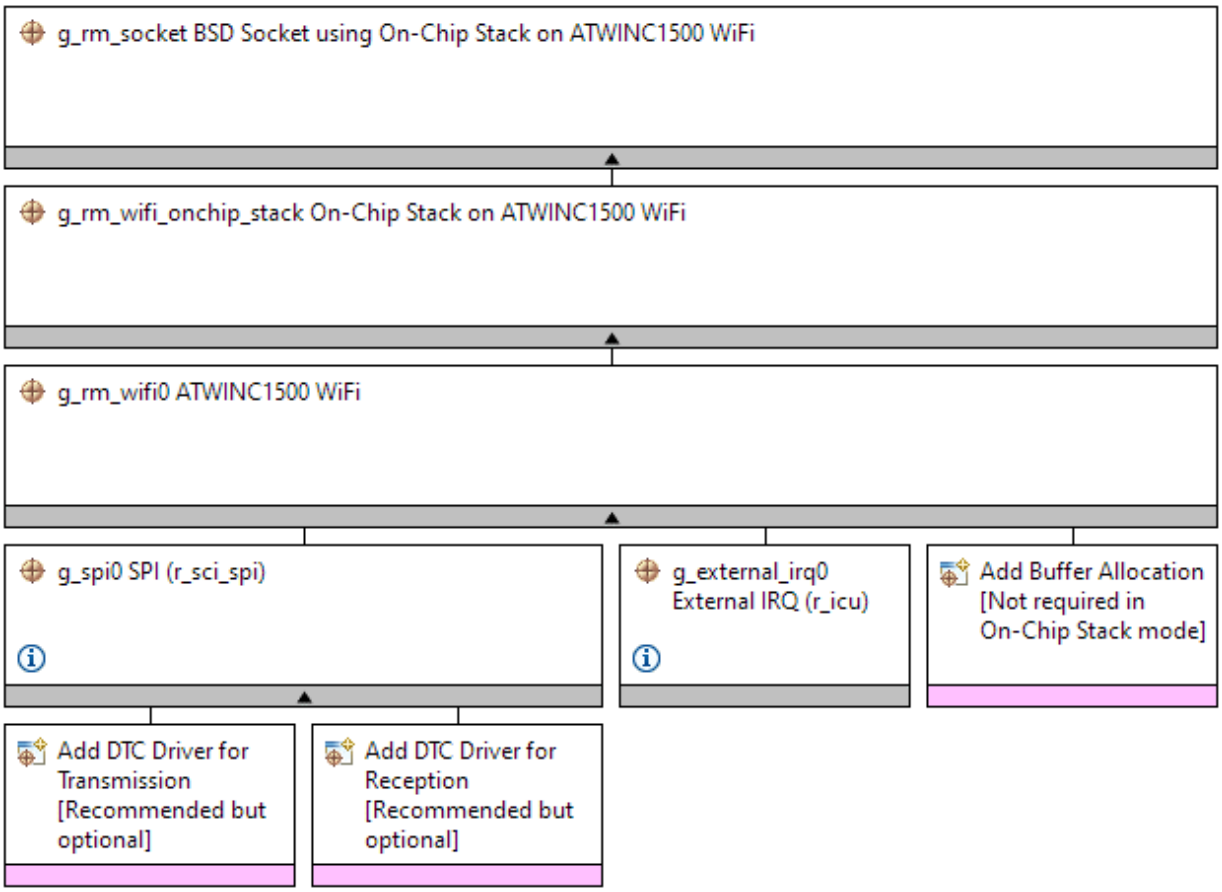

Figure 2: ATWINC15x0 Add-on, on-chip stack view.

1.1 Supported FSPs

ATWINC15X0 Wi-Fi Add-on Component for RA Platform currently supports the FSP v3.5.0 release.

2. API References: Wi-Fi Middleware on ATWINC1500

2.1 Amazon FreeRTOS WiFi Library

2.1.1 Summary

The Amazon FreeRTOS WiFi Library [4] has been ported on top of the ATWINC1500 driver; the user is free to choose between this API level or the lower level ATWINC1500 driver interface. The WiFi Library relies on the ATWINC1500 Wi-Fi Driver API so no differences should be observed.

2.1.2 Interface APIs

WiFi Library API	FreeRTOS+TCP stack mode	On-Chip stack mode
WIFI_On	supported	supported
WIFI_Off	supported	supported
WIFI_ConnectAP	supported	supported
WIFI_Disconnect	supported	supported
WIFI_Reset	supported	supported
WIFI_SetMode	not supported	not supported
WIFI_GetMode	supported	supported
WIFI_NetworkAdd	not supported	not supported
WIFI_NetworkGet	not supported	not supported
WIFI_NetworkDelete	not supported	not supported
WIFI_Ping	not supported	not supported
WIFI_GetIP	supported	supported
WIFI_GetMAC	supported	supported
WIFI_GetHostIP	supported	not supported
WIFI_Scan	supported	supported
WIFI_StartAP	supported	supported
WIFI_StopAP	supported	supported
WIFI_ConfigureAP	supported	supported
WIFI_SetPMMode	not supported	not supported
WIFI_GetPMMode	not supported	not supported
WIFI_IsConnected	supported	supported
WIFI_RegisterNetworkStateChangeEventCallback	supported	supported

2.2 Wi-Fi Driver

2.2.1 Summary

Below are the APIs exposed by the ATWINC1500 Wi-Fi Driver.

2.2.2 Data Structures

- `st_rm_wifi_winc1500_cfg` with data fields:
  - `spi_instance_t const * p_r_spi`: Driver SPI Interface used for Wi-Fi communications;
  - `external_irq_instance_t const * p_irq`: IRQ Interface used for Wi-Fi communications;
  - `bsp_io_port_pin_t pin_reset`: Pin used for resetting module;
  - `bsp_io_port_pin_t pin_enable`: Port pin used as chip enable;
  - `uint8_t internal_thread_priority`: Internal thread priority;
  - `uint32_t dhcp_address`: DHCP address IPv4;
  - `uint32_t bus_access_timeout_ms`: SPI Bus access timeout in milliseconds;
  - `uint32_t api_access_timeout_ms`: API access timeout (no simultaneous use of a single API) in milliseconds;
  - `uint32_t wait_result_timeout_ms`: Wait for result timeout in milliseconds;
  - `tpfAppSocketCb socket_callback`: Callback for socket APIs;
  - `tpfAppSocketCb dns_callback`: Callback for DNS APIs.

2.2.2.1 Struct Reference

- `#include <rm_wifi_winc1500.h>`

2.2.3 Defines

- `#define RM_WIFI_WINC1500_CODE_VERSION_MAJOR (0)`
  Major version of code that implements the API defined in WiFi Driver.
- `#define RM_WIFI_WINC1500_CODE_VERSION_MINOR (1)`
  Minor version of code that implements the API defined in WiFi Driver.

2.2.4 Interface APIs

- `fsp_err_t RM_WIFI_WINC1500_Open(rm_wifi_ctrl_t * p_ctrl, rm_wifi_cfg_t const * const p_cfg)`
  Initialize Wi-Fi module.
- `fsp_err_t RM_WIFI_WINC1500_Close(rm_wifi_ctrl_t * const p_ctrl)`
  Stop Wi-Fi module.
- `fsp_err_t RM_WIFI_WINC1500_MulticastListAdd(rm_wifi_ctrl_t * const p_ctrl, uint8_t const * const p_addr)`
  Add MAC address in multicast list.
- `fsp_err_t RM_WIFI_WINC1500_MulticastListDelete(rm_wifi_ctrl_t * const p_ctrl, uint8_t const * const p_addr)`
  Delete MAC address from multicast list.
- `fsp_err_t RM_WIFI_WINC1500_StatisticsGet(rm_wifi_ctrl_t * const p_ctrl, rm_wifi_stats_t * const p_wifi_device_stats)`
  Get the interface statistics.
- `fsp_err_t RM_WIFI_WINC1500_Transmit(rm_wifi_ctrl_t * const p_ctrl, uint8_t * const p_buf, uint32_t length)`
  Transmit data packets.
- `fsp_err_t RM_WIFI_WINC1500_ProvisioningSet(rm_wifi_ctrl_t * const p_ctrl, rm_wifi_provisioning_t const * const p_wifi_provisioning)`
  Provisions the Wi-Fi module.
- `fsp_err_t RM_WIFI_WINC1500_ProvisioningGet(rm_wifi_ctrl_t * const p_ctrl, rm_wifi_provisioning_t * const p_wifi_provisioning)`
  Reads the current Wi-Fi Provisioning information of the Wi-Fi module.
- `fsp_err_t RM_WIFI_WINC1500_InfoGet(rm_wifi_ctrl_t * const p_ctrl, rm_wifi_info_t * const p_wifi_info)`
  Get Wi-Fi module information.
- `fsp_err_t RM_WIFI_WINC1500_Scan(rm_wifi_ctrl_t * const p_ctrl, rm_wifi_scan_t * const p_scan, uint8_t * const p_cnt)`
  Scans for available APs.
- `fsp_err_t RM_WIFI_WINC1500_AccessControlListAdd(rm_wifi_ctrl_t * const p_ctrl, uint8_t const * const p_mac)`
  Add MAC address from Access control list.
- `fsp_err_t RM_WIFI_WINC1500_AccessControlListDelete(rm_wifi_ctrl_t * const p_ctrl, uint8_t const * const p_mac)`
  Delete MAC address from Access control list.
- `fsp_err_t RM_WIFI_WINC1500_MACAddressGet(rm_wifi_ctrl_t * const p_ctrl, uint8_t * const p_mac)`
  Get MAC address of Wi-Fi module.
- `fsp_err_t RM_WIFI_WINC1500_MACAddressSet(rm_wifi_ctrl_t * const p_ctrl, uint8_t const * const p_mac)`
  Set MAC address of Wi-Fi module.
- `fsp_err_t RM_WIFI_WINC1500_VersionGet(fsp_version_t * const p_version)`
  Set driver version based on compile time macros.

2.2.4.1 APIs Documentation

- **fsp_err_t RM_WIFI_WINC1500_Open(rm_wifi_ctrl_t * p_ctrl, rm_wifi_cfg_t const * const p_cfg)**
  Initializes Wi-Fi module.

Implements rm_wifi_api_t::open.
This function performs the following tasks:
- Initializes WINC1500 Wi-Fi Driver and Configure the parameters as per the p_cfg
- Update global variables for future use.

Return values:

FSP_SUCCESS	Driver initialization done successfully.
FSP_ERR_ALREADY_OPEN	Wi-Fi Driver is already opened.
FSP_ERR_ASSERTION	Argument NULL is passed or ThreadX resources initialization failed.
FSP_ERR_INVALID_HW_CONDITION	Module initialization failed. Verify pin configurations.
FSP_ERR_UNSUPPORTED	WINC1500 Module firmware version unsupported.
FSP_ERR_OUT_OF_MEMORY	Not enough available space in the heap for the internal thread.
FSP_ERR_WIFI_CONFIG_FAILED	Configuration of Wi-Fi driver failed.

- **fsp_err_t RM_WIFI_WINC1500_Close(rm_wifi_ctrl_t * const p_ctrl)**
  Stops Wi-Fi module.
  Implements rm_wifi_api_t::close.
  This function performs the following tasks:
  - Disable the Interrupt and suspend the WINC1500 driver task thread.

  Return values:

FSP_SUCCESS	Driver closed successfully.
FSP_ERR_ASSERTION	Argument NULL is passed.
FSP_ERR_NOT_OPEN	Driver already closed.
FSP_ERR_WIFI_FAILED	Timeout or error while waiting for API result.
FSP_ERR_TIMEOUT	Timeout while waiting for SPI bus access.
FSP_ERR_ABORTED	Error while waiting for SPI bus access.
FSP_ERR_INVALID_HW_CONDITION	Error closing Wi-Fi driver.
FSP_ERR_INTERNAL	Error deleting threadX objects.

- **fsp_err_t RM_WIFI_WINC1500_MulticastListAdd(rm_wifi_ctrl_t * const p_ctrl, uint8_t const * const p_addr)**
  Add MAC address in multicast list.
  Implements rm_wifi_api_t::multicastListAdd
  This function performs the following tasks:
  - Adds specified MAC address in Multicast list.
  IMPORTANT NOTE: this API is not supported in "On-Chip Stack" mode.

