ATWINC15X0
Wi-Fi Add-on Component

User's Manual

All information contained in these materials, including products and product specifications, represents information on the product at the time of publication and is subject to change by RELOC s.r.l. without notice.

Outline

Revisions

REVISION	DATE	DESCRIPTION	STATUS	AUTHOR	REVISER
0.1	03/07/2017	Document created.	draft	C. Tagliaferri	
0.2	04/07/2017	Document revised.	draft	C. Tagliaferri	A. Ricci
1.0	04/07/2017	Minor revisions.	release	C. Tagliaferri	A. Ricci
1.1	28/05/2018	SSP1.4.0 support added.	release	C. Tagliaferri	A. Ricci
1.2	03/10/2018	Add multicast APIs. Updated return values of the APIs.	release	C. Tagliaferri	A. Ricci
1.3	30/04/2019	Document updated for SSP 1.6.0 release.	release	C. Tagliaferri	A. Ricci

Disclaimer

All rights strictly reserved. Reproduction or issue to third parties in any form is not permitted without written authorization from RELOC s.r.l.

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 Synergy™ Wi-Fi Framework – User's Manual", document number: R11UM0050EU0108, Jun 9, 2017.

[2] Renesas Electronics, "Renesas Synergy™ Platform Wi-Fi Framework Module Guide", document number: R11AN0252EU0100, Aug 20, 2018.

[3] Renesas Electronics, "Renesas Synergy™ Software Package (SSP) v1.6.0 – User's Manual", Rev. 1.01, Mar 27, 2019.

[4] RELOC s.r.l., PMOD-WM1A webpage

http://www.reloc.it/products/pmod-wi-fi-atwinc1500/.

[5] Renesas Electronics, Renesas Synergy Gallery webpage

https://www.renesas.com/en-eu/products/synergy/gallery.html.

1. Overview

The Synergy Wi-Fi framework [1][2] consists of the following logical blocks:

- SF Wi-Fi APIs
- Network stack abstraction layer
- Wi-Fi device driver (vendor provided driver).
- SSP HAL interface

It also includes the following blocks to support the BSD Socket APIs in making use of an on-chip networking stack:

- On Chip Stack APIs
- Socket APIs.

Figure 1 provides an overview of the Synergy Wi-Fi framework layered architecture. Details are reported in [1], [2] and [3].

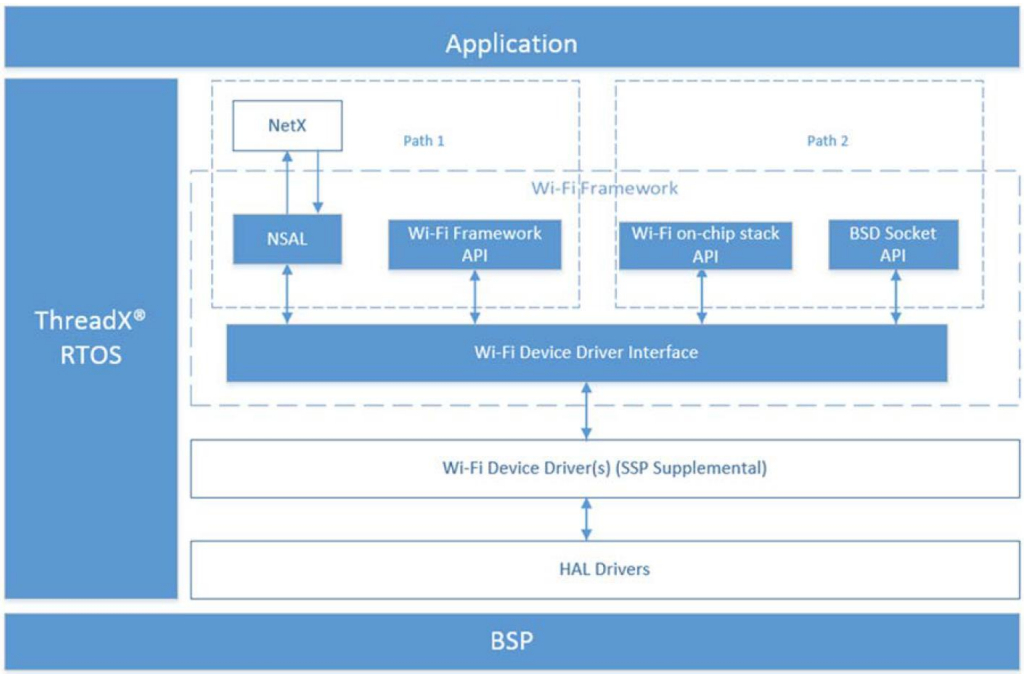

Figure 1: Synergy Wi-Fi Framework layered architecture.

1.1 Supported SSPs

ATWINC15X0 Synergy Framework Wi-Fi Add-on Component currently supports SSP 1.2.x, SSP 1.3.x, SSP 1.4.x, SSP 1.5.x, SSP 1.6.x releases (visit [4] and [5] for downloads and updates).

2. API References: Wi-Fi Framework on ATWINC1500

2.1 Wi-Fi Framework on ATWINC1500

2.1.1 Summary

The Wi-Fi framework module provides a generic interface for the network stack and applications irrespective of the Wi-Fi module. Below are the APIs exposed by the framework, supported by ATWINC1500 Wi-Fi module.

2.1.2 Data Structures

- `st_sf_wifi_winc1500_cfg` with data fields:
  - `sf_spi_instance_t const * p_sf_spi`: Framework SPI Interface used for Wi-Fi communications;
  - `external_irq_instance_t const * p_irq`: IRQ Interface used for Wi-Fi communications;
  - `ioport_port_pin_t pin_reset`: Pin used for resetting module;
  - `ioport_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;
  - `ULONG bus_access_timeout_ms`: SPI Bus access timeout in milliseconds;
  - `ULONG api_access_timeout_ms`: API access timeout (no simultaneous use of a single API) in milliseconds;
  - `ULONG wait_result_timeout_ms`: Wait for result timeout in milliseconds;
  - `tpfAppSocketCb socket_callback`: Callback for socket APIs;
  - `tpfAppSocketCb dns_callback`: Callback for DNS APIs.

2.1.2.1 Struct Reference

- `#include <sf_wifi_winc1500.h>`

2.1.3 Defines

- `#define SF_WIFI_WINC1500_CODE_VERSION_MAJOR (2)`
  Major version of code that implements the API defined in WiFi Framework.
- `#define SF_WIFI_WINC1500_CODE_VERSION_MINOR (6)`
  Minor version of code that implements the API defined in WiFi Framework.

