RauSach-HiTech Low Level Design

# Overview

This document will describe design for Hardware Abstraction Layer (HAL). This design will make rule for implementation of HAL devices. In scope of project, HAL devices have been limited with following HW:

* Measurement HW
* Controlled HW
* Wifi
* GSM/GPRS
* etc

## Abbreviation/Terminology

|  |  |
| --- | --- |
| **Abb/Term** | **Description/Definition** |
| **HAL** | Hardware Abstraction Layer |
| **HW** | Hardware |
| **COM** | Built-in Communication (Analog/Digital/SPI/I2C) |
| **Msn** | Module short name |
| **Fsn** | Function short name |
| **Csn** | Built-in Communication short name |

## References

|  |  |  |
| --- | --- | --- |
| **Document Name** | **Description** | **Link** |
| **Requirement Analysis** | Requirement Specification for RauSach-HiTech project. | Git |

# Design Strategies

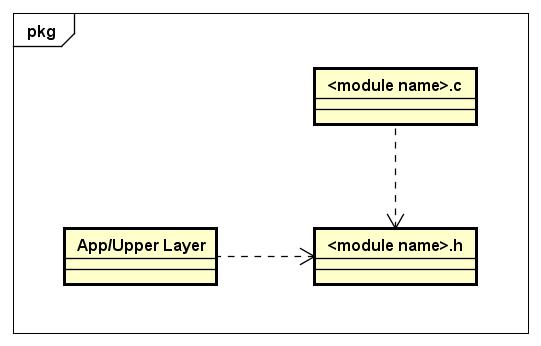
This document will make general rules about implementation for each kind of HW. Each kind of HW will be described separated for all communications which are built-in HW. The responsibility of developer is that they must choose the respective communication which is appropriate to be used to behavior of project.

# Restriction/Assumption/Dependency

# File Structure

Common file structure for all HWs is described as below:

* Source implementation files is named with the convention <module name>.c
* Header implementation files is named with the convention <module name>.h
* <module name>.c will include <module name>.h.
* <module name>.h is exported to User of that HW (Application, Middle Layers, etc).



# Data Model

## Pin\_ConfigSetType

|  |  |  |  |
| --- | --- | --- | --- |
| **Type Name** | **Pin\_ConfigSetType** | | |
| **Description** | This type is used to configure set of pin assigned for one Module | | |
| **Definition Type** | Struct | | |
| **Parameters** | Type | Name | Description |
| uint16\_t | PinNum | Number of Pins |
| Pin\_AssignType \* | PinSet | Set of Pins |

## Pin\_AssignType

|  |  |  |  |
| --- | --- | --- | --- |
| **Type Name** | **Pin\_AssignType** | | |
| **Description** | This type is used to configure separated pin assigned for one Module | | |
| **Definition Type** | Struct | | |
| **Parameters** | Type | Name | Description |
| uint16\_t | PinId | Logical ID used in set of Pins |
| uint8\_t | PinValue | Value of assigned Pin |

# Detailed Design

## Measurement HW

This kind includes variety of sensors such as moisture, temperature, humidity, brightness, pH, etc.

### Assigned Only One Pin

#### <Msn>\_Init

|  |  |  |
| --- | --- | --- |
| **Prototype** | **void <Msn>\_Init (uint8\_t AssignedPin)** | |
| **Description** | This service is used to initialize Input Pin for the Module. | |
| **In/Out Parameter** | AssignedPin | Assigned Input Pin used by this Module |
| **Return Value** | None |  |
| **Operation** | 1. Check if the Module State is not UNINITIALIZED state, the function will report error and return without any actions. 2. Store AssignedPin as global variable 3. Set AssignedPin to INPUT mode 4. Set Module State to INITIALIZED state | |