  Return value:

FSP_SUCCESS	Successfully added the mac address to the multicast list.
FSP_ERR_NOT_OPEN	Driver not opened.
FSP_ERR_WIFI_CONFIG_FAILED	Lower level error while adding the mac address to the multicast list.
FSP_ERR_TIMEOUT	Timeout while waiting for SPI bus access.
FSP_ERR_ABORTED	Error while waiting for SPI bus access.
FSP_ERR_UNSUPPORTED	API not supported ("On-Chip Stack" mode only)

- **fsp_err_t RM_WIFI_WINC1500_MulticastListDelete(rm_wifi_ctrl_t * const p_ctrl, uint8_t const * const p_addr)**
  Delete MAC address from multicast list.
  Implements rm_wifi_api_t::multicastListDelete
  This function performs the following tasks:
  - Deletes specified MAC address in Multicast list.
  IMPORTANT NOTE: this API is not supported in "On-Chip Stack" mode.

  Return value:

FSP_SUCCESS	Successfully removed the mac address from the multicast list.
FSP_ERR_NOT_OPEN	Driver not opened.
FSP_ERR_WIFI_CONFIG_FAILED	Lower level error while removing the mac address from the multicast list.
FSP_ERR_TIMEOUT	Timeout while waiting for SPI bus access.
FSP_ERR_ABORTED	Error while waiting for SPI bus access.
FSP_ERR_UNSUPPORTED	API not supported ("On-Chip Stack" mode only)

- **fsp_err_t RM_WIFI_WINC1500_StatisticsGet(rm_wifi_ctrl_t * const p_ctrl, rm_wifi_stats_t * const p_wifi_device_stats)**
  Get the interface statistics.
  Implements rm_wifi_api_t::statisticsGet
  This function performs the following tasks:
  - Collect the statistics information of Wi-Fi interface.

  Return values:

FSP_SUCCESS	Successfully get the Statistics information.
FSP_ERR_ASSERTION	Argument NULL is passed.
FSP_ERR_NOT_OPEN	Driver not open.

- **fsp_err_t RM_WIFI_WINC1500_Transmit(rm_wifi_ctrl_t * const p_ctrl, uint8_t * const p_buf, uint32_t length)**

Transmit data packets.
Implements rm_wifi_api_t::transmit
This function performs the following tasks:
- Adds packets in the transmit queue

Return values:

FSP_SUCCESS	Successfully added the packet in driver transmit queue.
FSP_ERR_ASSERTION	Argument NULL is passed.
FSP_ERR_WIFI_TRANSMIT_FAILED	Error while sending packets.
FSP_ERR_NOT_OPEN	Driver not open.
FSP_ERR_TIMEOUT	Timeout while waiting for SPI bus access.
FSP_ERR_IN_USE	Error while waiting for SPI bus access.

- **fsp_err_t RM_WIFI_WINC1500_ProvisioningSet(rm_wifi_ctrl_t * const p_ctrl, rm_wifi_provisioning_t const * const p_wifi_provisioning)**
Provisions the Wi-Fi module.
Implements rm_wifi_api_t::provisioningSet
This function performs the following tasks:
- Provisions the Wi-Fi driver;
- Start Wi-Fi interface in AP or STATION mode as provisioned.

Return values:

FSP_SUCCESS	Successfully provisioned the driver.
FSP_ERR_NOT_OPEN	Driver not open.
FSP_ERR_ASSERTION	Argument NULL is passed.
FSP_ERR_WIFI_FAILED	Error performing provision settings.
FSP_ERR_WIFI_INVALID_MODE	Invalid provisioning mode.
FSP_ERR_INVALID_ARGUMENT	Invalid parameter/argument is passed.
FSP_ERR_INVALID_HW_CONDITION	Error while sending provisioning request.
FSP_ERR_TIMEOUT	Timeout while waiting for SPI bus access.
FSP_ERR_ABORTED	Error while waiting for SPI bus access.

- **fsp_err_t RM_WIFI_WINC1500_ProvisioningGet(rm_wifi_ctrl_t * const p_ctrl, rm_wifi_provisioning_t * const p_wifi_provisioning)**
Reads the current Wi-Fi Provisioning information of the Wi-Fi module.
Implements rm_wifi_api_t::provisioningGet
This function performs the following tasks:
- Reads the provisioning information

Return values:

FSP_SUCCESS	Successfully reads provisioning information.

FSP_ERR_NOT_OPEN	Driver not open.
FSP_ERR_ASSERTION	Argument NULL has been passed.

- **fsp_err_t RM_WIFI_WINC1500_InfoGet(rm_wifi_ctrl_t * const p_ctrl, rm_wifi_info_t * const p_wifi_info)**
  Get Wi-Fi module information.
  Implements rm_wifi_api_t::infoGet
  This function performs the following tasks:
  - Get Wi-Fi module information like chipset/driver information, RSSI, noise level, link quality.

  Return value:

FSP_SUCCESS	Successfully read information.
FSP_ERR_NOT_OPEN	Driver not opened.
FSP_ERR_ASSERTION	Argument NULL is passed.
FSP_ERR_INVALID_HW_CONDITION	Error while sending information requests.
FSP_ERR_WIFI_FAILED	Timeout or error while waiting for API result.
FSP_ERR_IN_USE	Another thread is using this API and timeout for the access occurred.
FSP_ERR_TIMEOUT	Timeout while waiting for SPI bus access.
FSP_ERR_ABORTED	Error while waiting for SPI bus access.

- **fsp_err_t RM_WIFI_WINC1500_Scan(rm_wifi_ctrl_t * const p_ctrl, rm_wifi_scan_t * const p_scan, uint8_t * const p_cnt)**
  Scans for available APs.
  Implements rm_wifi_api_t::scan
  This function performs the following tasks:
  - Scans for available AP's SSID and return the list to caller.

  Return values:

FSP_SUCCESS	Successfully scan the network for available APs.
FSP_ERR_NOT_OPEN	Driver not opened.
FSP_ERR_ASSERTION	Argument NULL is passed.
FSP_ERR_WIFI_FAILED	Error in Scanning.
FSP_ERR_INVALID_MODE	Scan not supported in AP mode.
FSP_ERR_INVALID_HW_CONDITION	Error while sending information requests.
FSP_ERR_WIFI_FAILED	Timeout or error while waiting for API result.
FSP_ERR_IN_USE	Another thread is using this API and timeout for the access occurred.
FSP_ERR_TIMEOUT	Timeout while waiting for SPI bus access.

FSP_ERR_ABORTED	Error while waiting for SPI bus access.

- **fsp_err_t RM_WIFI_WINC1500_AccessControlListAdd(rm_wifi_ctrl_t * const p_ctrl, uint8_t const * const p_mac)**
  Add MAC address from Access control list.
  Implements rm_wifi_api_t::accessControlListAdd
  This function performs the following tasks:
  - Adds specified MAC address in access control list.

  Return value:

FSP_SUCCESS	Successfully added the mac address in access control list.
FSP_ERR_NOT_OPEN	Driver not opened.
FSP_ERR_ASSERTION	Argument NULL is passed.
FSP_ERR_UNSUPPORTED	Operation is not supported.

- **fsp_err_t RM_WIFI_WINC1500_AccessControlListDelete(rm_wifi_ctrl_t * const p_ctrl, uint8_t const * const p_mac)**
  Delete MAC address from Access control list.
  Implements rm_wifi_api_t::accessControlListDelete
  This function performs the following tasks:
  - Deletes specified MAC address in access control list.

  Return value:

FSP_SUCCESS	Successfully deleted the mac address from access control list.
FSP_ERR_NOT_OPEN	Driver not opened.
FSP_ERR_UNSUPPORTED	Operation is not supported.
FSP_ERR_ASSERTION	Argument NULL is passed.

- **fsp_err_t RM_WIFI_WINC1500_MACAddressGet(rm_wifi_ctrl_t * const p_ctrl, uint8_t * const p_mac)**
  Get MAC address of Wi-Fi module.
  Implements rm_wifi_api_t::getMACAddress
  This function performs the following tasks:
  - Reads configured MAC address of the Wi-Fi module.

  Return values:

FSP_SUCCESS	Successfully reads the mac address.
FSP_ERR_NOT_OPEN	Driver not open.
FSP_ERR_ASSERTION	Argument NULL is passed.

- **fsp_err_t RM_WIFI_WINC1500_MACAddressSet(rm_wifi_ctrl_t * const p_ctrl, uint8_t const * const p_mac)**
  Set MAC address of Wi-Fi module.
  Implements rm_wifi_api_t::setMACAddress
  This function performs the following tasks:
  - Configures MAC address of the Wi-Fi module.

  Return value:

FSP_SUCCESS	Successfully deleted the mac address from access control list.
FSP_ERR_UNSUPPORTED	Functionality is not supported.
FSP_ERR_NOT_OPEN	Driver not opened.
FSP_ERR_INVALID_HW_CONDITION	Error while sending information requests.
FSP_ERR_TIMEOUT	Timeout while waiting for SPI bus access.
FSP_ERR_ABORTED	Error while waiting for SPI bus access.

- **fsp_err_t RM_WIFI_WINC1500_VersionGet(fsp_version_t * const p_version)**
  Set driver version based on compile time macros.
  Implements rm_wifi_api_t::versionGet.
  This function performs the following tasks:
  - Returns driver version.

  Return value:

FSP_SUCCESS	Success.
FSP_ERR_ASSERTION	Argument NULL is passed.

2.3 Wi-Fi On Chip Networking Stack

2.3.1 Summary

The on-chip networking stack APIs can be used to configure the Wi-Fi module when using an on-chip networking stack, which helps to configure the IP address for the interface and start/stop DHCP server (when configured in the AP mode).

On-Chip Networking Stack Support Wi-Fi Module APIs are summarized in the following section.

2.3.2 Interface APIs

The Wi-Fi On Chip Networking Stack interface APIs are defined in `rm_wifi_onchip_stack_api_t` structure.

2.3.2.1 open

This API calls WiFi driver `open` API which initializes the WiFi module. Initialize driver configuration, enable the driver link, enable interrupts and make device ready for data transfer.

2.3.2.2 close

This API calls the WiFi driver `close` API which un-initializes the WiFi module. Close the driver, disable the driver link and disable interrupt.

2.3.2.3 ipAddressCfg

This API configures the IP address of the interface using an on-chip networking stack. It provides facility configure static IP address or using DHCP.

2.3.2.4 dhcpServerStart

This API starts the DHCP server on the interface (when configured in AP mode) using on-chip networking stack. It takes the range of IP addresses to be used by DHCP server.

2.3.2.5 dhcpServerStop

This API stops the DHCP server.

2.3.3 Data structures

- `rm_wifi_onchip_stack_ip_cfg_t`
- `rm_wifi_onchip_stack_cfg_t`
- `rm_wifi_onchip_stack_ctrl_t`
- `rm_wifi_onchip_stack_instance`

2.3.4 Enumerations

- `rm_wifi_onchip_stack_ip_addr_mode_t`

2.3.5 Defines

- `#define RM_WIFI_ONCHIP_STACK_API_VER_MAJOR (1U)`

  Major Version of the API defined in WiFi On-Chip Stack.
- `#define RM_WIFI_ONCHIP_STACK_API_VER_MINOR (0U)`

  Minor Version of the API defined in WiFi On-Chip Stack.

2.4 Socket WIFI Interface

RTOS-integrated Socket WiFi Interface.

2.4.1 Summary

The Socket WiFi Interface provides access to On-Chip stack BSD Socket API.

2.4.2 Functions

- socket
- close
- bind
- listen
- connect
- accept
- send
- recv
- sendto
- recvfrom
- setsockopt
- getsockopt
- select

2.4.3 Interface API

The Socket Wi-Fi Interface APIs are defined in `rm_socket_api_t` structure.

2.4.3.1 open

Function which initializes the network interface for data transfers. Initial driver configuration, enable the driver link, enable interrupts and make device ready for data transfer.