2.1.4 Interface APIs

- `ssp_err_t SF_WIFI_WINC1500_Open(sf_wifi_ctrl_t * p_ctrl, sf_wifi_cfg_t const * const p_cfg)`
  Initialize Wi-Fi module.
- `ssp_err_t SF_WIFI_WINC1500_Close(sf_wifi_ctrl_t * const p_ctrl)`
  Stop Wi-Fi module.
- `ssp_err_t SF_WIFI_WINC1500_MulticastListAdd(sf_wifi_ctrl_t * const p_ctrl, uint8_t const * const p_addr)`
  Add MAC address in multicast list.
- `ssp_err_t SF_WIFI_WINC1500_MulticastListDelete(sf_wifi_ctrl_t * const p_ctrl, uint8_t const * const p_addr)`
  Delete MAC address from multicast list.
- `ssp_err_t SF_WIFI_WINC1500_StatisticsGet(sf_wifi_ctrl_t * const p_ctrl, sf_wifi_stats_t * const p_wifi_device_stats)`
  Get the interface statistics.
- `ssp_err_t SF_WIFI_WINC1500_Transmit(sf_wifi_ctrl_t * const p_ctrl, uint8_t * const p_buf, uint32_t length)`
  Transmit data packets.
- `ssp_err_t SF_WIFI_WINC1500_ProvisioningSet(sf_wifi_ctrl_t * const p_ctrl, sf_wifi_provisioning_t const * const p_wifi_provisioning)`
  Provisions the Wi-Fi module.
- `ssp_err_t SF_WIFI_WINC1500_ProvisioningGet(sf_wifi_ctrl_t * const p_ctrl, sf_wifi_provisioning_t * const p_wifi_provisioning)`
  Reads the current Wi-Fi Provisioning information of the Wi-Fi module.
- `ssp_err_t SF_WIFI_WINC1500_InfoGet(sf_wifi_ctrl_t * const p_ctrl, sf_wifi_info_t * const p_wifi_info)`
  Get Wi-Fi module information.
- `ssp_err_t SF_WIFI_WINC1500_Scan(sf_wifi_ctrl_t * const p_ctrl, sf_wifi_scan_t * const p_scan, uint8_t * const p_cnt)`
  Scans for available APs.
- `ssp_err_t SF_WIFI_WINC1500_AccessControlListAdd(sf_wifi_ctrl_t * const p_ctrl, uint8_t const * const p_mac)`
  Add MAC address from Access control list.
- `ssp_err_t SF_WIFI_WINC1500_AccessControlListDelete(sf_wifi_ctrl_t * const p_ctrl, uint8_t const * const p_mac)`
  Delete MAC address from Access control list.
- `ssp_err_t SF_WIFI_WINC1500_MACAddressGet(sf_wifi_ctrl_t * const p_ctrl, uint8_t * const p_mac)`
  Get MAC address of Wi-Fi module.
- `ssp_err_t SF_WIFI_WINC1500_MACAddressSet(sf_wifi_ctrl_t * const p_ctrl, uint8_t const * const p_mac)`
  Set MAC address of Wi-Fi module.
- `ssp_err_t SF_WIFI_WINC1500_VersionGet(ssp_version_t * const p_version)`
  Set driver version based on compile time macros.

2.1.4.1 APIs Documentation

- **ssp_err_t SF_WIFI_WINC1500_Open(sf_wifi_ctrl_t * p_ctrl, sf_wifi_cfg_t const * const p_cfg)**
  Initializes Wi-Fi module.

Implements sf_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:

SSP_SUCCESS	Driver initialization done successfully.
SSP_ERR_ALREADY_OPEN	Wi-Fi Driver is already opened.
SSP_ERR_ASSERTION	Argument NULL is passed or ThreadX resources initialization failed.
SSP_ERR_INVALID_HW_CONDITION	Module initialization failed. Verify pin configurations.
SSP_ERR_UNSUPPORTED	WINC1500 Module firmware version unsupported.
SSP_ERR_OUT_OF_MEMORY	Not enough available space in the heap for the internal thread.
SSP_ERR_WIFI_CONFIG_FAILED	Configuration of Wi-Fi driver failed.

- **ssp_err_t SF_WIFI_WINC1500_Close(sf_wifi_ctrl_t * const p_ctrl)**
  Stops Wi-Fi module.
  Implements sf_wifi_api_t::close.
  This function performs the following tasks:
  - Disable the Interrupt and suspend the WINC1500 driver task thread.

  Return values:

SSP_SUCCESS	Driver closed successfully.
SSP_ERR_ASSERTION	Argument NULL is passed.
SSP_ERR_NOT_OPEN	Driver already closed.
SSP_ERR_WIFI_FAILED	Timeout or error while waiting for API result.
SSP_ERR_TIMEOUT	Timeout while waiting for SPI bus access.
SSP_ERR_ABORTED	Error while waiting for SPI bus access.
SSP_ERR_INVALID_HW_CONDITION	Error closing Wi-Fi driver.
SSP_ERR_INTERNAL	Error deleting threadX objects.

- **ssp_err_t SF_WIFI_WINC1500_MulticastListAdd(sf_wifi_ctrl_t * const p_ctrl, uint8_t const * const p_addr)**
  Add MAC address in multicast list.
  Implements sf_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:

SSP_SUCCESS	Successfully added the mac address to the multicast list.
SSP_ERR_NOT_OPEN	Driver not opened.
SSP_ERR_WIFI_CONFIG_FAILED	Lower level error while adding the mac address to the multicast list.
SSP_ERR_TIMEOUT	Timeout while waiting for SPI bus access.
SSP_ERR_ABORTED	Error while waiting for SPI bus access.
SSP_ERR_UNSUPPORTED	API not supported ("On-Chip Stack" mode only)

- **ssp_err_t SF_WIFI_WINC1500_MulticastListDelete(sf_wifi_ctrl_t * const p_ctrl, uint8_t const * const p_addr)**
  Delete MAC address from multicast list.
  Implements sf_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:

SSP_SUCCESS	Successfully removed the mac address from the multicast list.
SSP_ERR_NOT_OPEN	Driver not opened.
SSP_ERR_WIFI_CONFIG_FAILED	Lower level error while removing the mac address from the multicast list.
SSP_ERR_TIMEOUT	Timeout while waiting for SPI bus access.
SSP_ERR_ABORTED	Error while waiting for SPI bus access.
SSP_ERR_UNSUPPORTED	API not supported ("On-Chip Stack" mode only)

- **ssp_err_t SF_WIFI_WINC1500_StatisticsGet(sf_wifi_ctrl_t * const p_ctrl, sf_wifi_stats_t * const p_wifi_device_stats)**
  Get the interface statistics.
  Implements sf_wifi_api_t::statisticsGet
  This function performs the following tasks:
  - Collect the statistics information of Wi-Fi interface.

  Return values:

SSP_SUCCESS	Successfully get the Statistics information.
SSP_ERR_ASSERTION	Argument NULL is passed.
SSP_ERR_NOT_OPEN	Driver not open.

