ATWINC15X0
Wi-Fi Add-on Component
User's Manual

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

Disclaimer

RELOC s.r.l.
HEADQUARTERS
Via Borsari, 23/A 43126 – Parma (Italy)
info@reloc.it – www.reloc.it
fb www.facebook.com/relocsrl
Land +39-0521-1759942

References

[1] Renesas Synergy™ Wi-Fi Framework – document number: R11UM0050EU0108.

1. Overview

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

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

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.

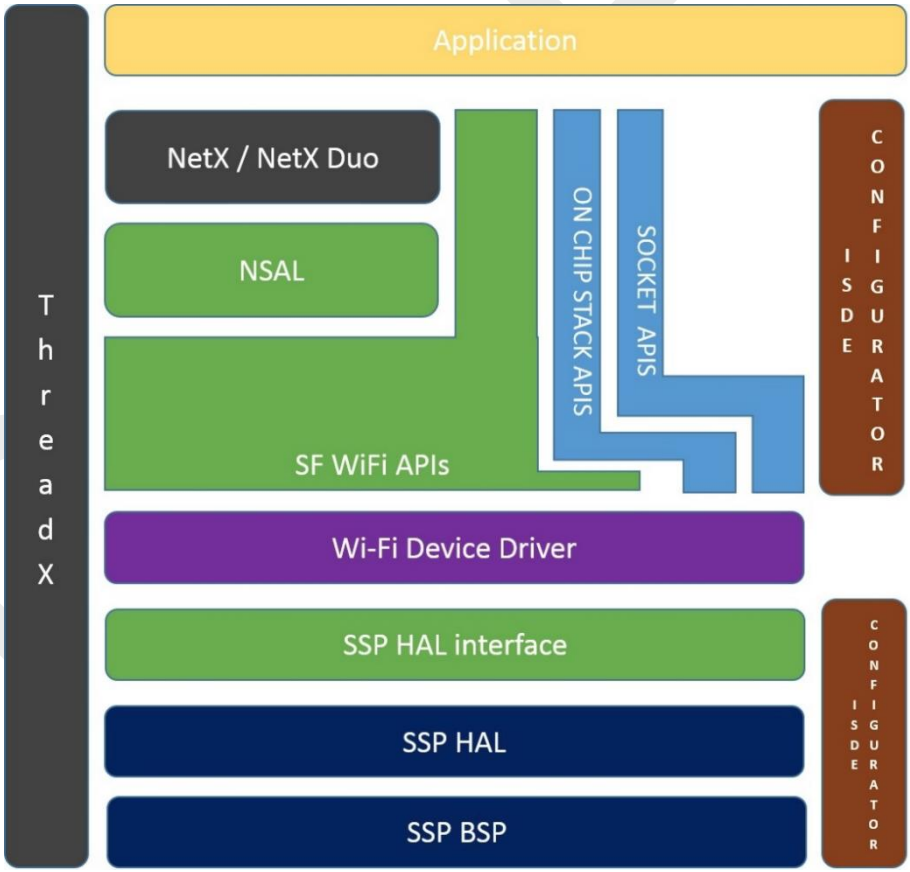

Figure 1: Synergy Wi-Fi Framework layered architecture

2. API References: Wi-Fi Framework on ATWINC1500

2.1 Wi-Fi Framework on ATWINC1500

RTOS-integrated Wi-Fi Framework implementation of ATWINC1500 Driver. It implements the following interfaces:

2.1.1 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 internal_thread_stack_size`: Internal thread stack size;
  - `uint32_t dhcp_address`: address IPv4.

2.1.2 Struct Reference

- `#include <sf_wifi_winc1500.h>`

2.1.3 Detailed Description

2.1.4 Macros

- #define SF_WIFI_WINC1500_CODE_VERSION_MAJOR (2)
- #define SF_WIFI_WINC1500_CODE_VERSION_MINOR (1)

2.1.4.1 Macro Definition Documentation

- **SF_WIFI_WINC1500_CODE_VERSION_MAJOR**
  Wi-Fi Interface. Major version of code that implements the API defined in this file
- **SF_WIFI_WINC1500_CODE_VERSION_MINOR**
  Wi-Fi Interface. Minor version of code that implements the API defined in this file

2.1.5 Functions

- `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. Implements sf_wifi_api_t::versionGet.

2.1.5.1 Function 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 passed or ThreadX resources initialization failed.
SSP_ERR_INVALID_HW_CONDITION	Module initialization failed.
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 close successfully.
SSP_ERR_ASSERTION	Argument NULL passed.
SSP_ERR_NOT_OPEN	Driver already closed.
SSP_ERR_IN_USE	Unable to get access to the SPI bus.
SSP_ERR_INVALID_HW_CONDITION	Error closing Wi-Fi driver.

- **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.

  Return value:

SSP_ERR_UNSUPPORTED	Functionality not supported.

- **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.

  Return value:

SSP_ERR_UNSUPPORTED	Functionality not supported.