2.4.3.2 close

Function which un-initialize the network interface and may put it in low power mode or power it off. Close the driver, disable the driver link and disable interrupt.

2.4.3.3 versionGet

Gets version and stores it in provided pointer p_version.

2.4.4 Data structures

- in_addr
- sockaddr
- sockaddr_in
- sf_socket_ctrl_t

- sf_socket_cfg_t
- sf_socket_instance_t

2.4.5 Typedefs

- socklen_t

2.4.6 Defines

- `#define SF_SOCKET_WIFI_ API_VER_MAJOR (1U)`

  Major Version of the API defined in WiFi Socket Interface.
- `#define SF_SOCKET_WIFI_ API_VER_MINOR (0U)`

  Minor Version of the API defined in WiFi Socket Interface.

3.Setting examples

3.1 ATWINC1500 on EK-RA6M3 – FreeRTOS+TCP stack mode

This section describes an example of connections and configuration details used by ATWINC1500 WiFi module on Renesas Evaluation Kit EK-RA6M3.

The FreeRTOS+TCP stack will be used.

The Renesas Evaluation Kit EK-RA6M3 is an easy way to access the entire RA Platform, enabling full development using the vast majority of all Flexible Software Package (FSP) functions.

Figure 3 shows the orientation of the ATWINC1500 module adapter board when connected to PMOD1 (J26) on the EK-RA6M3 Ver.1.

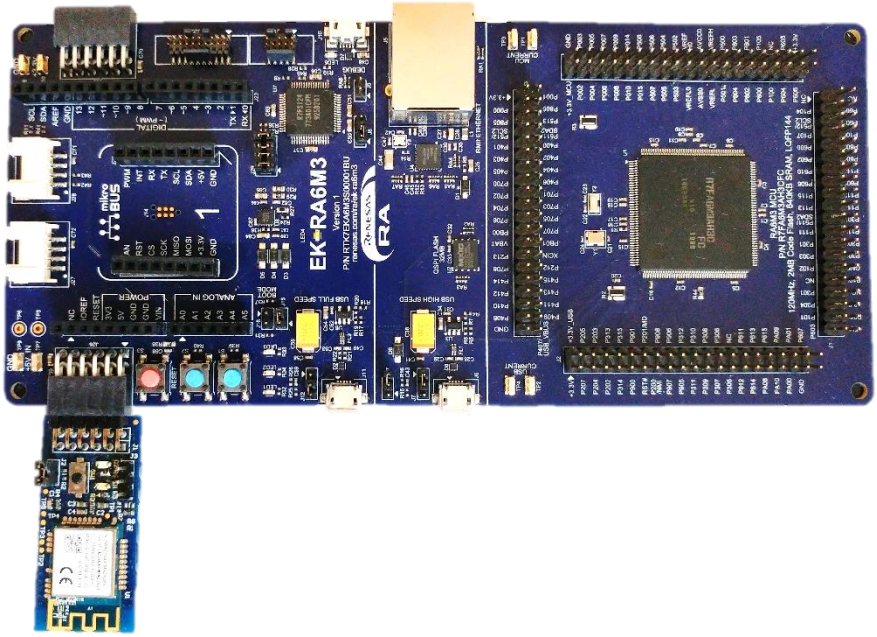

Figure 3: EK-RA6M3 Board with the WINC15X0 module connected.

3.1.1 Heap configuration

The FreeRTOS+TCP stack requires the implementation of a random number generation function named "`xApplicationGetRandomNumber()`". A weak defined implementation has been included with the driver and placed in the "`NetworkInterface.c`" file. The example implementation uses the standard `rand()` function that requires some heap space.

Allocate some heap space in the BSP tab of the configuration.xml as shown in Figure 4.

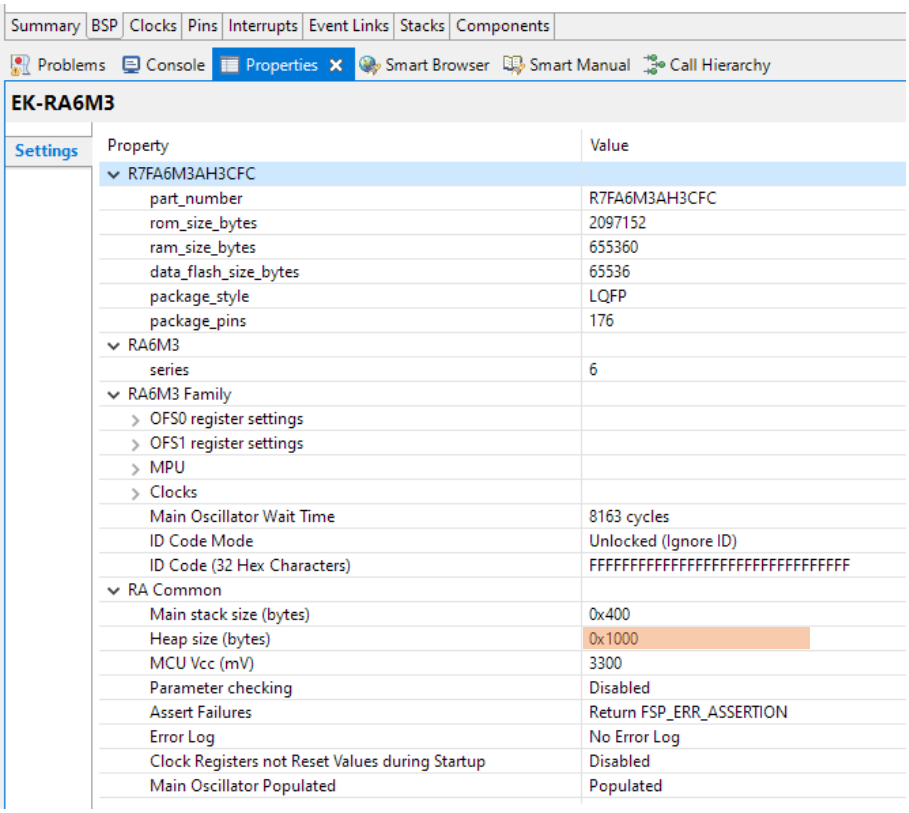

Figure 4: Add some heap space to the project.

3.1.2 Stack panel setup

Create a new thread from the Stacks panel and add a "**FreeRTOS+TCP**" instance (**New Stack /Networking /FreeRTOS+TCP**) as shown in Figure 5.

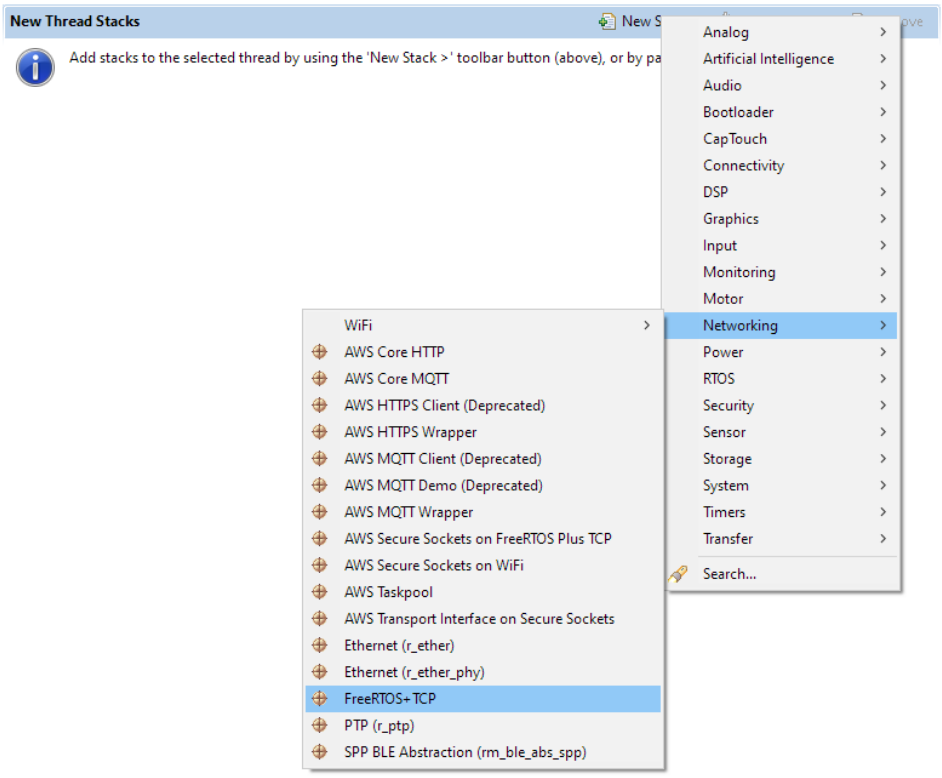

Figure 5: Add a **"FreeRTOS+TCP"** Instance.

Under the "**FreeRTOS+TCP**" instance add the "**ATWINC1500 WiFi**" as shown in Figure 6.

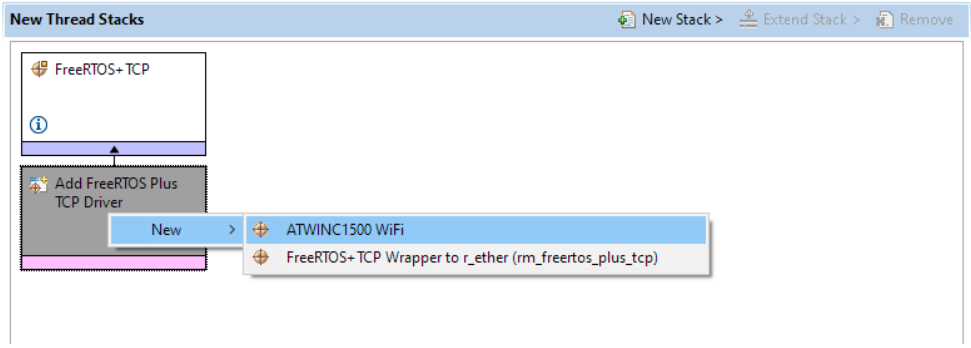

Figure 6: Adding the **"ATWINC1500 Wi-Fi"** instance.

Now a new Heap instance must be added to the New Thread Stacks.

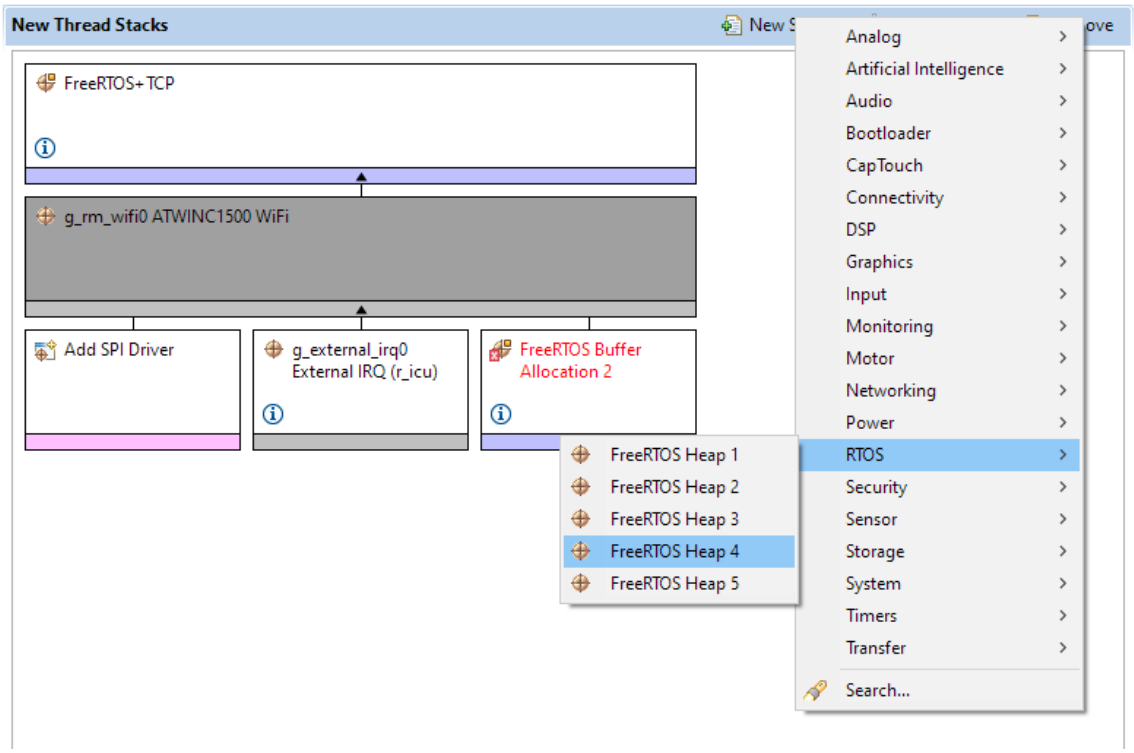

Figure 7: Adding, for example, the **"Heap 4"**.