- **ssp_err_t SF_WIFI_WINC1500_Transmit(sf_wifi_ctrl_t * const p_ctrl, uint8_t * const p_buf, uint32_t length)**

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

Return values:

SSP_SUCCESS	Successfully added the packet in driver transmit queue.
SSP_ERR_ASSERTION	Argument NULL is passed.
SSP_ERR_WIFI_TRANSMIT_FAILED	Error while sending packets.
SSP_ERR_NOT_OPEN	Driver not open.
SSP_ERR_TIMEOUT	Timeout while waiting for SPI bus access.
SSP_ERR_IN_USE	Error while waiting for SPI bus access.

- **ssp_err_t SF_WIFI_WINC1500_ProvisioningSet(sf_wifi_ctrl_t * const p_ctrl, sf_wifi_provisioning_t const * const p_wifi_provisioning)**
Provisions the Wi-Fi module.
Implements sf_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:

SSP_SUCCESS	Successfully provisioned the driver.
SSP_ERR_NOT_OPEN	Driver not open.
SSP_ERR_ASSERTION	Argument NULL is passed.
SSP_ERR_WIFI_FAILED	Error performing provision settings.
SSP_ERR_WIFI_INVALID_MODE	Invalid provisioning mode.
SSP_ERR_INVALID_ARGUMENT	Invalid parameter/argument is passed.
SSP_ERR_INVALID_HW_CONDITION	Error while sending provisioning request.
SSP_ERR_TIMEOUT	Timeout while waiting for SPI bus access.
SSP_ERR_ABORTED	Error while waiting for SPI bus access.

- **ssp_err_t SF_WIFI_WINC1500_ProvisioningGet(sf_wifi_ctrl_t * const p_ctrl, sf_wifi_provisioning_t * const p_wifi_provisioning)**
Reads the current Wi-Fi Provisioning information of the Wi-Fi module.
Implements sf_wifi_api_t::provisioningGet
This function performs the following tasks:
- Reads the provisioning information

Return values:

SSP_SUCCESS	Successfully reads provisioning information.

SSP_ERR_NOT_OPEN	Driver not open.
SSP_ERR_ASSERTION	Argument NULL has been passed.

- **ssp_err_t SF_WIFI_WINC1500_InfoGet(sf_wifi_ctrl_t * const p_ctrl, sf_wifi_info_t * const p_wifi_info)**
  Get Wi-Fi module information.
  Implements sf_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:

SSP_SUCCESS	Successfully read information.
SSP_ERR_NOT_OPEN	Driver not opened.
SSP_ERR_ASSERTION	Argument NULL is passed.
SSP_ERR_INVALID_HW_CONDITION	Error while sending information requests.
SSP_ERR_WIFI_FAILED	Timeout or error while waiting for API result.
SSP_ERR_IN_USE	Another thread is using this API and timeout for the access occurred.
SSP_ERR_TIMEOUT	Timeout while waiting for SPI bus access.
SSP_ERR_ABORTED	Error while waiting for SPI bus access.

- **ssp_err_t SF_WIFI_WINC1500_Scan(sf_wifi_ctrl_t * const p_ctrl, sf_wifi_scan_t * const p_scan, uint8_t * const p_cnt)**
  Scans for available APs.
  Implements sf_wifi_api_t::scan
  This function performs the following tasks:
  - Scans for available AP's SSID and return the list to caller.

  Return values:

SSP_SUCCESS	Successfully scan the network for available APs.
SSP_ERR_NOT_OPEN	Driver not opened.
SSP_ERR_ASSERTION	Argument NULL is passed.
SSP_ERR_WIFI_FAILED	Error in Scanning.
SSP_ERR_INVALID_MODE	Scan not supported in AP mode.
SSP_ERR_INVALID_HW_CONDITION	Error while sending information requests.
SSP_ERR_WIFI_FAILED	Timeout or error while waiting for API result.
SSP_ERR_IN_USE	Another thread is using this API and timeout for the access occurred.
SSP_ERR_TIMEOUT	Timeout while waiting for SPI bus access.

SSP_ERR_ABORTED	Error while waiting for SPI bus access.

- **ssp_err_t SF_WIFI_WINC1500_AccessControlListAdd(sf_wifi_ctrl_t * const p_ctrl, uint8_t const * const p_mac)**
  Add MAC address from Access control list.
  Implements sf_wifi_api_t::accessControlListAdd
  This function performs the following tasks:
  - Adds specified MAC address in access control list.

  Return value:

SSP_SUCCESS	Successfully added the mac address in access control list.
SSP_ERR_NOT_OPEN	Driver not opened.
SSP_ERR_ASSERTION	Argument NULL is passed.
SSP_ERR_UNSUPPORTED	Operation is not supported.

- **ssp_err_t SF_WIFI_WINC1500_AccessControlListDelete(sf_wifi_ctrl_t * const p_ctrl, uint8_t const * const p_mac)**
  Delete MAC address from Access control list.
  Implements sf_wifi_api_t::accessControlListDelete
  This function performs the following tasks:
  - Deletes specified MAC address in access control list.

  Return value:

SSP_SUCCESS	Successfully deleted the mac address from access control list.
SSP_ERR_NOT_OPEN	Driver not opened.
SSP_ERR_UNSUPPORTED	Operation is not supported.
SSP_ERR_ASSERTION	Argument NULL is passed.

- **ssp_err_t SF_WIFI_WINC1500_MACAddressGet(sf_wifi_ctrl_t * const p_ctrl, uint8_t * const p_mac)**
  Get MAC address of Wi-Fi module.
  Implements sf_wifi_api_t::getMACAddress
  This function performs the following tasks:
  - Reads configured MAC address of the Wi-Fi module.

  Return values:

SSP_SUCCESS	Successfully reads the mac address.
SSP_ERR_NOT_OPEN	Driver not open.
SSP_ERR_ASSERTION	Argument NULL is passed.

- **ssp_err_t SF_WIFI_WINC1500_MACAddressSet(sf_wifi_ctrl_t * const p_ctrl, uint8_t const * const p_mac)**
  Set MAC address of Wi-Fi module.
  Implements sf_wifi_api_t::setMACAddress
  This function performs the following tasks:
  - Configures MAC address of the Wi-Fi module.

  Return value:

SSP_SUCCESS	Successfully deleted the mac address from access control list.
SSP_ERR_UNSUPPORTED	Functionality is not supported.
SSP_ERR_NOT_OPEN	Driver not opened.
SSP_ERR_INVALID_HW_CONDITION	Error while sending information requests.
SSP_ERR_TIMEOUT	Timeout while waiting for SPI bus access.
SSP_ERR_ABORTED	Error while waiting for SPI bus access.

- **ssp_err_t SF_WIFI_WINC1500_VersionGet(ssp_version_t * const p_version)**
  Set driver version based on compile time macros.
  Implements sf_wifi_api_t::versionGet.
  This function performs the following tasks:
  - Returns driver version.

  Return value:

SSP_SUCCESS	Success.
SSP_ERR_ASSERTION	Argument NULL is passed.