- **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 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 passed.
SSP_ERR_WIFI_TRANSMIT_FAILED	Error while sending packets.
SSP_ERR_NOT_OPEN	Driver not open.
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 passed.
SSP_ERR_WIFI_FAILED	Error performing provision settings.
SSP_ERR_WIFI_INVALID_MODE	Invalid provisioning mode.
SSP_ERR_INVALID_ARGUMENT	Invalid parameter/argument passed.
SSP_ERR_INVALID_HW_CONDITION	Error while sending provisioning request.

SSP_ERR_IN_USE	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 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_IN_USE	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_IN_USE	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 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_IN_USE	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 passed.

2.2 Wi-Fi On Chip Stack on ATWINC1500

Function under development.

2.3 BSD Socket on ATWINC1500

Function under development.

3. Setting examples

3.1 ATWINC1500 on SK-S7G2

This section describes example of connections and configuration details used by ATWINC1500 WiFi module on Renesas Synergy™ Starter Kit SK-S7G2.

The Renesas Synergy™ Starter Kit SK-S7G2 is an extremely 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 PMODB on the SK-S7G2 Rev.3.0.

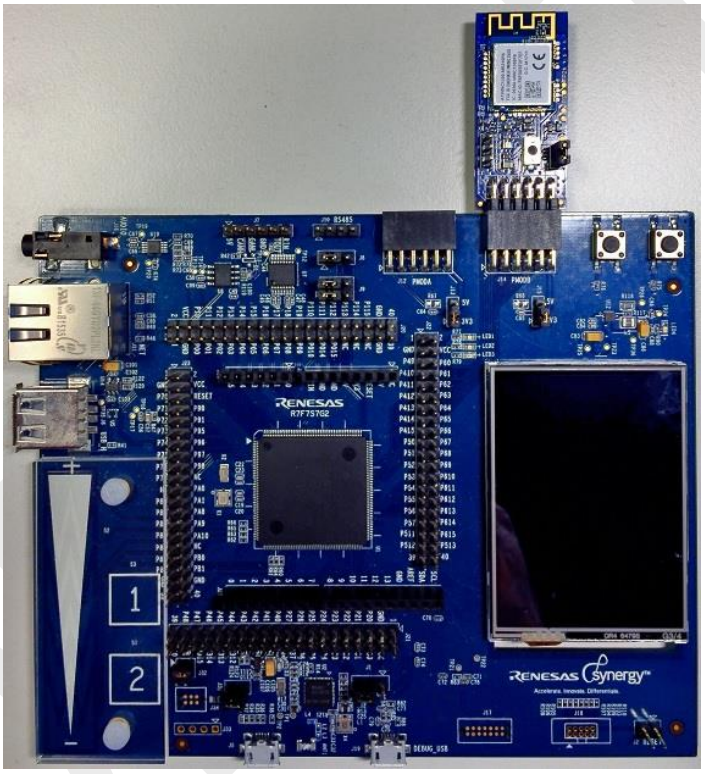

Figure 2: SK-S7G2 Board with WINC15X0 module connected.

Make sure J15 is on 3V3 position for PMODB

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.

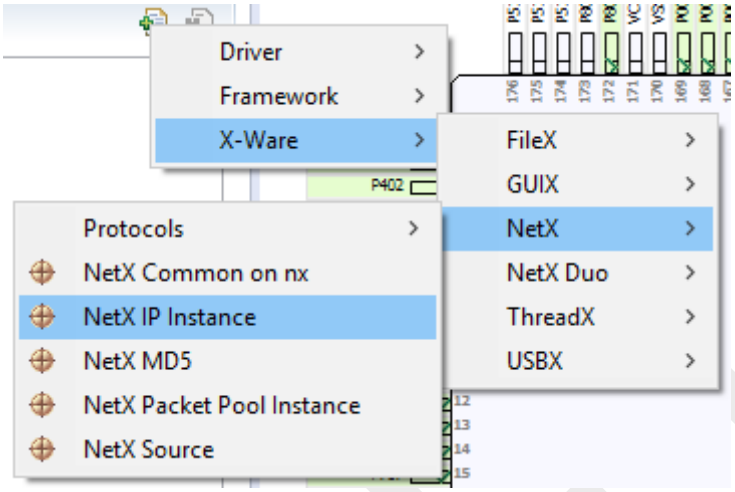

Figure 3: Add a NetX IP Instance.

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 necessary needed.