The driver communicates with the module through an SPI Interface; hence, the FSP configurator automatically suggest selecting whether to use the driver for the *SPI* or the *SCI.* Fox example, the PMOD1 connector of the EK-RA6M3 exposes one of the RA6M3 *SCI*, the *SCI9*.

Left click on the box and choose ***New > SPI (r_sci_spi)***.

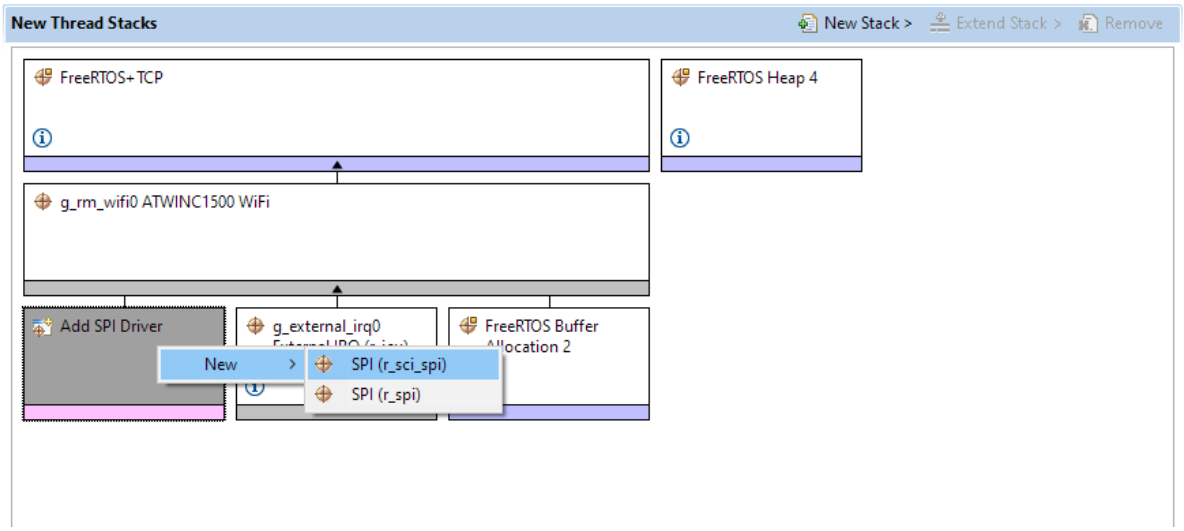

Figure 8: SPI Driver selection.

Figure 9 shows the complete FreeRTOS+TCP stack instance.

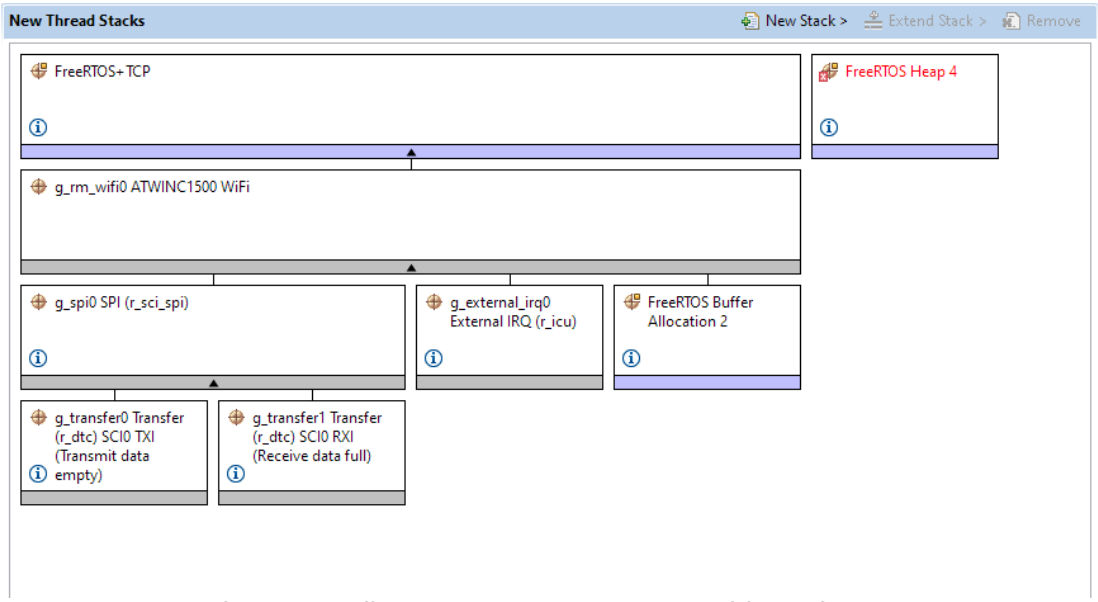

Figure 9: Full FreeRTOS+TCP Instance hierarchy.

Lastly, we need to:

- activate the xTimerPendFunctionCall() API of the FreeRTOS, required by the driver
- increase the thread's stack
- enable the "Support Dynamic Allocation"
- edit the "Total Heap Size", 0x5000 is used for a simple test project, greater values are required for high level network stacks (0x20000 if MQTT is used, for example).

Select the thread "New Thread" and configure the properties as the followings:

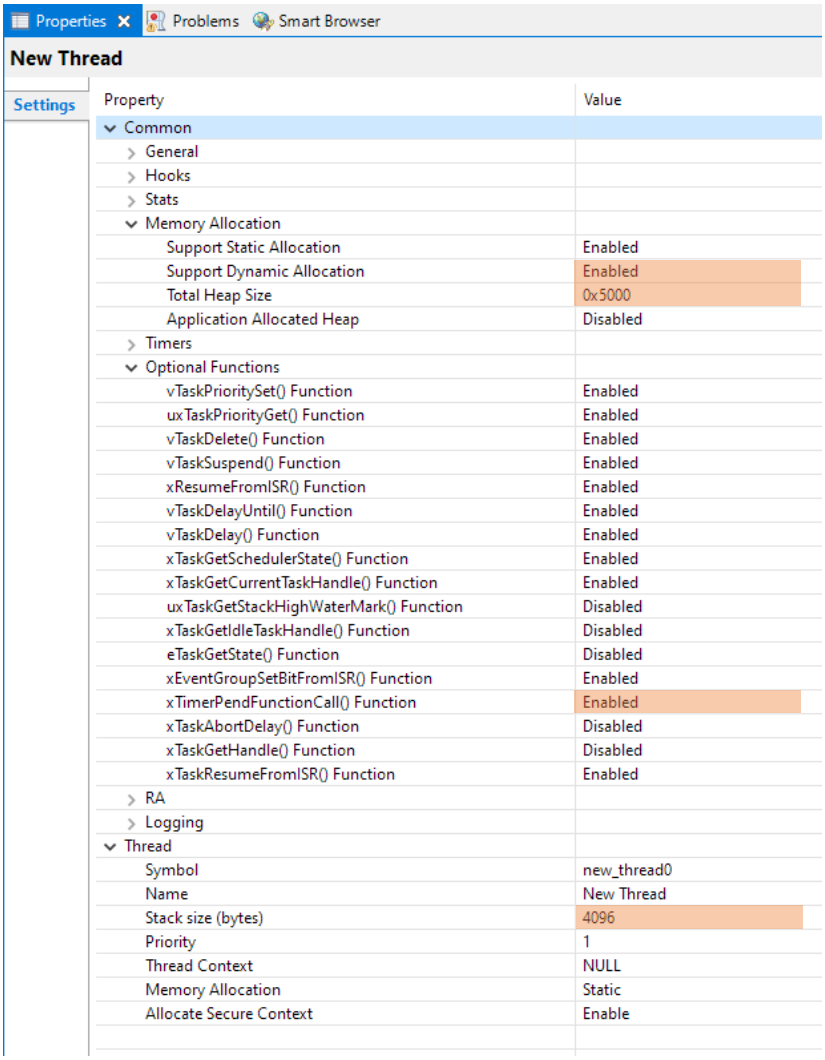

Figure 10: Thread properties.

We have now completed the selection and the configuration of the needed FSP components. We now need to configure some of the component's parameters.

3.1.3 FSP Components configuration

3.1.3.1 g_rm_wifi0 ATWINC1500 WiFi

Select the *g_rm_wifi0 ATWINC1500 WiFi* block and change its settings to reflect the following:

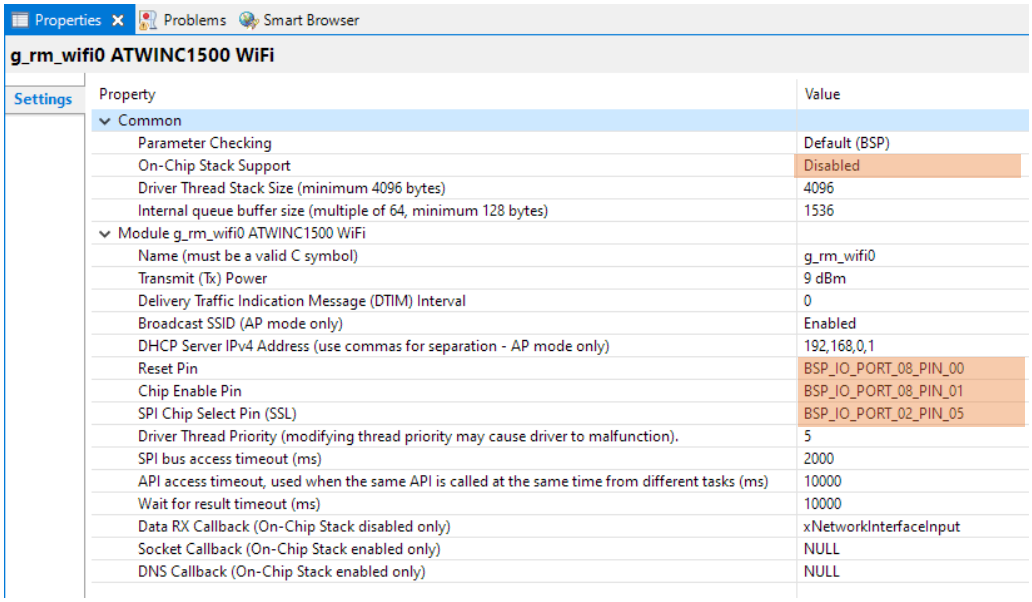

Figure 11: ATWINC1500 WiFi Driver configuration.