#### <Msn>\_DeInit

|  |  |  |
| --- | --- | --- |
| **Prototype** | **void <Msn>\_DeInit (void)** | |
| **Description** | This service is used to de-initialize the Module. | |
| **In/Out Parameter** | None |  |
| **Return Value** | None |  |
| **Operation** | 1. Set Module State to UNINITIALIZED state 2. Reset all global variables and HW Registers (if any) | |

#### <Msn>\_Get<Csn><Fsn>Value

|  |  |  |
| --- | --- | --- |
| **Prototype** | **<Return Type> <Msn>\_Get<Csn><Fsn>Value (void)** | |
| **Description** | This service is used to get measurement value from HW | |
| **In/Out Parameter** | None |  |
| **Return Value** | Value is get from HW. The <Return Type> will be accorded to COM of that HW as described below:   * “uint8\_t” if COM is Digital * “float” if COM is Analog/I2C/SPI | |
| **Operation** | 1. Check if the Module State is not UNINITIALIZED state, the function will report error and return 0 without any actions. 2. Read Raw Value on communication with Assigned Pin parameter 3. Compute the Raw Value to Required Value 4. Return the Required Value | |

### Assigned Set of Pins

#### <Msn>\_Init

|  |  |  |
| --- | --- | --- |
| **Prototype** | **void <Msn>\_Init (const Pin\_ConfigSetType\* AssignedPins)** | |
| **Description** | This service is used to initialize Input Pin for the Module. | |
| **In/Out Parameter** | AssignedPins | Set of Assigned Input Pins used by this Module |
| **Return Value** | None |  |
| **Operation** | 1. Check if the Module State is not UNINITIALIZED state, the function will report error and return without any actions. 2. Store AssignedPins as global variable 3. Set all Assigned Pins to INPUT mode 4. Set Module State to INITIALIZED state | |

#### <Msn>\_DeInit

|  |  |  |
| --- | --- | --- |
| **Prototype** | **void <Msn>\_DeInit (void)** | |
| **Description** | This service is used to de-initialize the Module. | |
| **In/Out Parameter** | None |  |
| **Return Value** | None |  |
| **Operation** | 1. Set Module State to UNINITIALIZED state 2. Reset all global variables and HW Registers (if any) | |

#### <Msn>\_Get<Csn><Fsn>Value

|  |  |  |
| --- | --- | --- |
| **Prototype** | **uint8\_t <Msn>\_Get<Csn><Fsn>Value (<Return Type>\* ResultArr)** | |
| **Description** | This service is used to get measurement values from Assigned Pins connected to HW | |
| **In/Out Parameter** | ResultArr | Array to return measurement values  Value is get from HW. The <Return Type> will be accorded to COM of that HW as described below:   * “uint8\_t” if COM is Digital * “float” if COM is Analog/I2C/SPI |
| **Return Value** | NOT\_OK: 0 | Has been broken |
| OK: 1 | The function is OK |
| **Operation** | 1. Check if the Module State is not UNINITIALIZED state, the function will report error and return NOT\_OK without any actions. 2. Read Raw Values from All Assigned Pins 3. Compute the Raw Values to Required Values 4. Store the Required Value to ResultArr 5. Return OK | |

## Controlled HW

### Assigned Only One Pin

#### <Msn>\_Init

|  |  |  |
| --- | --- | --- |
| **Prototype** | **void <Msn>\_Init (uint8\_t AssignedPin)** | |
| **Description** | This service is used to initialize Input Pin for the Module. | |
| **In/Out Parameter** | AssignedPin | Assigned Input Pin used by this Module |
| **Return Value** | None |  |
| **Operation** | 1. Check if the Module State is not UNINITIALIZED state, the function will report error and return without any actions. 2. Store AssignedPin as global variable 3. Set AssignedPin to OUTPUT mode 4. Set Module State to INITIALIZED state | |

#### <Msn>\_DeInit

|  |  |  |
| --- | --- | --- |
| **Prototype** | **void <Msn>\_DeInit (void)** | |
| **Description** | This service is used to de-initialize the Module. | |
| **In/Out Parameter** | None |  |
| **Return Value** | None |  |
| **Operation** | 1. Set Module State to UNINITIALIZED state 2. Reset all global variables and HW Registers (if any) | |