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

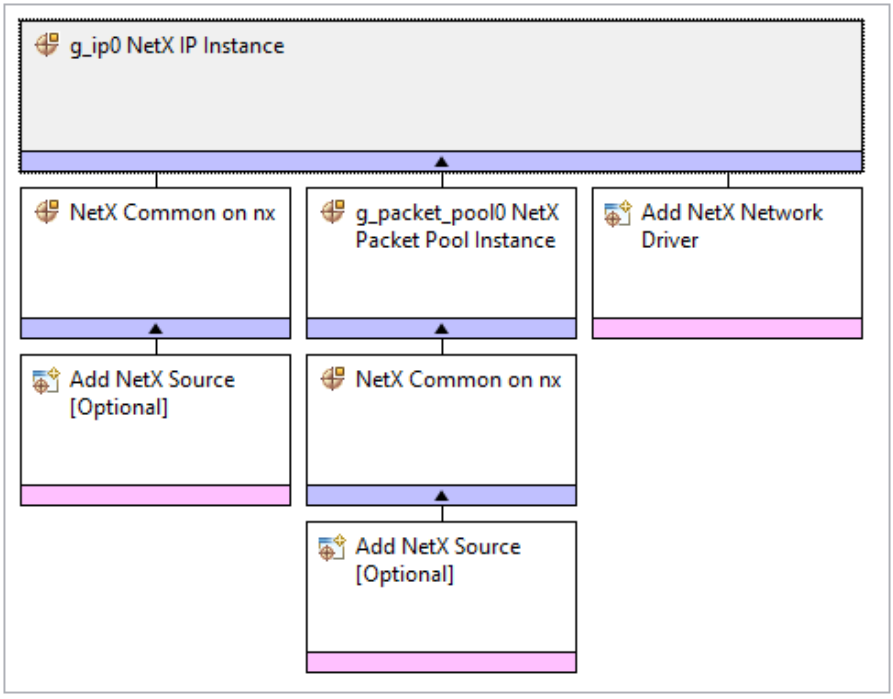

Figure 4: NetX IP Instance hierarchy.

The only mandatory user's action required by the SSP configurator is to add a NetX Network Driver, which can be the driver for the Synergy S7 device Ethernet physical port or the Wi-Fi Framework. Left click on the box and choose the one over the Wi-Fi Framework.

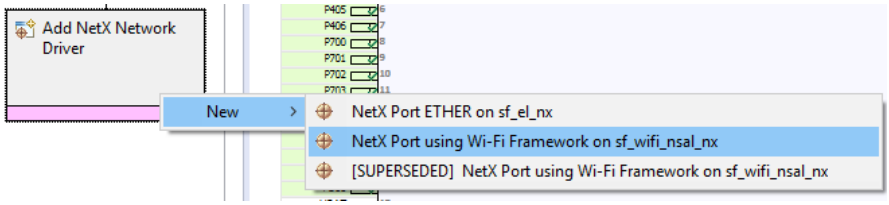

Figure 5: NetX Network Driver addition.

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 PMODB connector of the SK-S7G2 exposes one of the S7 *SCI*.

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

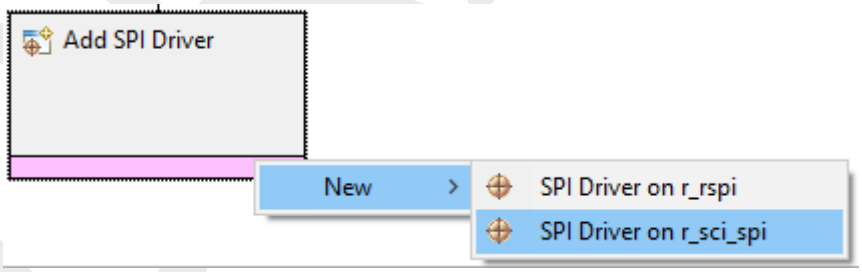

Figure 7: SPI Driver selection.

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.

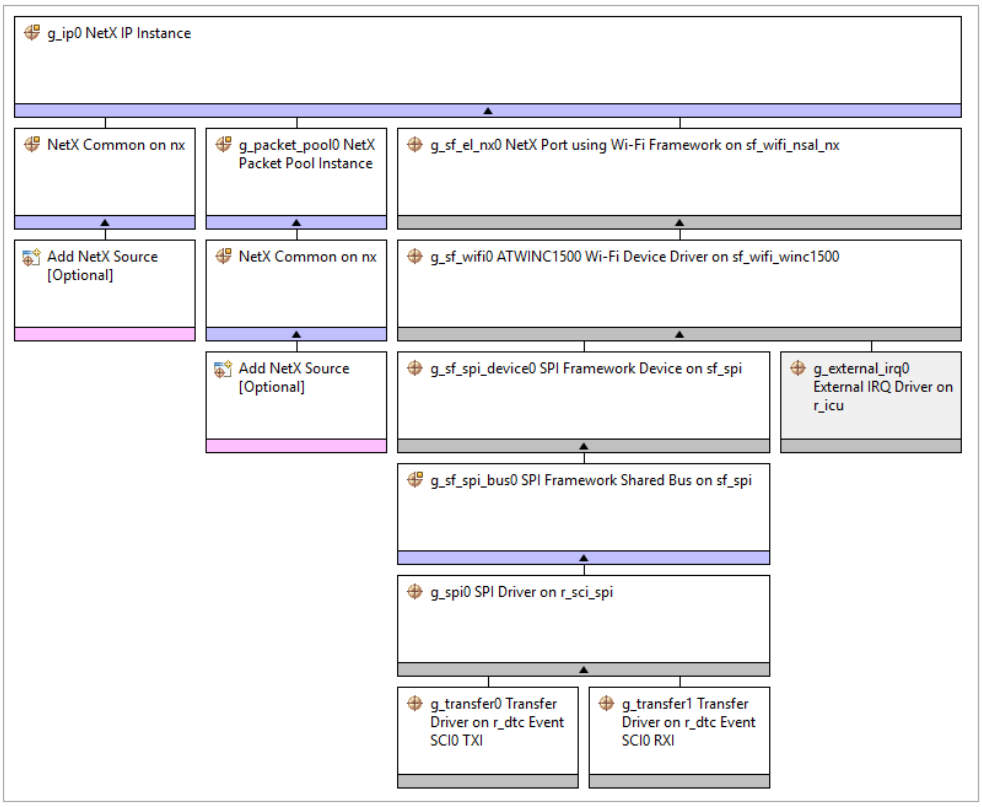

Figure 8: Full NetX IP Instance hierarchy.