3.1.3.2 g_spi0 SPI (r_sci_spi)

Select the g_spi0 SPI (r_sci_spi) block and change its settings to reflect the following:

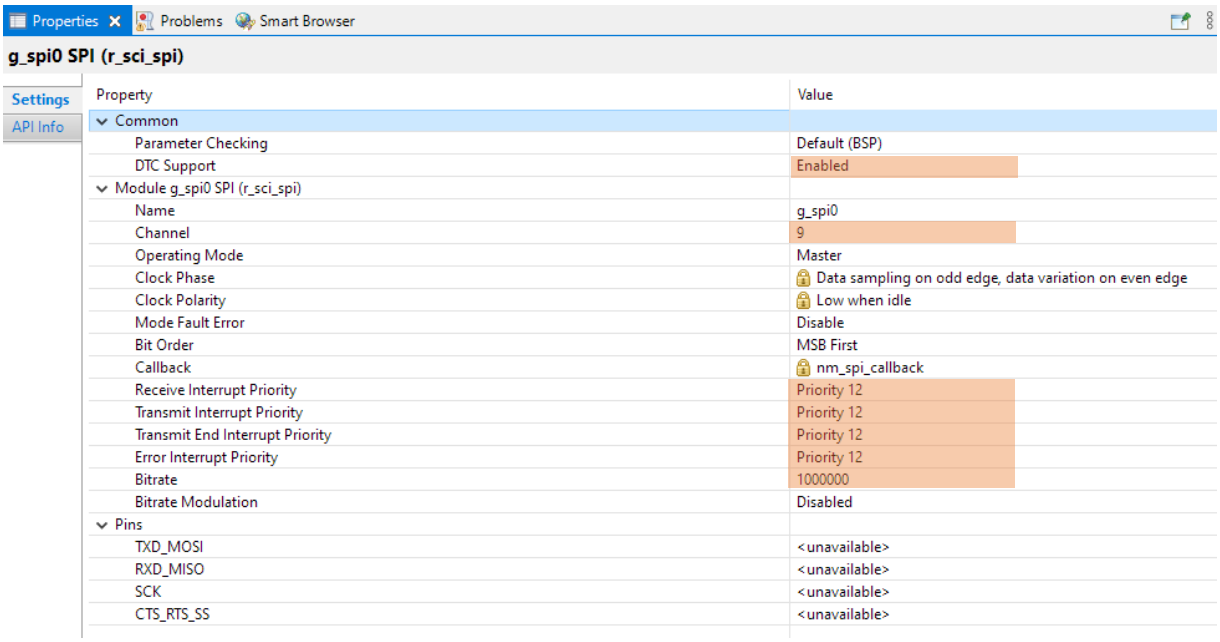

Figure 12: SPI Driver on r_sci_spi configuration.

3.1.3.3 g_external_irq0 External IRQ (r_icu)

Select the g_external_irq0 External IRQ (r_icu) block and change its settings to reflect the following:

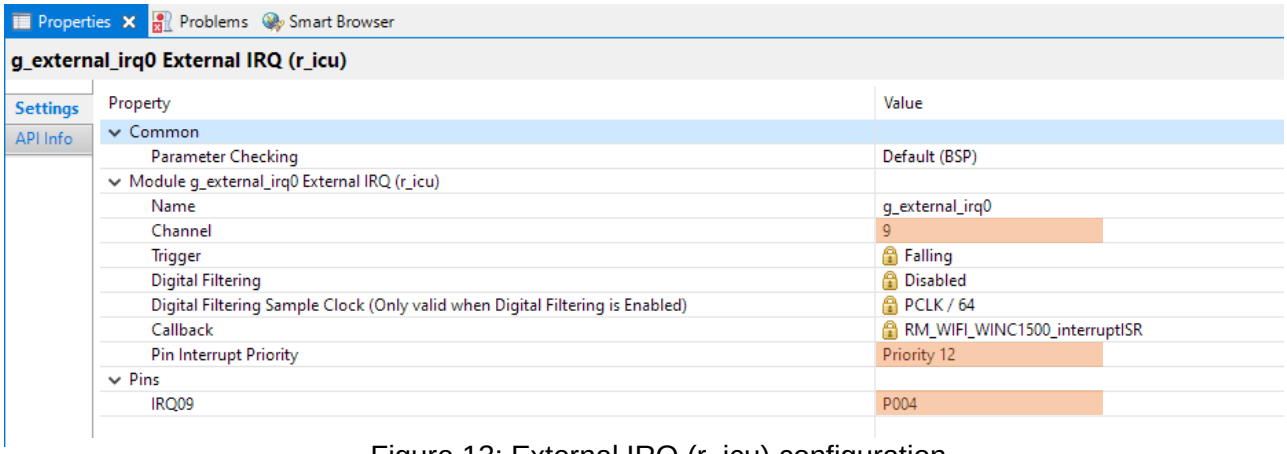

Figure 13: External IRQ (r_icu) configuration.

3.1.4 Pins and peripherals configurations

3.1.4.1 SCI9 Configuration

Set up the SCI9 for the EK-RA6M3

Go to **Pins tab** and from the **Pin Selection** section go to **Peripherals > Connectivity:SCI > SCI9**. Go to **Pin Configuration** section and change the settings to reflect the Figure 14.

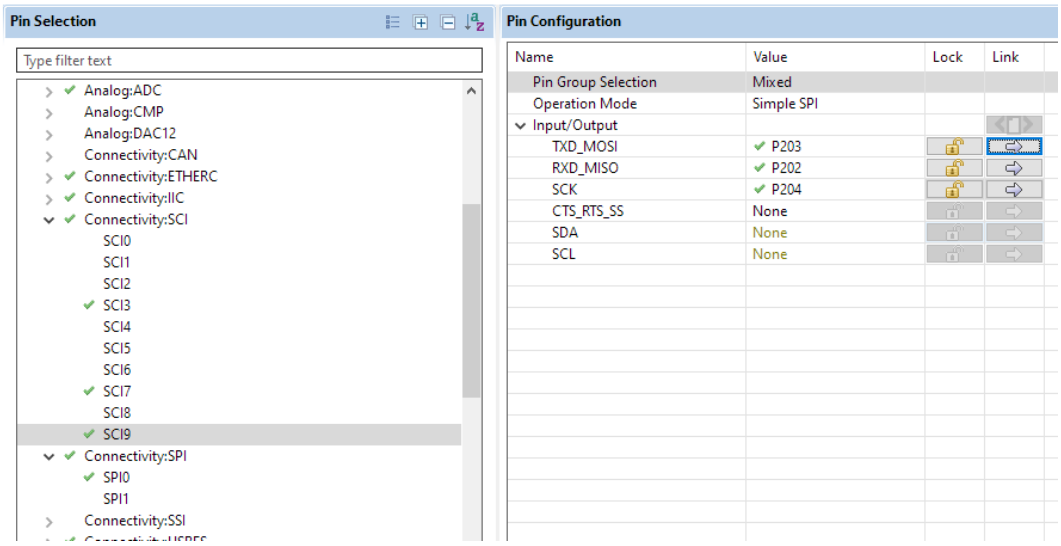

Figure 14: SCI9 Pins configuration.

Note that, as default, the required pins are configured for the SPI1 peripheral, so SPI1 must be firstly disabled.

Set up the SCI9 Chip Select pin P205 for the EK-RA6M3

Go to **Pins tab** and from the **Pin Selection** section go to **Ports > P2 > P205**. Go to **Pin Configuration** section and change the settings to setup the chip select pin for the SCI:

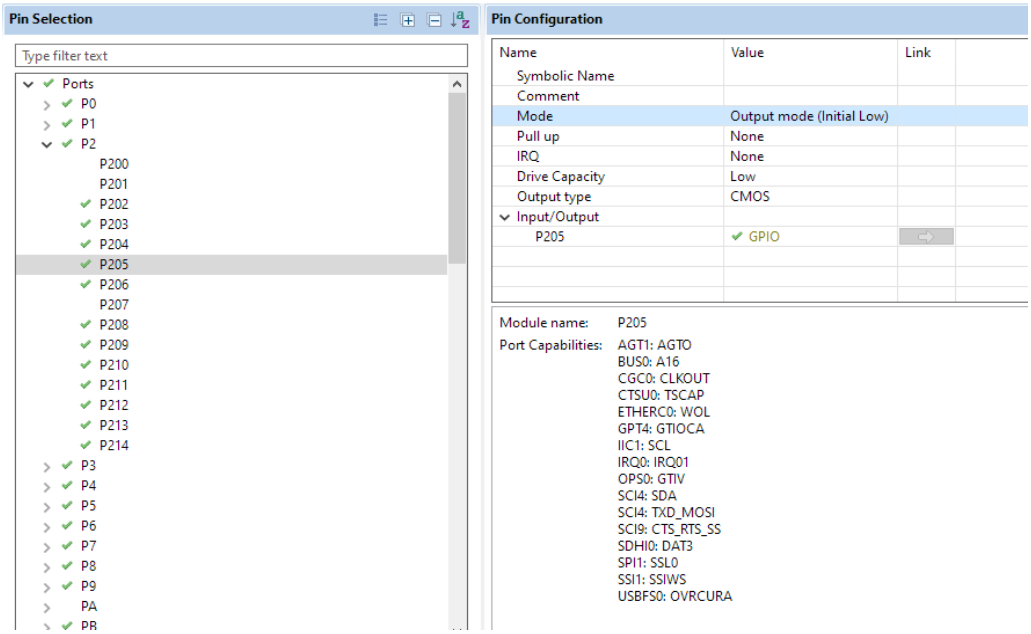

Figure 15: Chip Select pin configuration.

3.1.4.2 IRQ Configuration

Set up the IRQ pin P004 for the EK-RA6M3. This is IRQ09-DS

Go to **Pins tab** and from the **Pin Selection** section go to **Ports > P0 > P004.** Go to **Pin Configuration** section and change the settings to setup the IRQ for the Wi-Fi module:

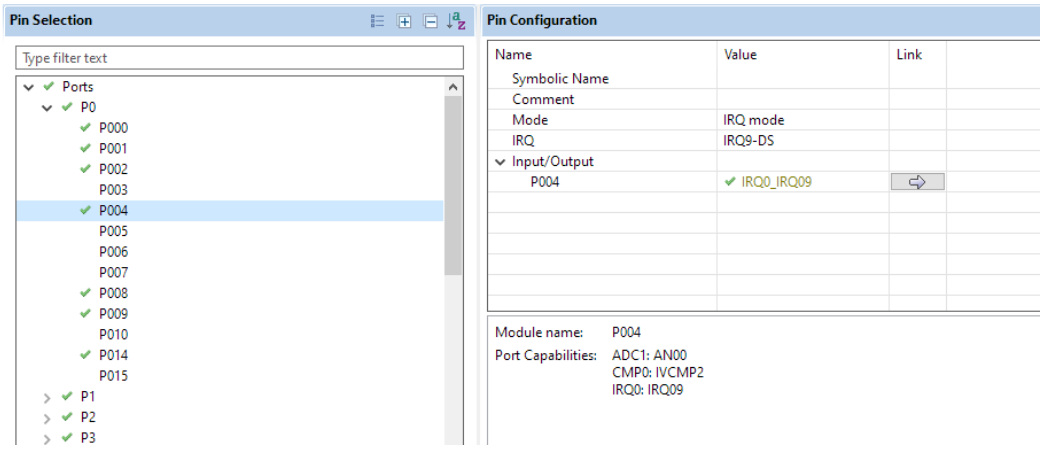

Figure 16: IRQ pin configuration.

3.1.4.3 ATWINC1500 Reset and CE pins

Set up the Reset Pin for the module, P800 on the EK-RA6M3.

Go to **Pins tab** and from the **Pin Selection** section go to **Ports > P8 > P801**. Go to **Pin Configuration** section and change the settings to setup the Reset Pin for the Wi-Fi module:

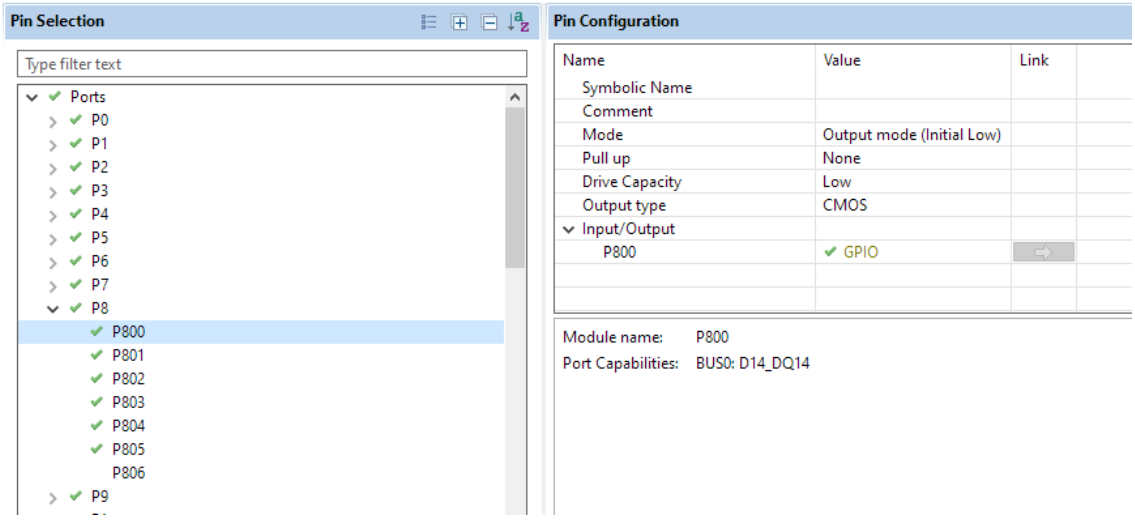

Figure 17: Reset pin configuration.