#### <Msn>\_Open

|  |  |  |
| --- | --- | --- |
| **Prototype** | **uint8\_t <Msn>\_Open (void)** | |
| **Description** | This service is used to control the Module. this make sense that the service is used to switch ON the Module | |
| **In/Out Parameter** | None |  |
| **Return Value** | NOT\_OK: 0 | Has been broken |
| OK: 1 | The function is OK |
| **Operation** | 1. Check if the Module State is not UNINITIALIZED state, the function will report error and return NOT\_OK without any actions. 2. Write HIGH to Assigned Pin 3. Return OK | |

#### <Msn>\_Close

|  |  |  |
| --- | --- | --- |
| **Prototype** | **uint8\_t <Msn>\_Close (void)** | |
| **Description** | This service is used to control the Module. this make sense that the service is used to switch OFF the Module | |
| **In/Out Parameter** | None |  |
| **Return Value** | NOT\_OK: 0 | Has been broken |
| OK: 1 | The function is OK |
| **Operation** | 1. Check if the Module State is not UNINITIALIZED state, the function will report error and return NOT\_OK without any actions. 2. Write LOW to Assigned Pin 3. Return OK | |

### Assigned Set of Pins

#### <Msn>\_Init

|  |  |  |
| --- | --- | --- |
| **Prototype** | **void <Msn>\_Init (const Pin\_ConfigSetType\* AssignedPins)** | |
| **Description** | This service is used to initialize Input Pin for the Module. | |
| **In/Out Parameter** | AssignedPins | Set of Assigned Input Pins used by this Module |
| **Return Value** | None |  |
| **Operation** | 1. Check if the Module State is not UNINITIALIZED state, the function will report error and return without any actions. 2. Store AssignedPins as global variable 3. Set all Assigned Pins to INPUT mode 4. Set Module State to INITIALIZED state | |

#### <Msn>\_DeInit

|  |  |  |
| --- | --- | --- |
| **Prototype** | **void <Msn>\_DeInit (void)** | |
| **Description** | This service is used to de-initialize the Module. | |
| **In/Out Parameter** | None |  |
| **Return Value** | None |  |
| **Operation** | 1. Set Module State to UNINITIALIZED state 2. Reset all global variables and HW Registers (if any) | |

#### <Msn>\_Open

|  |  |  |
| --- | --- | --- |
| **Prototype** | **uint8\_t <Msn>\_Open (void)** | |
| **Description** | This service is used to control the Module. this make sense that the service is used to switch ON the Module | |
| **In/Out Parameter** | None |  |
| **Return Value** | NOT\_OK: 0 | Has been broken |
| OK: 1 | The function is OK |
| **Operation** | 1. Check if the Module State is not UNINITIALIZED state, the function will report error and return NOT\_OK without any actions. 2. Write HIGH to all Assigned Pins 3. Return OK | |

#### <Msn>\_Close

|  |  |  |
| --- | --- | --- |
| **Prototype** | **uint8\_t <Msn>\_Close (void)** | |
| **Description** | This service is used to control the Module. this make sense that the service is used to switch OFF the Module | |
| **In/Out Parameter** | None |  |
| **Return Value** | NOT\_OK: 0 | Has been broken |
| OK: 1 | The function is OK |
| **Operation** | 1. Check if the Module State is not UNINITIALIZED state, the function will report error and return NOT\_OK without any actions. 2. Write LOW to all Assigned Pins 3. Return OK | |

## Wifi

<TBD>

## GSM/GPRS

<TBD>

# Error Strategies

<TBD>

# Future improvement

* Add more HW features along with supported more communication.
* Improve file structure, API prototype.
* Consider this design with using OS.

# Alternate Design Approach

<TBD>