2.2 Wi-Fi On Chip Networking Stack on ATWINC1500

2.2.1 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 Framework Module APIs are summarized in the following section.

2.2.2 Interface APIs

The Wi-Fi On Chip Networking Stack interface APIs are defined in `sf_wifi_onchip_stack_api_t` structure. Additional details are reported in both [1], [2] and [3].

2.2.2.1 open

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

2.2.2.2 close

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

2.2.2.3 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.

2.2.2.4 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.

2.2.2.5 dhcpServerStop

This API stops the DHCP server.

2.2.3 Data structures

- `sf_wifi_onchip_stack_ip_cfg_t`
- `sf_wifi_onchip_stack_cfg_t`
- `sf_wifi_onchip_stack_ctrl_t`
- `sf_wifi_onchip_stack_instance`

2.2.4 Enumerations

- `sf_wifi_onchip_stack_ip_addr_mode_t`

2.2.5 Defines

- `#define SF_WIFI_ONCHIP_STACK_API_VER_MAJOR (1U)`

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

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

2.2.6 API Data

Synergy standardized WiFi Framework API Data are detailed in [2] and [3].

2.2.7 API Structures

Synergy standardized WiFi Framework API Structures are detailed in [2] and [3].

2.3 SF Socket WIFI Framework Interface

RTOS-integrated SF Socket WIFI Framework Interface.

2.3.1 Summary

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

2.3.2 Functions

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

2.3.3 Interface API

The SF Socket Wi-Fi Framework interface APIs are defined in `sf_socket_api_t` structure. Additional details are reported in [1], [2] and [3].

2.3.3.1 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.

2.3.3.2 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.

2.3.3.3 versionGet

Gets version and stores it in provided pointer p_version.

2.3.4 Data structures

- in_addr
- sockaddr
- sockaddr_in
- sf_socket_ctrl_t
- sf_socket_cfg_t
- sf_socket_instance_t

2.3.5 Typedefs

- socklen_t

2.3.6 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.

2.3.7 Functions Data

Socket WiFi Framework Functions Data are detailed in [2] and [3].

2.3.8 API Data

Synergy standardized WiFi Framework API Data are detailed in [2] and [3].

2.3.9 API Structures

Synergy standardized WiFi Framework API Structures are detailed in [2] and [3].

3.Setting examples

3.1 ATWINC1500 on PK-S5D9

This section describes example of connections and configuration details used by ATWINC1500 WiFi module on Renesas Synergy™ Promotion Kit PK-S5D9.

The Renesas Synergy™ Promotion Kit PK-S5D9 is a low-cost way to access the entire Synergy Platform, enabling full development using the vast majority of all Synergy Software Package (SSP) functions.

Figure 2 shows the orientation of the ATWINC1500 module adapter board when connected to PMODA on the PK-S5D9 Rev.1.0.

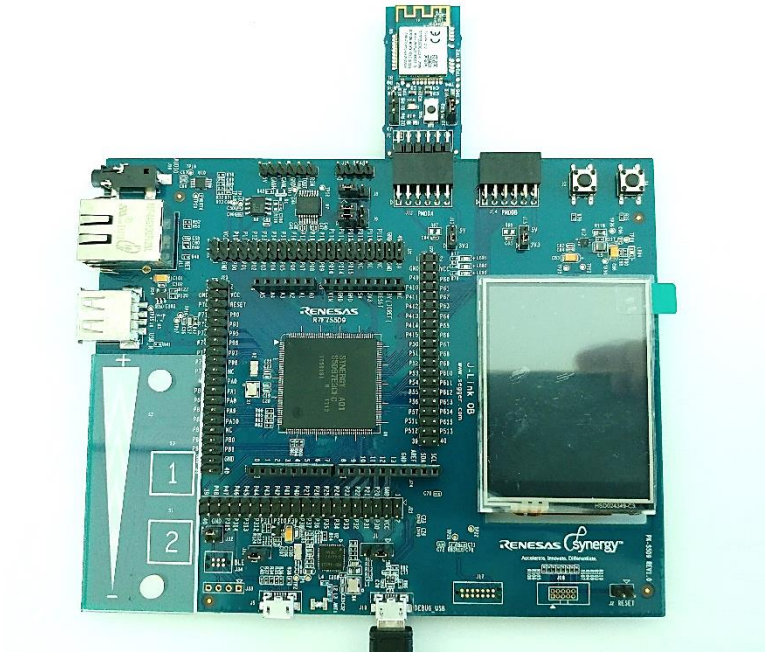

Figure 2: PK-D5D9 Board with WINC15X0 module connected.

Make sure J13 is on 3V3 position for PMODA.

3.1.1 Thread panel setup

Create a new thread from the thread panel and add a NetX IP Instance as shown in Figure 3.

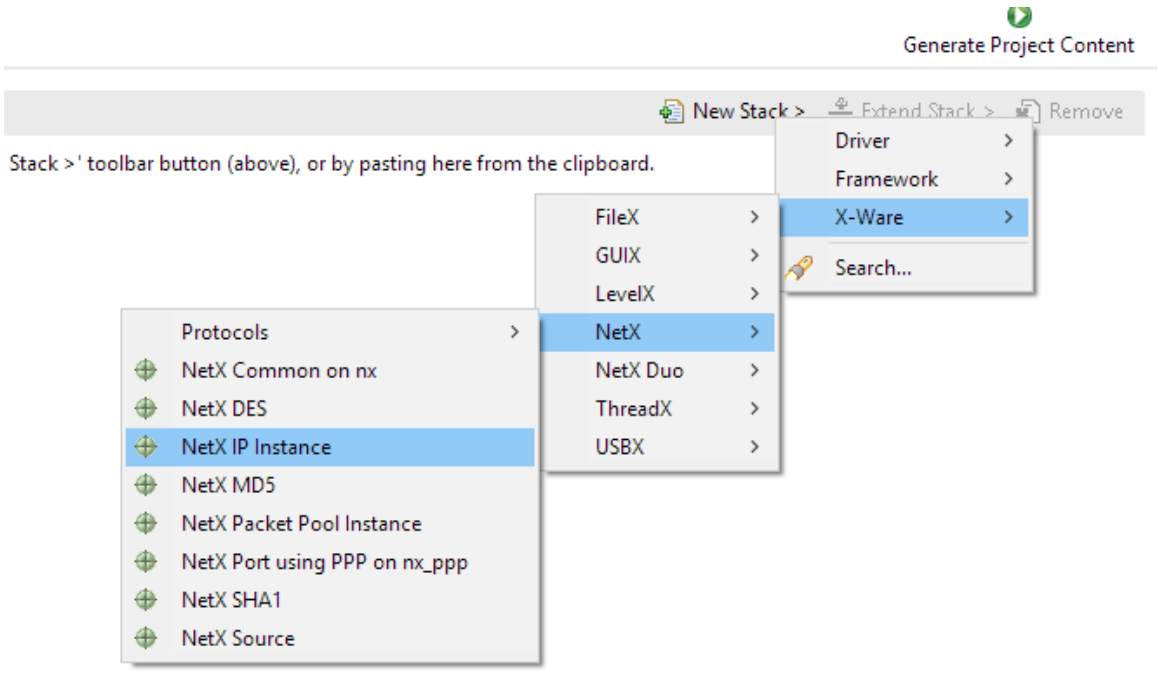

Figure 3: Add a NetX IP Instance.