Set up the Chip Enable pin for the module, P801 on the EK-RA6M3.

Go to **Pins tab** and from the **Pin Selection** section go to **Ports > P8 > P801**. Go to **Pin Configuration** section and change the settings to setup the Chip Enable Pin for the Wi-Fi module:

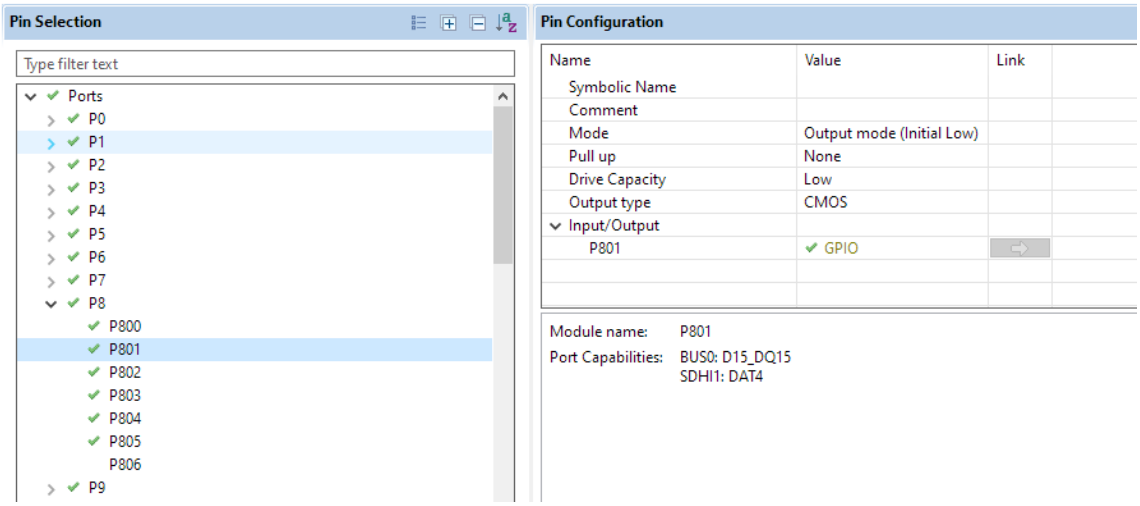

Figure 18: Chip Enable pin configuration.

3.1.5 Callbacks

Some user's callbacks must be provided in order to complete the example project:

```
const char *pcApplicationHostnameHook( void )
{
    return "my_hostname";
}

void vApplicationIPNetworkEventHook( eIPCallbackEvent_t eNetworkEvent )
{
    if( eNetworkEvent == eNetworkUp )
    {
        /* Connected! */
    }
    else
    {
        /* Disconnected! */
    }
}
```

3.2 ATWINC1500 on EK-RA6M3 – On-Chip stack mode

This section describes an example of the On-Chip stack configuration using again a Renesas Evaluation Kit EK-RA6M3

Figure 19 shows the orientation of the ATWINC1500 module adapter board when connected to PMOD1 (J26) on the EK-RA6M3 Ver.1.

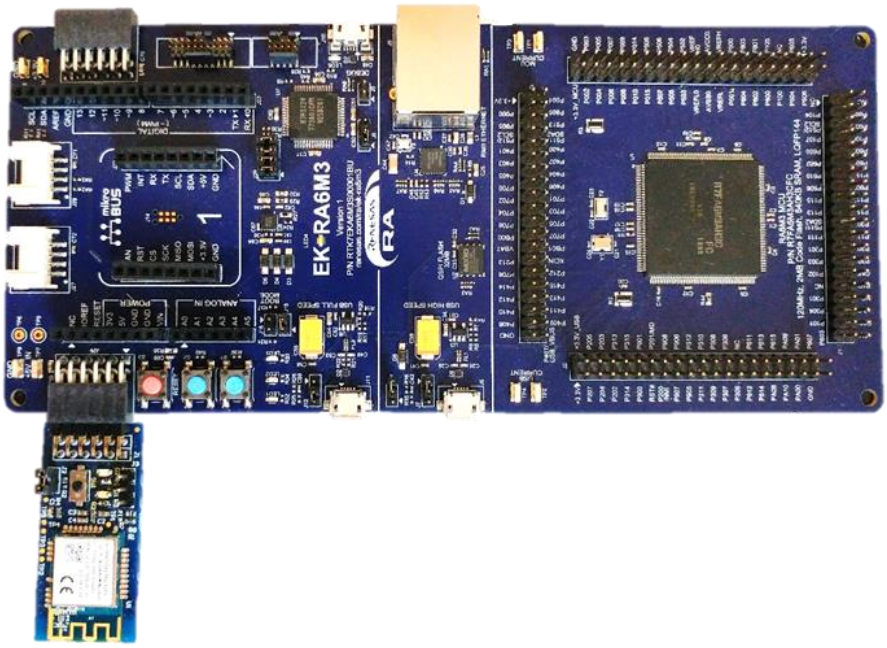

Figure 19: EK-RA6M3 Board with the WINC15X0 module connected.

3.2.1 Stack panel setup

Create a new thread from the Stacks panel and add a **"BSP Socket using On-Chip stack on ATWINC1500 WiFi"** Instance as shown in Figure 20.

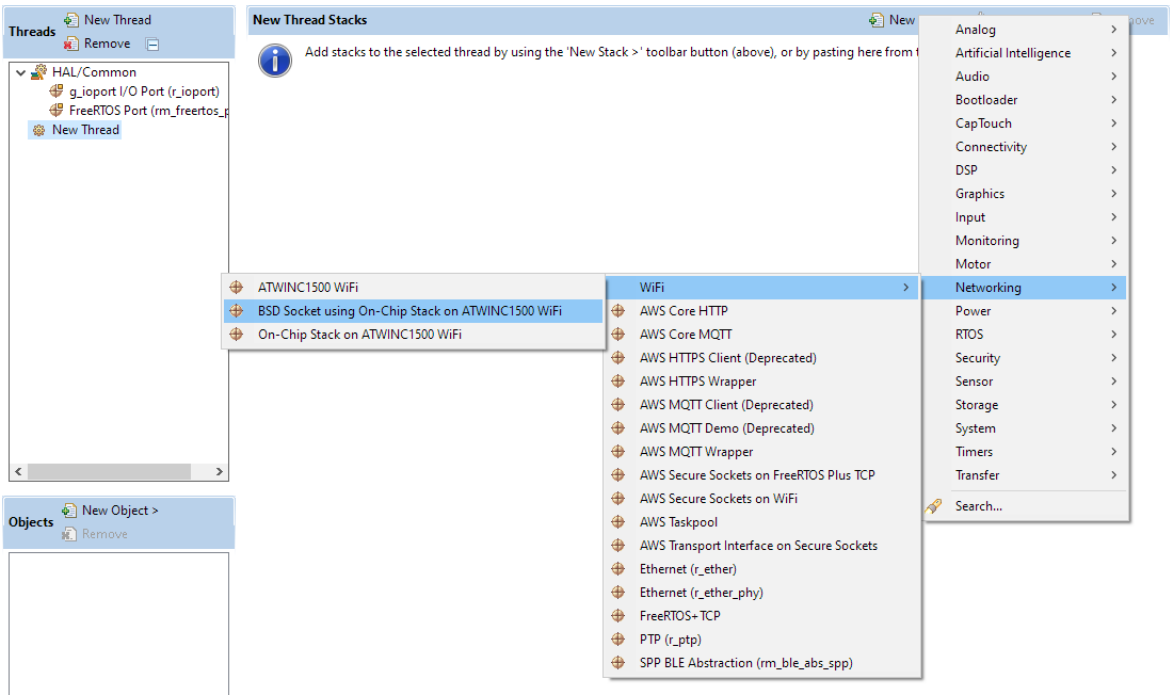

Figure 20: Add a BSP Socket using WINC1500 On-Chip Stack Instance.

The FSP configurator will build the BSP Socket instance automatically as far as it can. Figure 21 shows what the FSP configurator should be able to build.

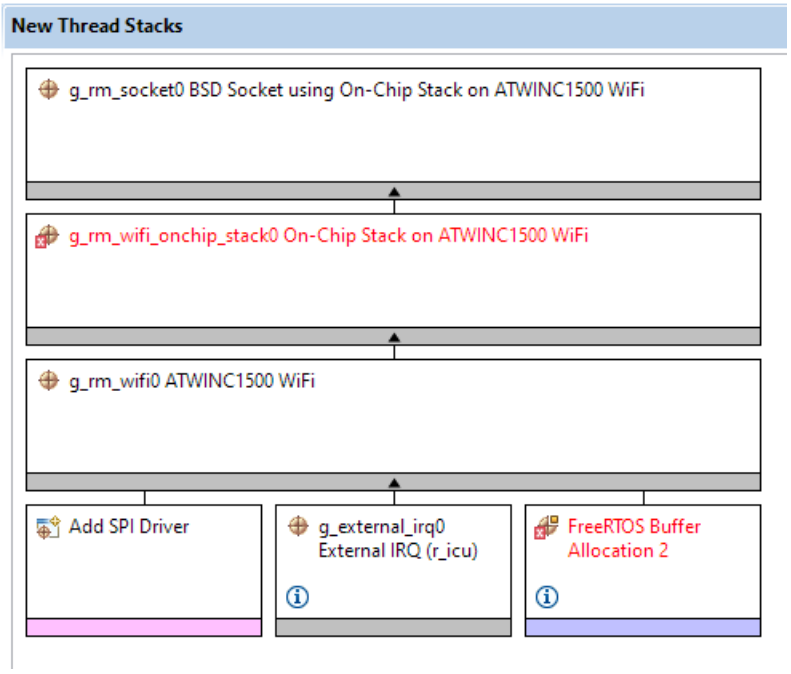

Figure 21: **"BSP Socket using On-Chip Stack on WINC1500 WiFi"** Instance hierarchy.

For the On-Chip stack the **"FreeRTOS Buffer Allocation 2"** module is not required so it can be removed.

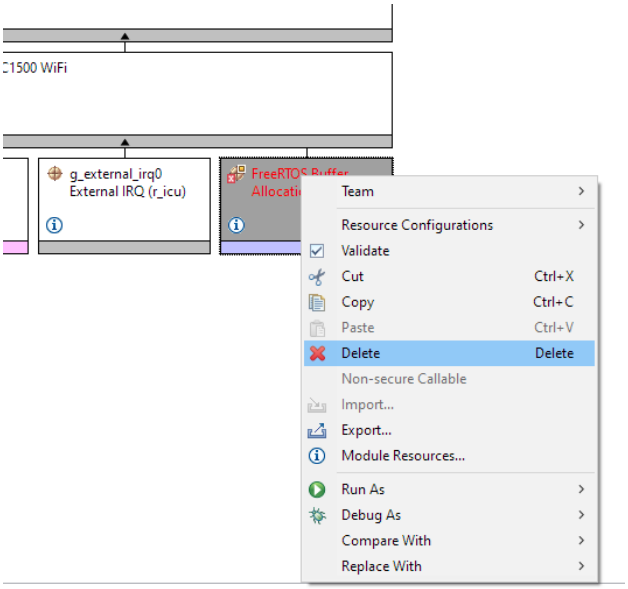

Figure 22: **"FreeRTOS Buffer Allocation 2"** removal.

The driver communicates with the module through an SPI Interface; hence, the FSP configurator automatically suggest selecting whether to use the driver for the *SPI* or the *SCI.* Fox example, the PMOD1 connector of the EK-RA6M3 exposes one of the RA6M3 *SCI*, the *SCI9*.