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:

Property	Value
Common	
Parameter Checking	Default (BSP)
Module g_sf_wifi0 ATWINC1500 Wi-Fi Device Driver on sf_wifi_winc1500	
Name	g_sf_wifi0
Transmit (Tx) Power	9 dBm
Delivery Traffic Indication Message (DTIM) Interval	0
Broadcast SSID (AP mode only)	Enabled
DHCP IPv4 Address (use commas for separation - AP mode only)	192,168,0,1
Reset Pin	IOPORT_PORT_06_PIN_03
Chip Enable Pin	IOPORT_PORT_06_PIN_04
Driver Task Thread Priority (Modifying Task Thread Priority may cause Driver to malfunction).	5
Driver Task Thread Stack Size (Modifying Task Thread Priority may cause Driver to malfunction).	4096
Callback	NULL

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:

Property	Value
Common	
Parameter Checking	Default (BSP)
Module g_sf_spi_device0 SPI Framework Device on sf_spi	
Name	g_sf_spi_device0
Clock Phase	Data sampling on odd edge, data variation on even edge
Clock Polarity	Low when idle
Chip Select Port	04
Chip Select Pin	13
Chip Select Active Level	Low

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:

Property	Value
Common	
Parameter Checking	Default (BSP)
Module g_spi0 SPI Driver on r_sci_spi	
Name	g_spi0
Channel	0
Operating Mode	Master
Clock Phase	Data sampling on odd edge, data variation on even edge
Clock Polarity	Low when idle
Mode Fault Error	Disable
Bit Order	MSB First
Bitrate	10000000
Bit Rate Modulation Enable	Enable
Callback	NULL
Receive Interrupt Priority	Priority 2
Transmit Interrupt Priority	Priority 2
Transmit End Interrupt Priority	Priority 2
Error Interrupt Priority	Priority 2

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 IRW Driver on r_icu configuration.

Property	Value
Common	
Parameter Checking	Default (BSP)
Module g_external_irq0 External IRQ Driver on r_icu	
Name	g_external_irq0
Channel	0
Trigger	Falling
Digital Filtering	Disabled
Digital Filtering Sample Clock (Only valid when Digital Filtering is Enabled)	PCLK / 64
Interrupt enabled after initialization	True
Callback	SF_WINC1500_interruptISR
Interrupt Priority	Priority 7 (CM4: valid, CM0+: invalid)

3.1.3 Pins and peripherals configurations

3.1.3.1 SCI0 Configuration

Set up the SCI0 for SK-S7G2

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:

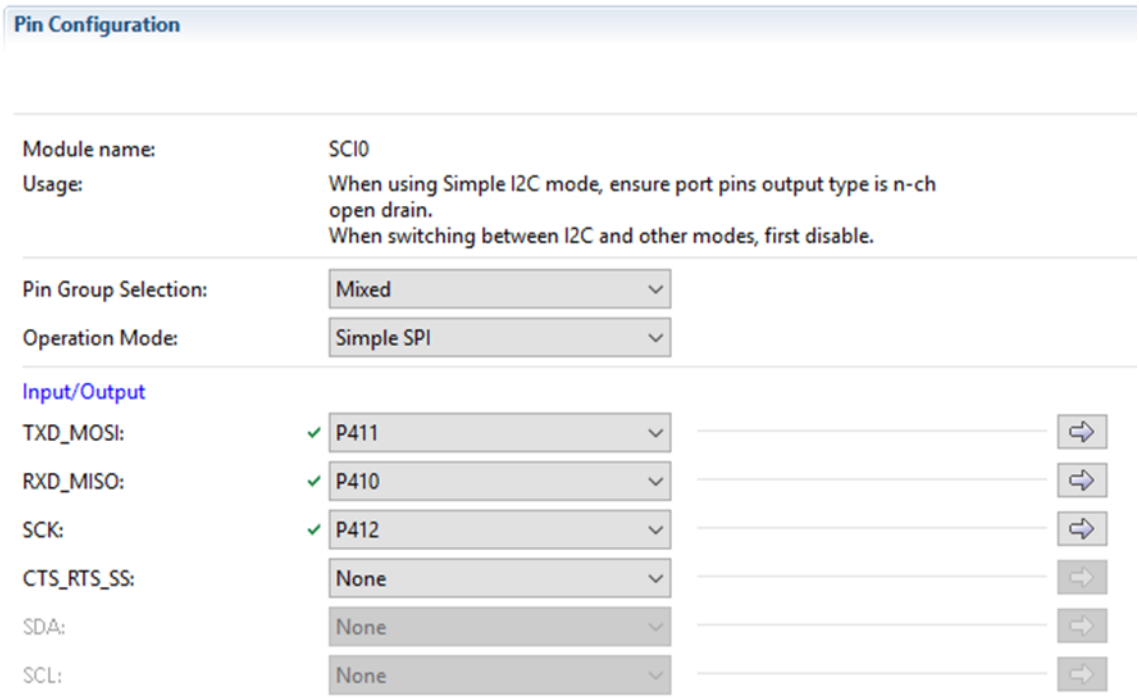

Figure 13: SCI0 Pins configuration.