The SSP configurator will build the NetX IP instance as far as it can. However, there is still some configuration to be done and a pink base, for the boxes where an action is needed, highlights this. If in a box with pink base there is the key [Optional], the required action is possible but not mandatory.

Figure 4 shows what the SSP configurator has been able to build.

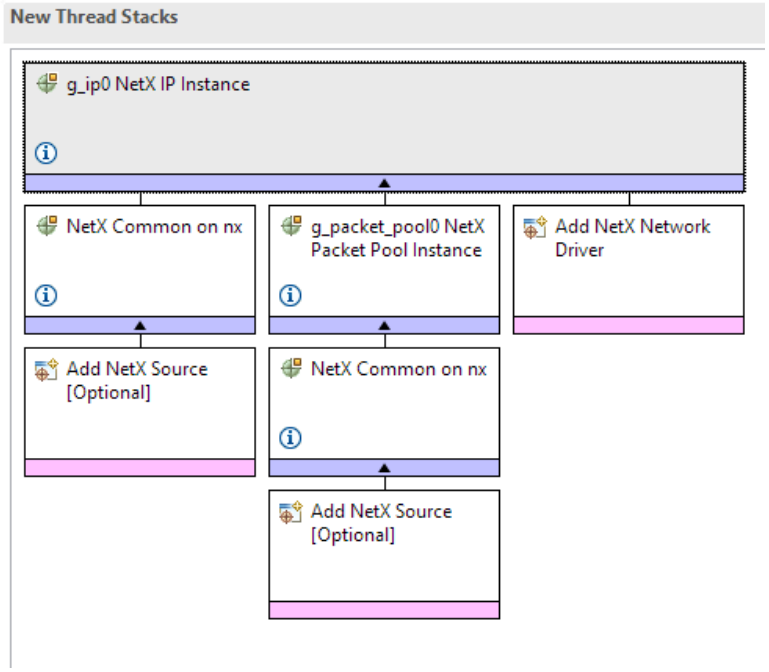

Figure 4: NetX IP Instance hierarchy.

The only mandatory user's action required by the SSP configurator is to add a NetX Network Driver, which can be chosen between the different network interfaces available. Left click on the box and choose the one over the Wi-Fi Framework.

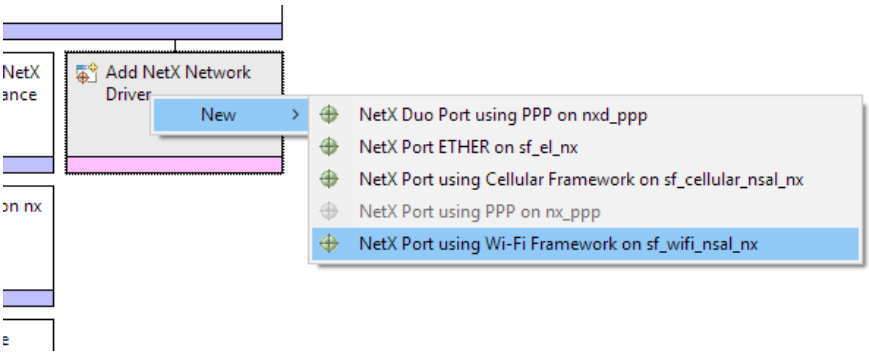

Figure 5: NetX Network Driver addition.

If there are more than one Wi-Fi Framework Device Drivers, the SSP configurator will add a new box with the pink base to let the user choose which one, of the installed device driver, to use for the NetX IP Instance.

Left click on the box and choose ***New > ATWINC1500 Wi-Fi Device Driver on sf_wifi_winc1500***.

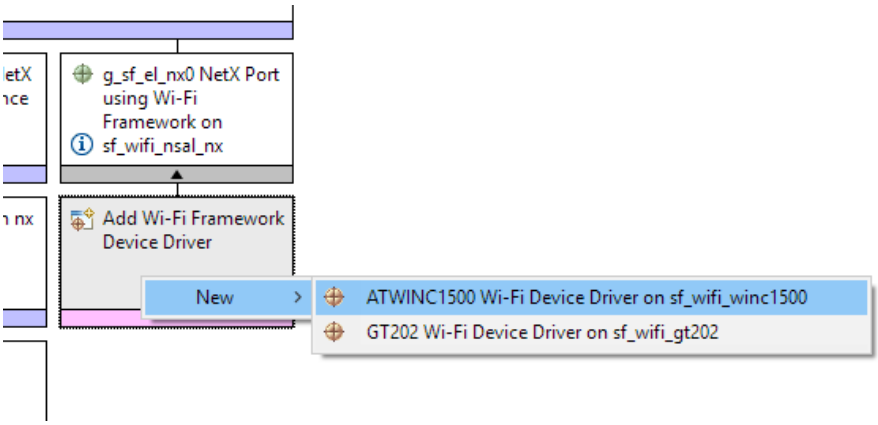

Figure 6: Wi-Fi Framework Device Driver selection.

The device driver communicates with the module through an SPI Interface; hence, the SSP configurator automatically adds the framework for the SPI interface, which will show another box with pink base to select whether to use the driver for the *RSPI* or the *SCI.* The PMODA connector of the PK-S5D9 exposes one of the S5D9 *SCI*.

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

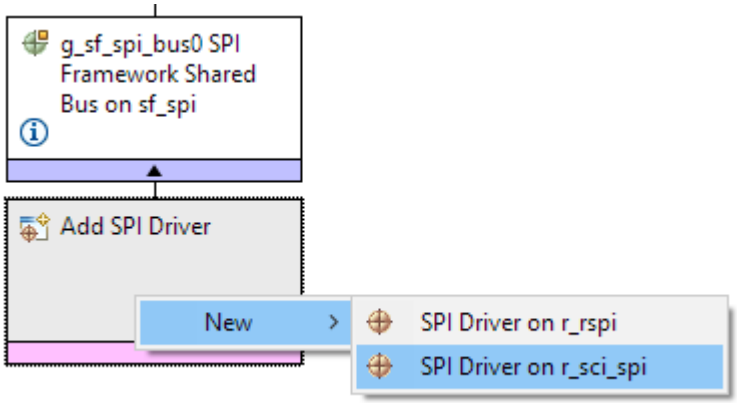

Figure 7: SPI Driver selection.

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

Figure 8 shows the complete NetX IP instance.