Left click on the box and choose ***New > SPI (r_sci_spi)***.

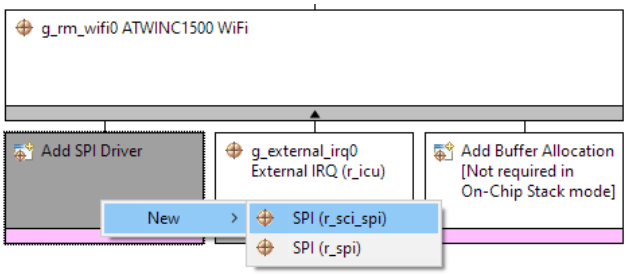

Figure 23: SPI Driver selection.

Figure 24 shows the complete BSP socket instance.

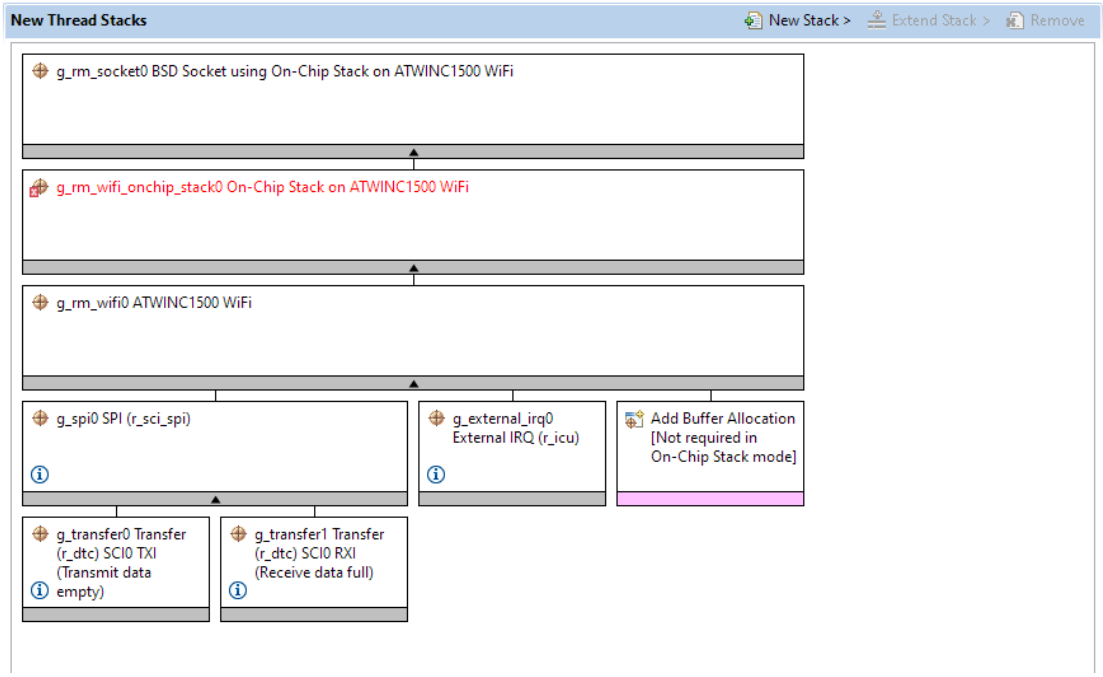

Figure 24: Full BSP Socket Instance hierarchy.

Lastly, we need to activate the xTimerPendFunctionCall() API of the FreeRTOS, required by the driver, and increase the stack. Select the thread "New Thread" and configure the properties as the followings:

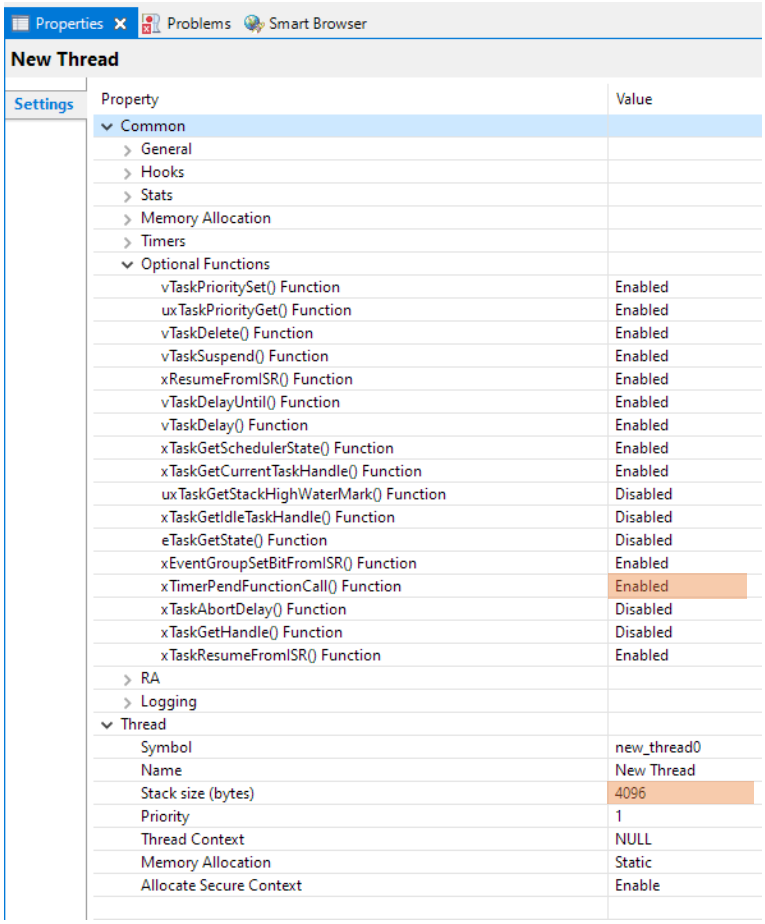

Figure 25: Thread's properties.

We have now completed the selection and the configuration of the needed FSP components. We now need to configure some of the component's parameters.

3.2.2 FSP Components configuration

3.2.2.1 g_rm_wifi0 ATWINC1500 Wi-Fi

Select the *g_rm_wifi0 ATWINC1500 Wi-Fi* block and change its settings to reflect the following:

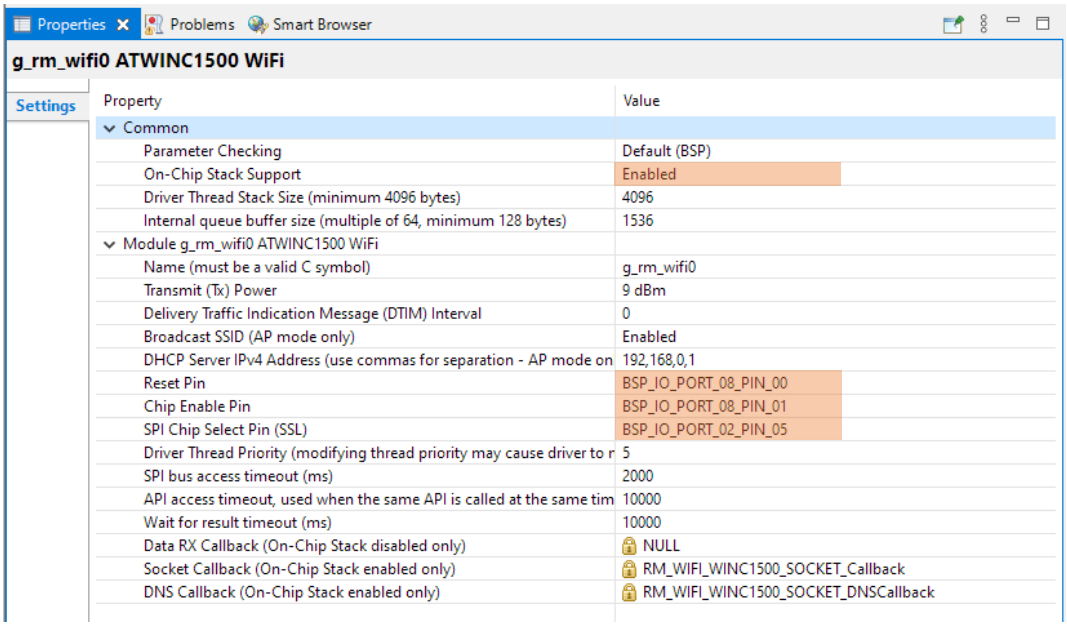

Figure 26: ATWINC1500 Wi-Fi configuration.

3.2.2.2 g_spi0 SPI (r_sci_spi)

Select the g_spi0 SPI (r_sci_spi) block and change its settings to reflect the following:

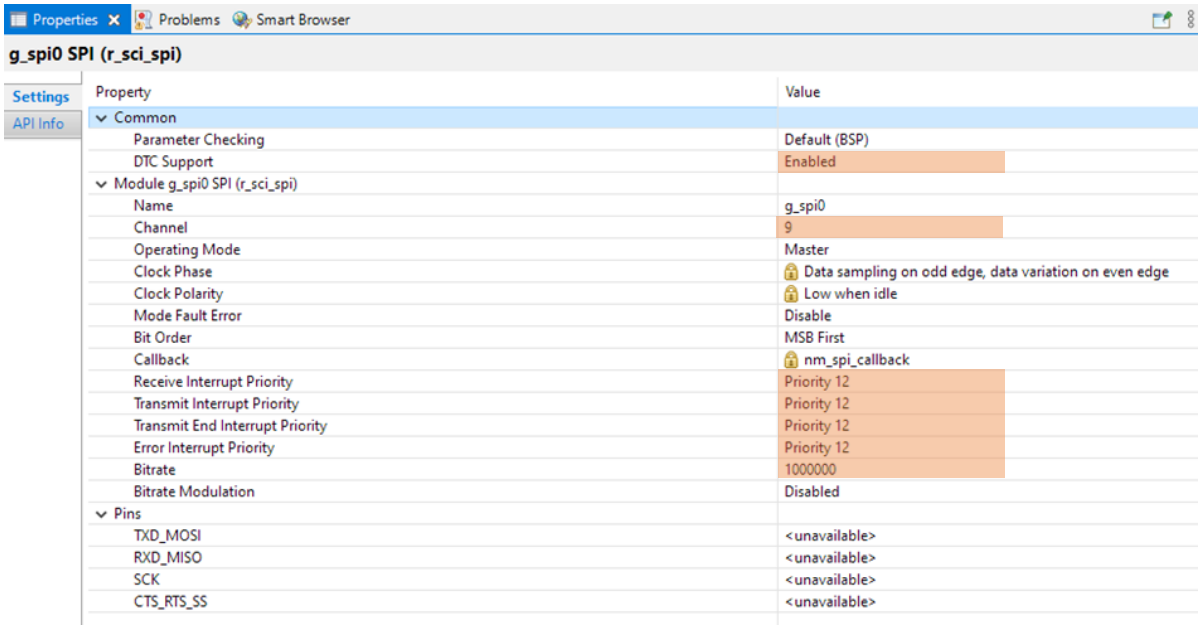

Figure 27: SPI (r_sci_spi) configuration.

3.2.2.3 g_external_irq0 External IRQ (r_icu)

Select the g_external_irq0 External IRQ (r_icu) block and change its settings to reflect the following:

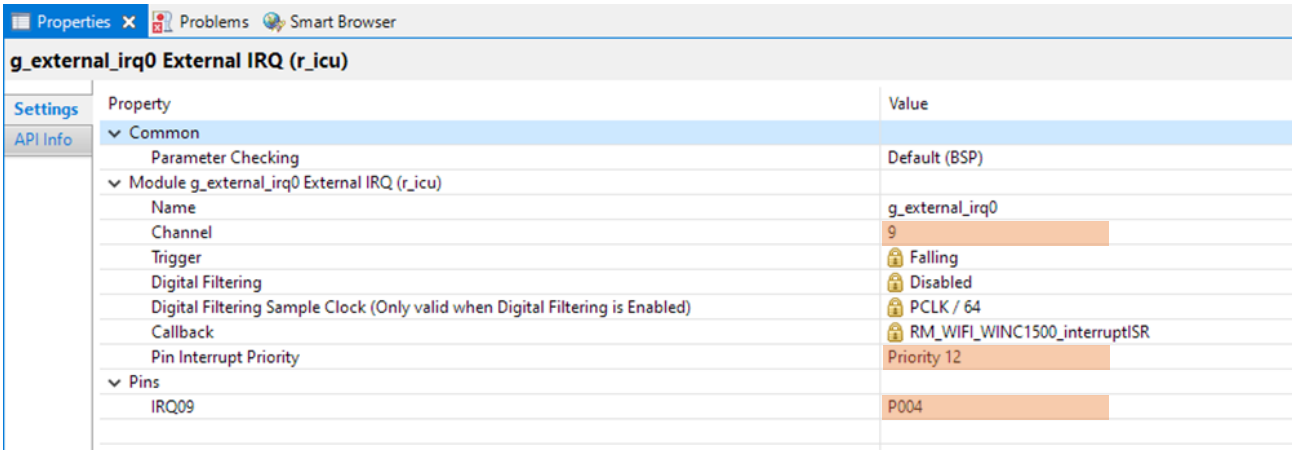

Figure 28: External IRQ (r_icu) configuration.