Set up the SCI0 Chip Select pin P413 for SK-S7G2

Go to **Pins tab** and from the **Pin Selection** section go to **Ports > P4 > P413**. Go to **Pin Configuration** section and change the settings to setup the chip select pin for the SCI:

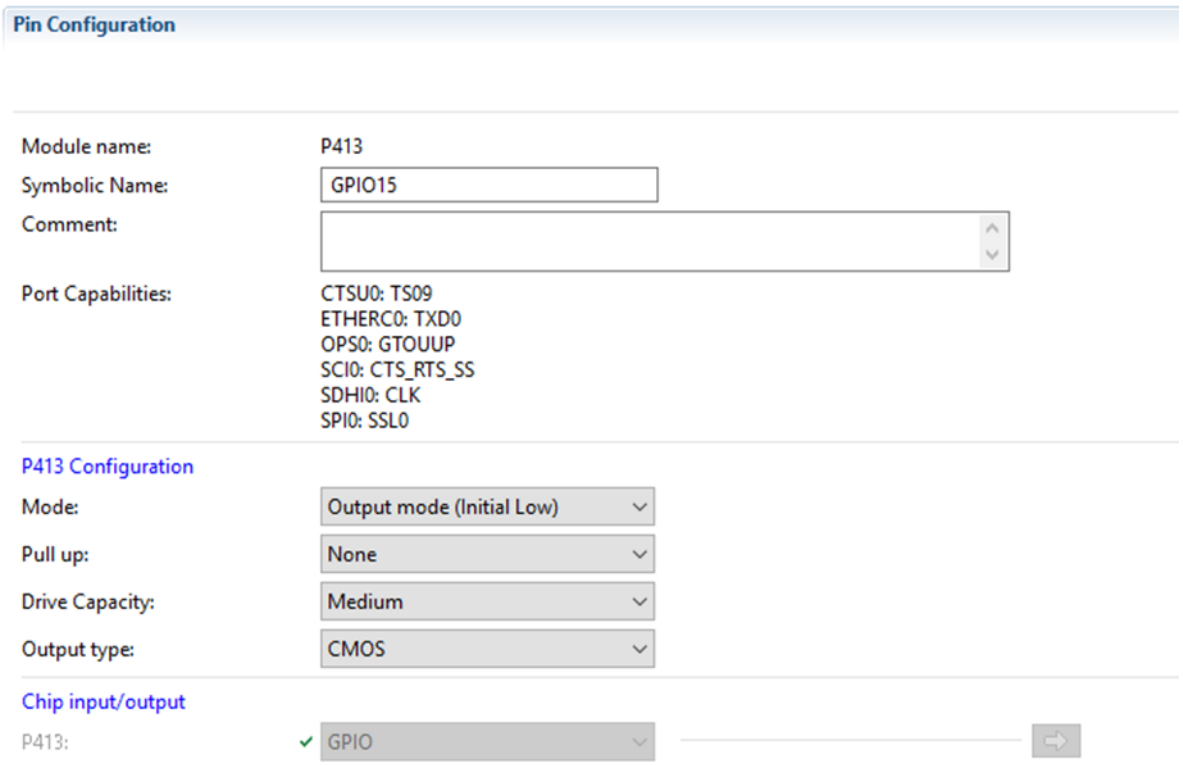

Figure 14: Chip Select pin configuration.

3.1.3.2 IRQ Configuration

Set up the IRQ pin P400 for SK-S7G2. This is IRQ00

Go to **Pins tab** and from the **Pin Selection** section go to **Ports > P4 > P400**. Go to **Pin Configuration** section and change the settings to setup the IRQ for the Wi-Fi module:

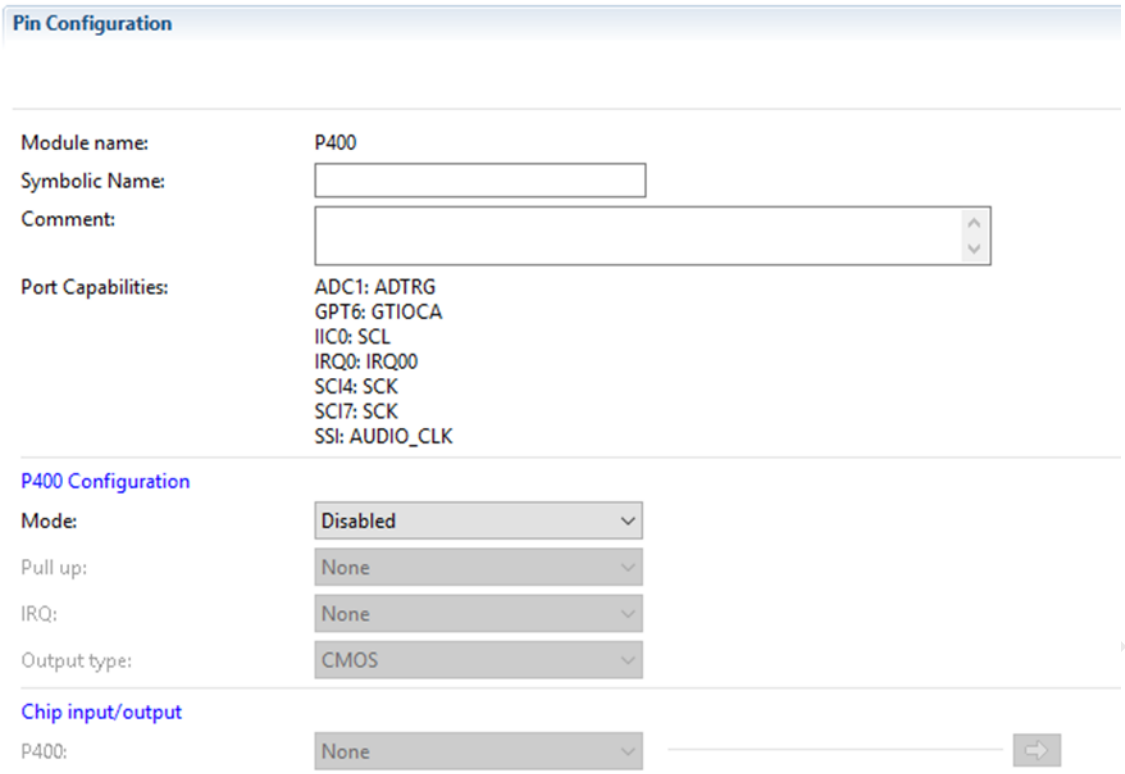

Figure 15: IRQ pin configuration.

Go to **Pins tab** and from the **Pin Selection** section go to **Peripherals > Input:IRQ > IRQ0**. Go to **Pin Configuration** section and change the settings to setup the IRQ for the Wi-Fi module:

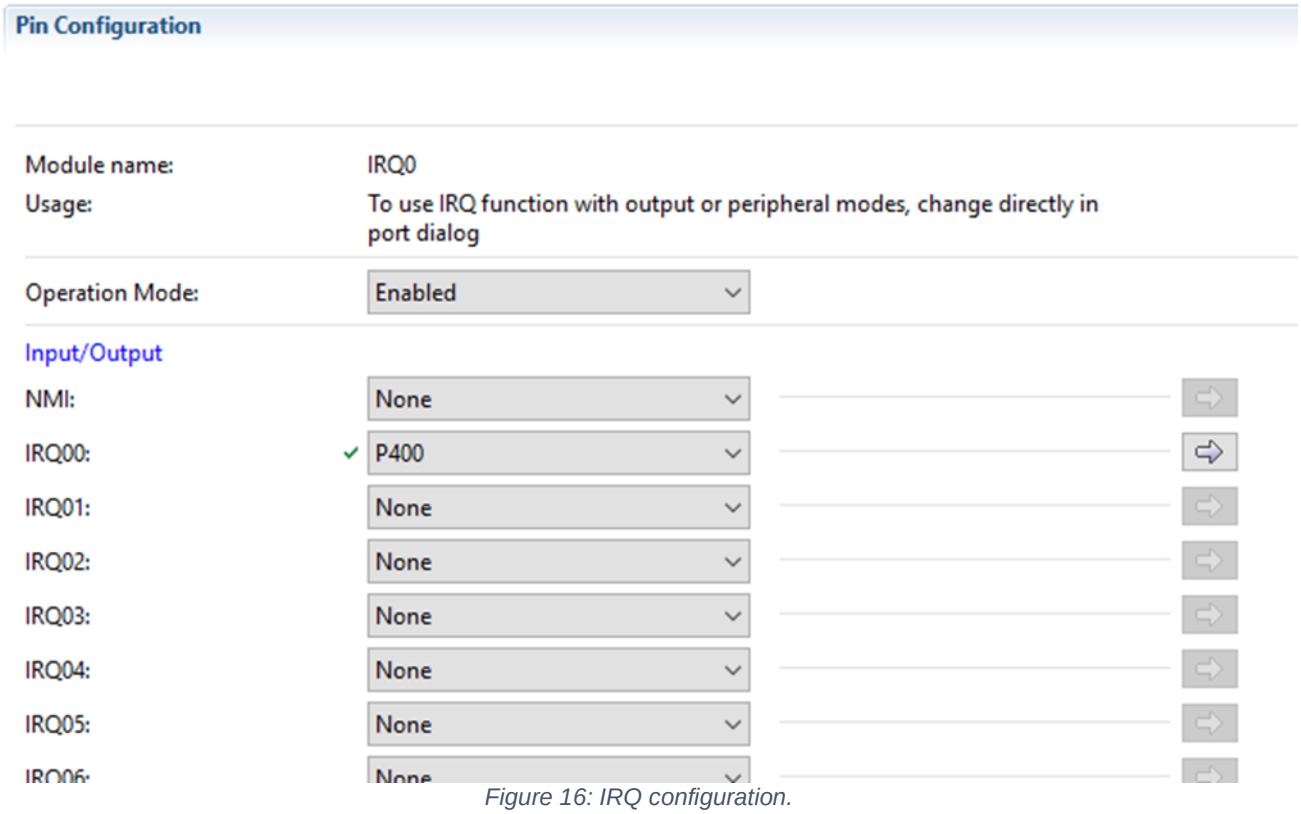

Figure 16: IRQ configuration.

3.1.3.3 ATWINC1500 Reset and CE pins

Set up the Reset Pin pin P603 for SK-S7G2. This is the Reset pin for the module

Go to **Pins tab** and from the **Pin Selection** section go to **Ports > P6 > P603**. Go to **Pin Configuration** section and change the settings to setup the Reset Pin for the Wi-Fi module:

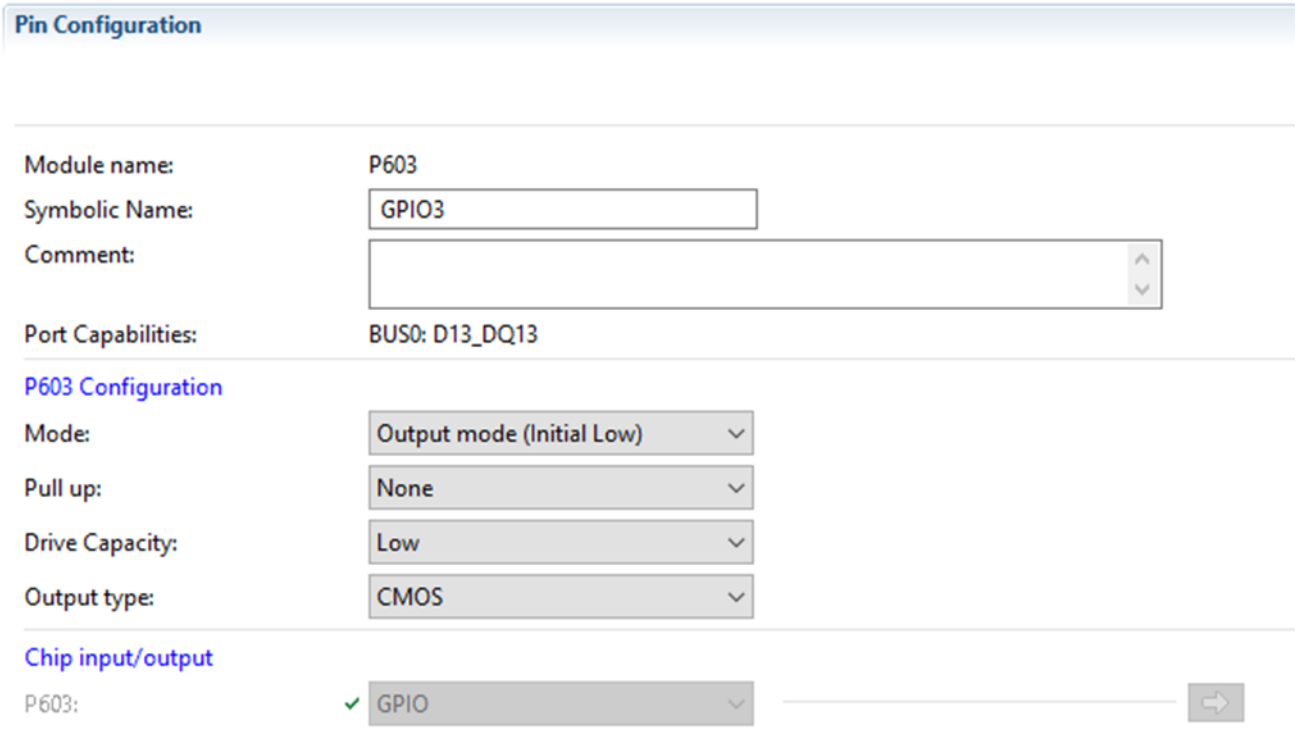

Figure 17: Reset pin configuration.

Set up the Chip Enable pin P604 for SK-S7G2. This is the Chip Enable pin for the module.