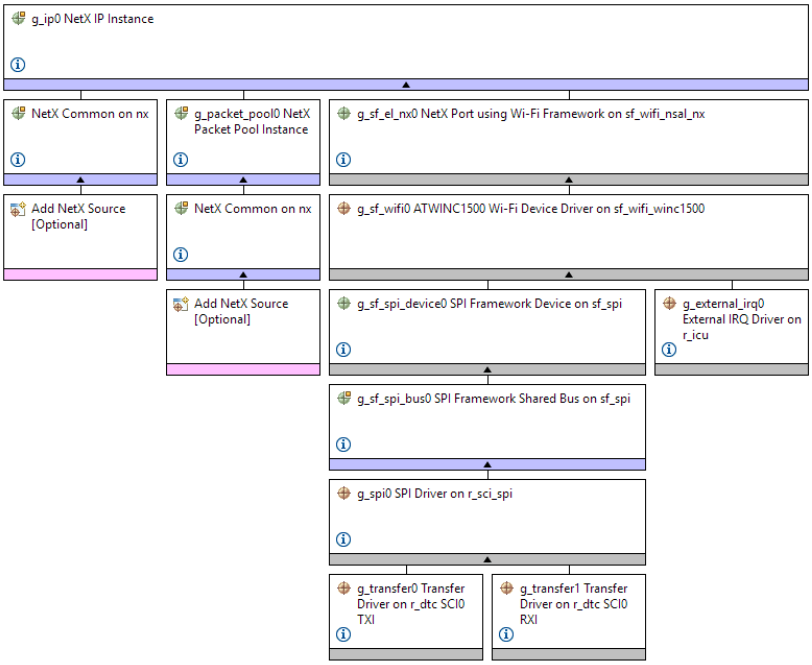

Figure 8: Full NetX IP Instance hierarchy.

3.1.2 SSP Components' configuration

3.1.2.1 g_sf_wifi0 ATWINC1500 Wi-Fi Device Driver on sf_wifi_winc1500

Select the *g_sf_wifi0 ATWINC1500 Wi-Fi Device Driver on sf_wifi_winc1500* block and change its settings to reflect the following:

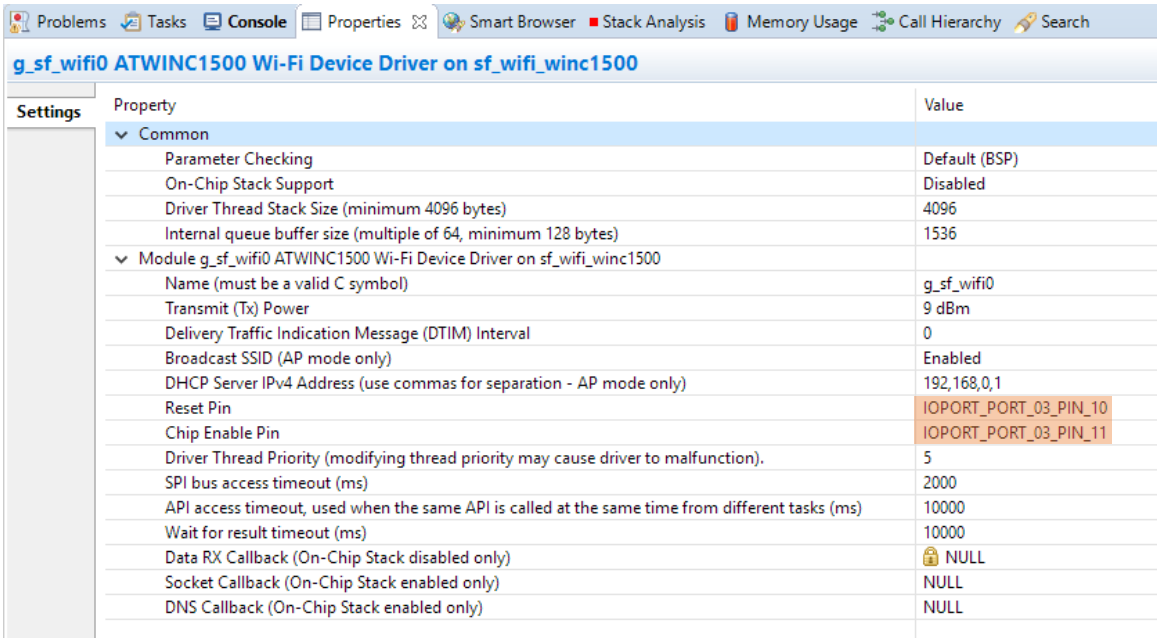

Figure 9: ATWINC1500 Wi-Fi Device Driver on sf_wifi_winc1500 configuration.

3.1.2.2 g_sf_spi_device0 SPI Framework Device on sf_spi

Select the *g_sf_spi_device0 SPI Framework Device on sf_spi* block and change its settings to reflect the following:

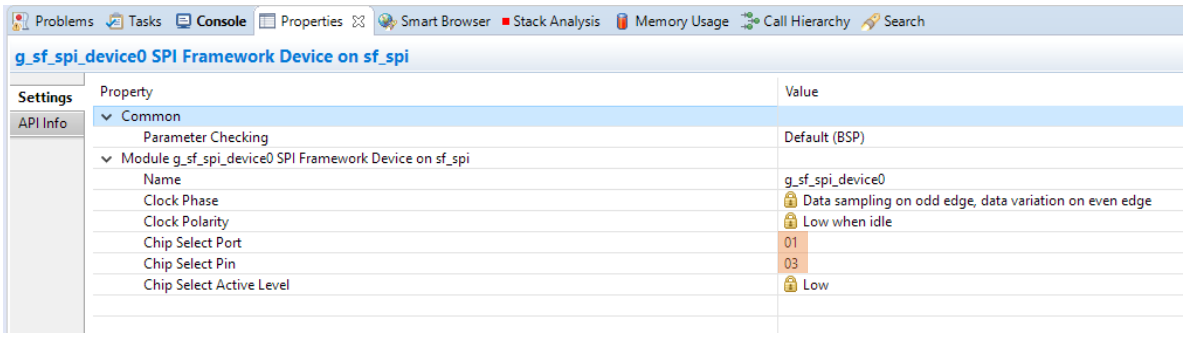

Figure 10: SPI Framework Device on sf_spi configuration.