3.2.3 Pins and peripherals configurations

3.2.3.1 SCI9 Configuration

Set up the SCI9 for the EK-RA6M3

Go to **Pins tab** and from the **Pin Selection** section go to **Peripherals > Connectivity:SCI > SCI9**. Go to **Pin Configuration** section and change the settings to reflect the Figure 29Figure 14.

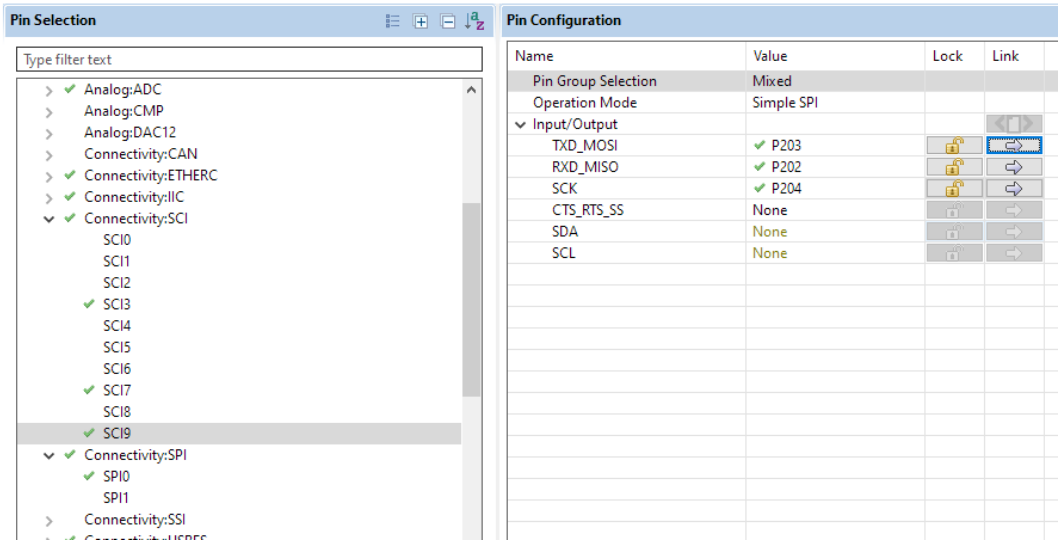

Figure 29: SCI9 Pins configuration.

Note that, as default, the required pins are configured for the SPI1 peripheral, so SPI1 must be firstly disabled.

Set up the SCI9 Chip Select pin P205 for the EK-RA6M3

Go to **Pins tab** and from the **Pin Selection** section go to **Ports > P2 > P205**. Go to **Pin Configuration** section and change the settings to setup the chip select pin for the SCI:

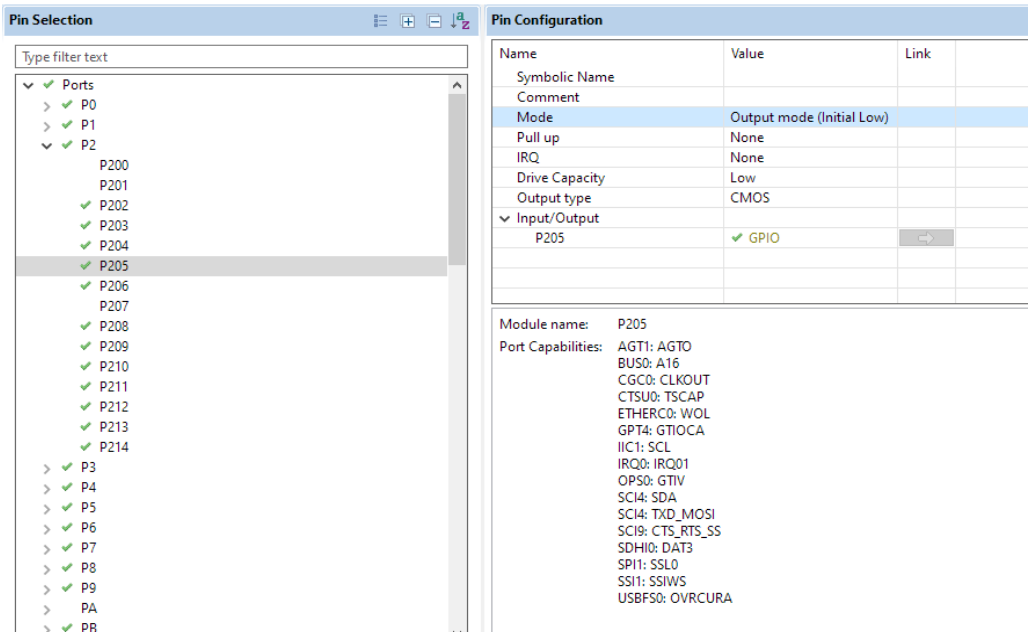

Figure 30: Chip Select pin configuration.

3.2.3.2 IRQ Configuration

Set up the IRQ pin P004 for the EK-RA6M3. This is IRQ09-DS

Go to **Pins tab** and from the **Pin Selection** section go to **Ports > P0 > P004.** Go to **Pin Configuration** section and change the settings to setup the IRQ for the Wi-Fi module:

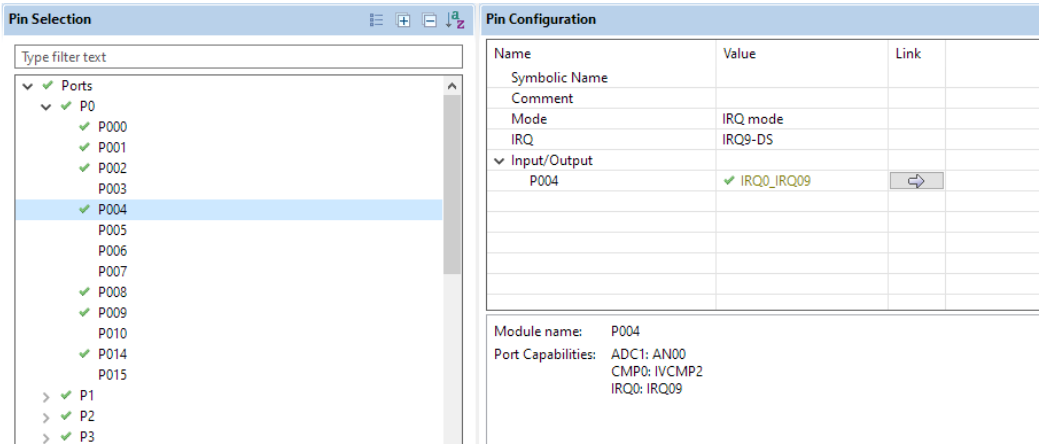

Figure 31: IRQ pin configuration.

3.2.3.3 ATWINC1500 Reset and CE pins

Set up the Reset Pin for the module, P800 on the EK-RA6M3.

Go to **Pins tab** and from the **Pin Selection** section go to **Ports > P8 > P801**. Go to **Pin Configuration** section and change the settings to setup the Reset Pin for the Wi-Fi module:

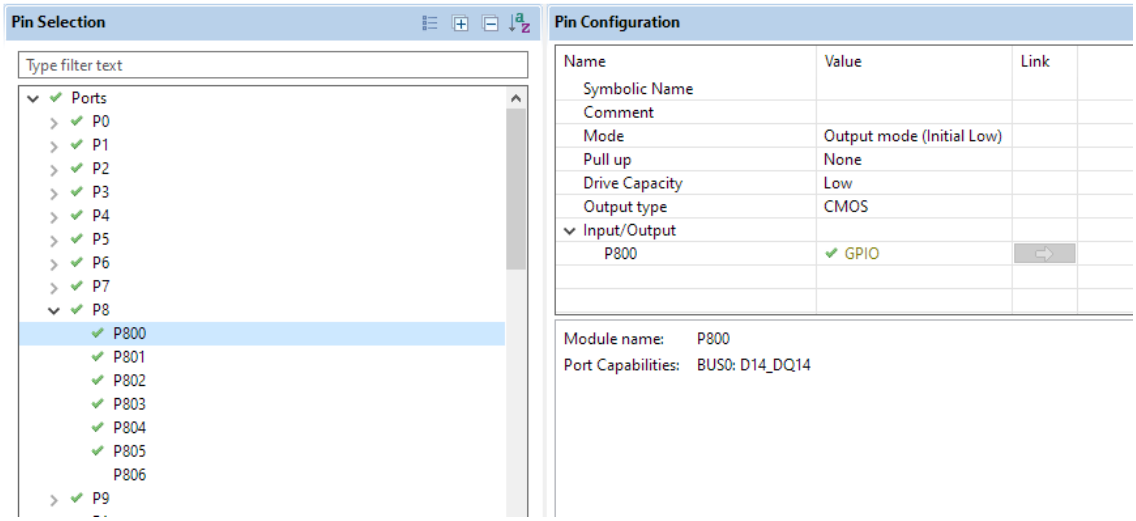

Figure 32: Reset pin configuration.

Set up the Chip Enable pin for the module, P801 on the EK-RA6M3.

Go to **Pins tab** and from the **Pin Selection** section go to **Ports > P8 > P801**. Go to **Pin Configuration** section and change the settings to setup the Chip Enable Pin for the Wi-Fi module:

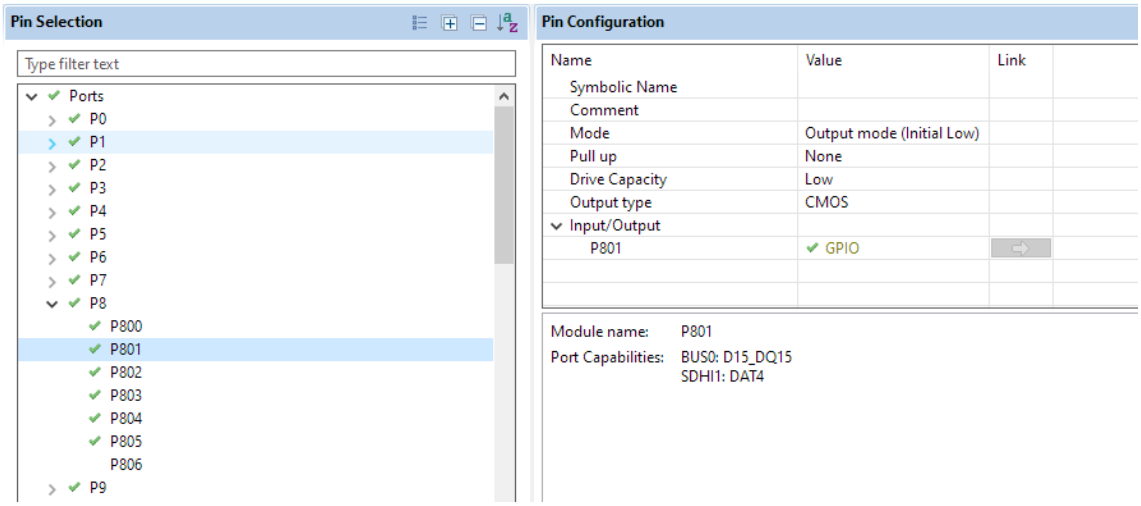

Figure 33: Chip Enable pin configuration.