Go to **Pins tab** and from the **Pin Selection** section go to **Ports > P6 > P604**. Go to **Pin Configuration** section and change the settings to setup the Chip Enable Pin for the Wi-Fi module:

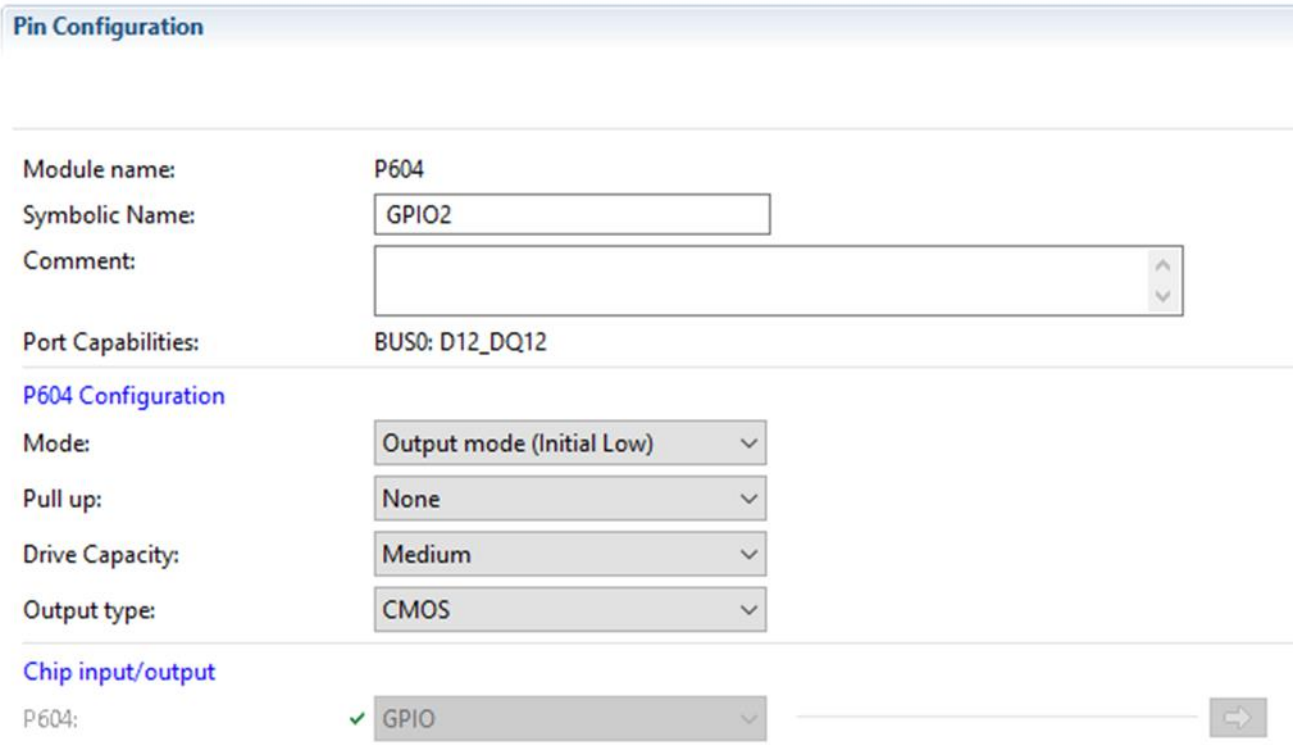

Figure 18: Chip Enable pin configuration.

3.1.3.4 Heap configuration

The Synergy default heap must be increased in order to be able for the WINC1500 driver to allocate the internal stack (4096 bytes, as seen in Figure 9). A minimum of 0x2000 is recommended.

Go to **BSP tab** and look for "Heap size" in the properties:

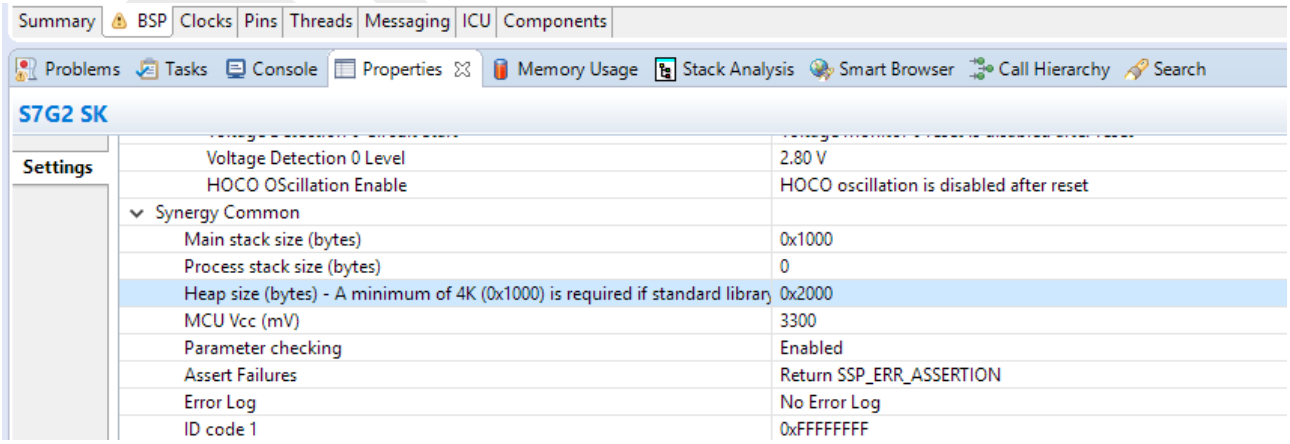

Figure 19: Heap size modification.