3.1.2.3 g_spi0 SPI Driver on r_sci_spi

Select the g_spi0 SPI Driver on r_sci_spi block and change its settings to reflect the following:

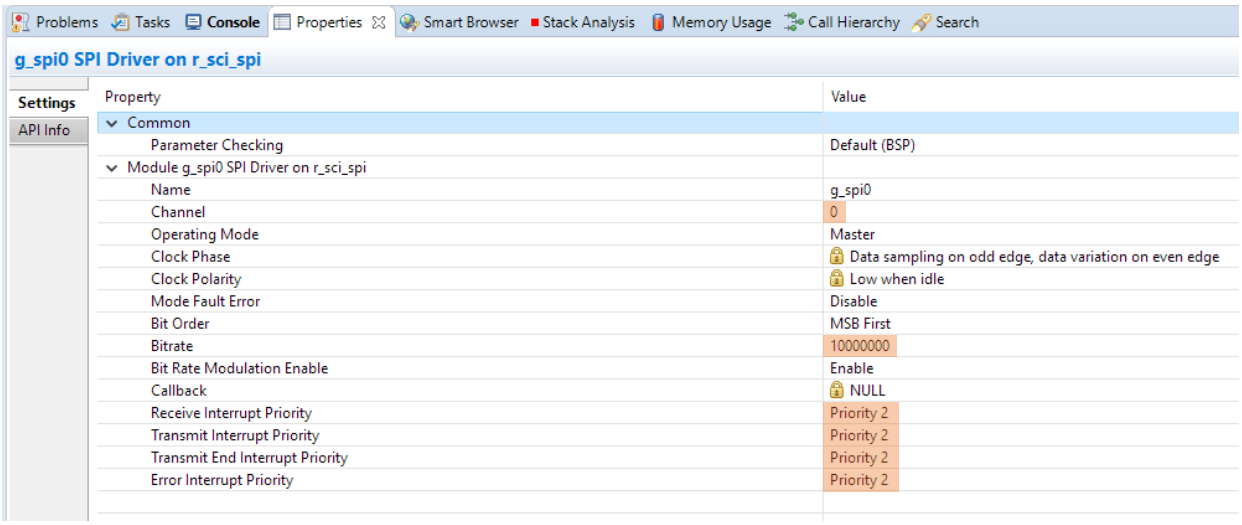

Figure 11: SPI Driver on r_sci_spi configuration.

3.1.2.4 g_external_irq0 External IRQ Driver on r_icu

Select the g_external_irq0 External IRQ Driver on r_icu block and change its settings to reflect the following:

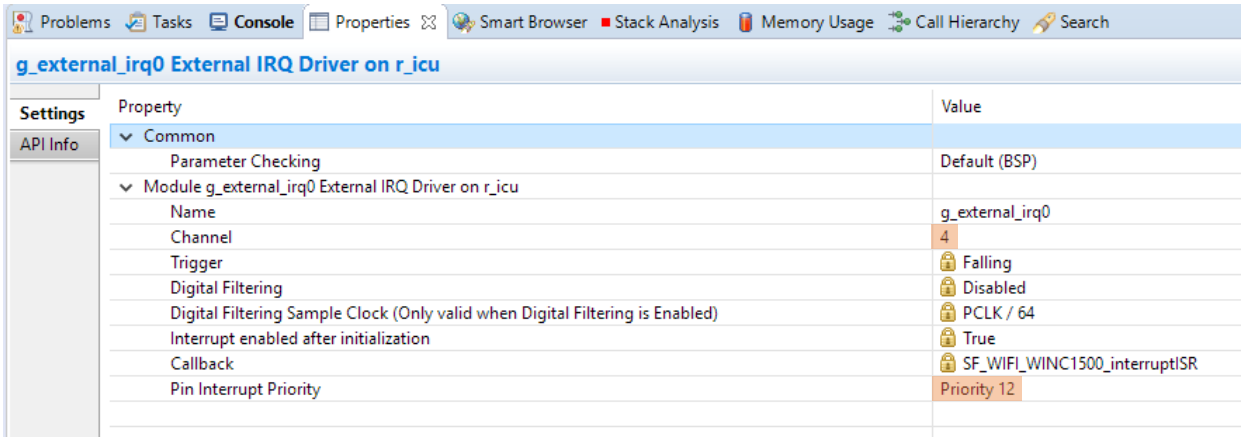

Figure 12: External IRQ Driver on r_icu configuration.

3.1.3 Pins and peripherals configurations

3.1.3.1 SCI0 Configuration

Set up the SCI0 for PK-S5D9

Go to **Pins tab** and from the **Pin Selection** section go to **Peripherals > Connectivity:SCI > SCI0**. Go to **Pin Configuration** section and change the settings to reflect the following:

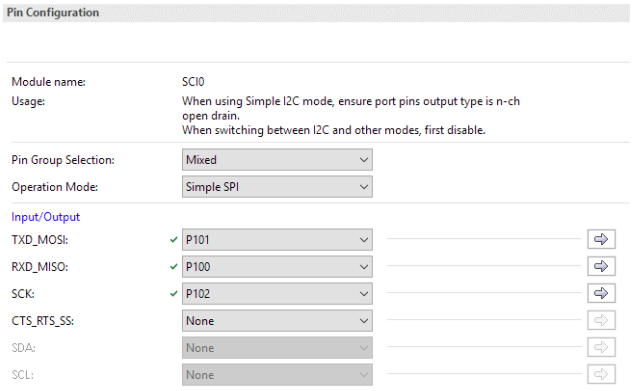

Figure 13: SCI0 Pins configuration.

Set up the SCI0 Chip Select pin P103 for PK-S5D9

Go to **Pins tab** and from the **Pin Selection** section go to **Ports > P1 > P103**. Go to **Pin Configuration** section and change the settings to setup the chip select pin for the SCI:

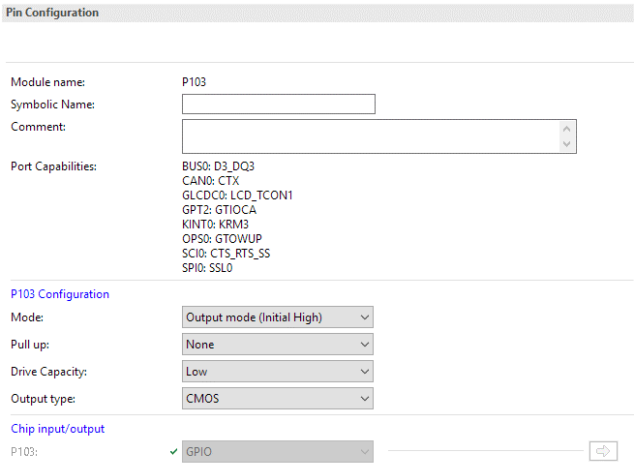

Figure 14: Chip Select pin configuration.

3.1.3.2 IRQ Configuration

Set up the IRQ pin P111 for PK-S5D9. This is IRQ04

Go to **Pins tab** and from the **Pin Selection** section go to **Ports > P1 > P411.** Go to **Pin Configuration** section and change the settings to setup the IRQ for the Wi-Fi module:

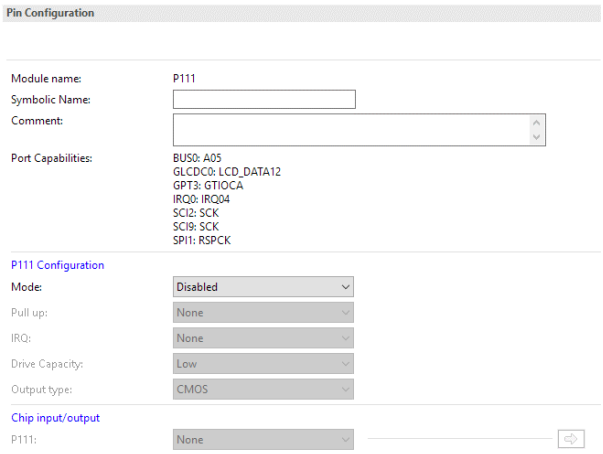

Figure 15: IRQ pin configuration.

Go to **Pins tab** and from the **Pin Selection** section go to **Peripherals > Input:IRQ > IRQ04**. Go to **Pin Configuration** section and change the settings to setup the IRQ for the Wi-Fi module:

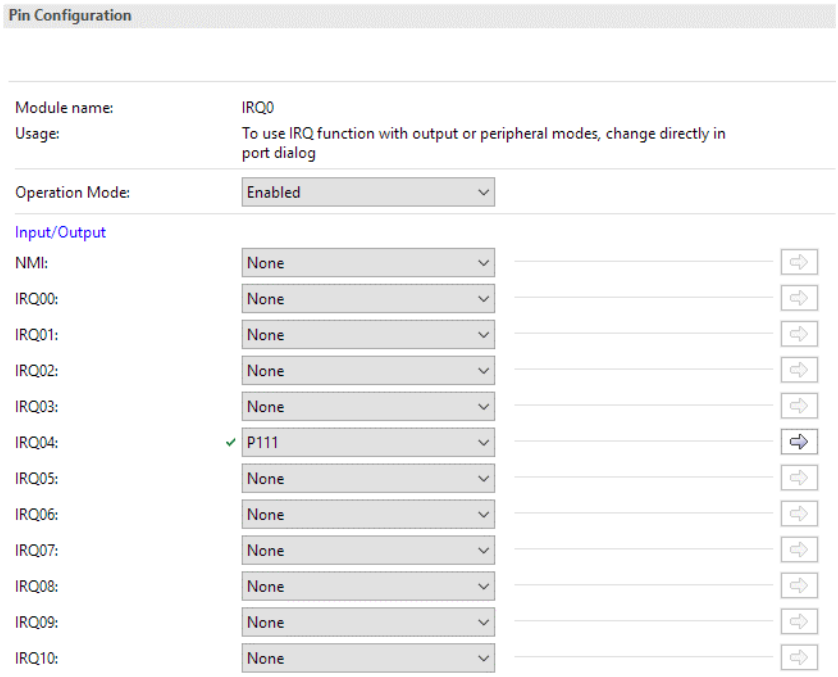

Figure 16: IRQ configuration.

3.1.3.3 ATWINC1500 Reset and CE pins

Set up the Reset Pin pin P310 for PK-S5D9. This is the Reset pin for the module

Go to **Pins tab** and from the **Pin Selection** section go to **Ports > P3 > P310**. Go to **Pin Configuration** section and change the settings to setup the Reset Pin for the Wi-Fi module:

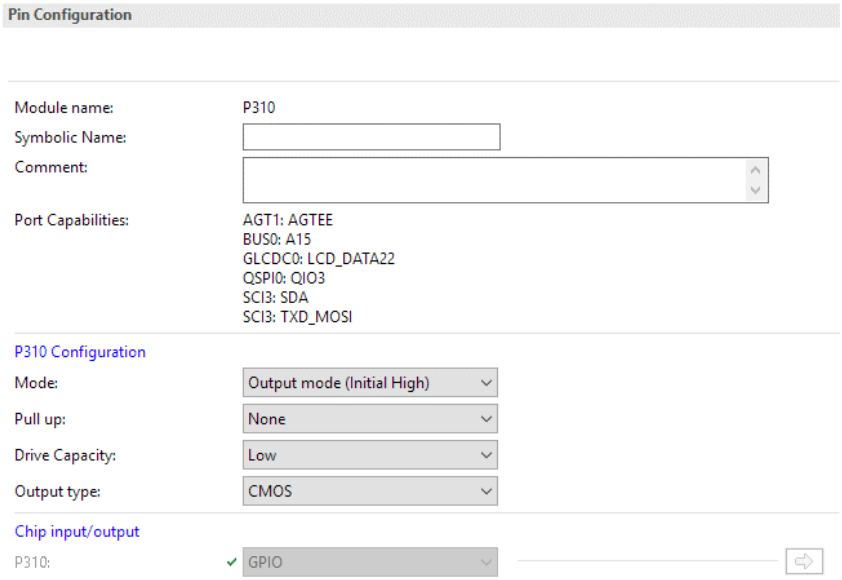

Figure 17: Reset pin configuration.

Set up the Chip Enable pin P311 for PK-S5D9. This is the Chip Enable pin for the module.

Go to **Pins tab** and from the **Pin Selection** section go to **Ports > P3 > P311**. Go to **Pin Configuration** section and change the settings to setup the Chip Enable Pin for the Wi-Fi module:

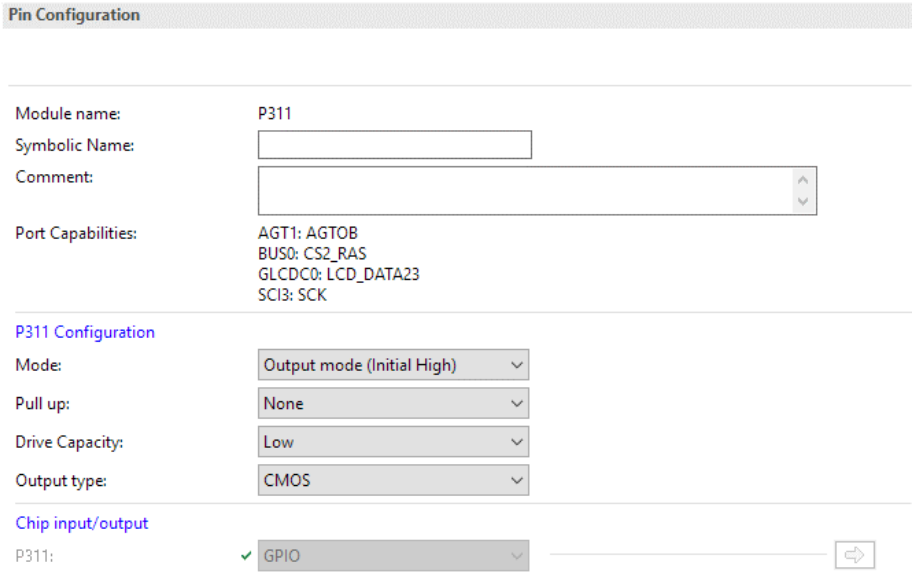

Figure 18: Chip Enable pin configuration.