|  |  |
| --- | --- |
| **Title:** | **Door Control Module**  **Button Component** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **History** | | | | |
| **Issue status**  (Index) | **Maturity/Date**  (draft/invalid/valid)  (dd-mmm-yyyy) | **Author**  Department | **Check/Release**  Department | **Description** |
| 1.0 | Draft  30/08/2021 | Marco Mares | Marco Mares | Creation of the document |

Table of Contents

[1 Purpose 3](#__RefHeading___Toc357768773)

[2 Definitions and abbreviations 3](#__RefHeading___Toc357768774)

[3 Realization constraints and targets 3](#__RefHeading___Toc357768775)

[4 SW Conceptual design 3](#__RefHeading___Toc357768776)

[5 SW Component internal breakdown 4](#__RefHeading___Toc357768777)

[5.1 Functional Decomposition 4](#__RefHeading___Toc357768778)

[*5.2* Function *<Type> <function name> (type par 1, .., type par n)* 5](#__RefHeading___Toc357768779)

[*5.3* Function <Type*> <function name> (type par 1, .., type par n)* 5](#__RefHeading___Toc357768780)

# Purpose

This document has been created as a demonstration the students to how to create the detailed design from requirements and SW architecture.

# Definitions and abbreviations

**Definitions**

|  |  |
| --- | --- |
| Special Byte | Special byte received to change the baud rate 55h |
|  |  |
|  |  |
|  |  |
|  |  |

**Abbreviations**

Only SW Component specific abbreviations.

References

|  |  |  |
| --- | --- | --- |
| **N°** | **Document name** | **Reference** |
| 1 | SWA\_DCU | SW Architecture description |
| 2 | Traceability Matrix Template.xlsm | SW Requirements description |
|  |  |  |
|  |  |  |

# Realization constraints and targets

The main purpose of the Button Physical Software Component of the ECUAbtraction Layer is to implement all the logic for Button handling on the ECU. It is also responsible to determine button status and report them to corresponding Application SWC for every button on ECU.

# SW Conceptual design

Button Component is connected to the following components using the mentioned interfaces:

* Dio Component – iDio Interface
* HWConfig Component – iHWConfig Interface
* DoorApp Component – iButton Interface

The parts not mentioned in the list above are outside the system boundary.

Dio Component uses the following operations to get Button information about Button Status:

* Dio\_Read\_DoorLock\_Button():PIN\_VALUE
* Dio\_Read\_DoorUnlock\_Button():PIN\_VALUE
* Dio\_Read\_PassengerWindowClose\_Button():PIN\_VALUE
* Dio\_Read\_PassengerWindowOpen\_Button():PIN\_VALUE
* Dio\_Read\_RearLeftWindowClose\_Button():PIN\_VALUE
* Dio\_Read\_RearLeftWindowOpen\_Button():PIN\_VALUE
* Dio\_Read\_RearRightWindowClose\_Button():PIN\_VALUE
* Dio\_Read\_RearRightWindowOpen\_Button():PIN\_VALUE
* Dio\_Read\_RearWindowLock\_Button():PIN\_VALUE
* Dio\_Read\_WindowClose\_Button():PIN\_VALUE
* Dio\_Read\_WindowOpen\_Button():PIN\_VALUE

Button Component uses the following operations to get Button information from Dio:

* Button\_Get\_Door\_Lock():BUTTON\_STATUS
* Button\_Get\_Door\_Unlock():BUTTON\_STATUS
* Button\_Get\_PassengerWindow\_Close():BUTTON\_STATUS
* Button\_Get\_PassengerWindow\_Open():BUTTON\_STATUS
* Button\_Get\_RearLeftWindow\_Close():BUTTON\_STATUS
* Button\_Get\_RearLeftWindow\_Open():BUTTON\_STATUS
* Button\_Get\_RearRightWindow\_Close():BUTTON\_STATUS
* Button\_Get\_RearRightWindow\_Open():BUTTON\_STATUS
* Button\_Get\_RearWindow\_Lock():BUTTON\_STATUS
* Button\_Get\_Window\_Close():BUTTON\_STATUS
* Button\_Get\_Window\_Open():BUTTON\_StATUS

HWConfig component provides the following operations and uses the mentioned data types:

* HWConfig\_IsAny(config: uint8, config\_mask: uint8):Boolean

A list of the used data types and their values is listed below:

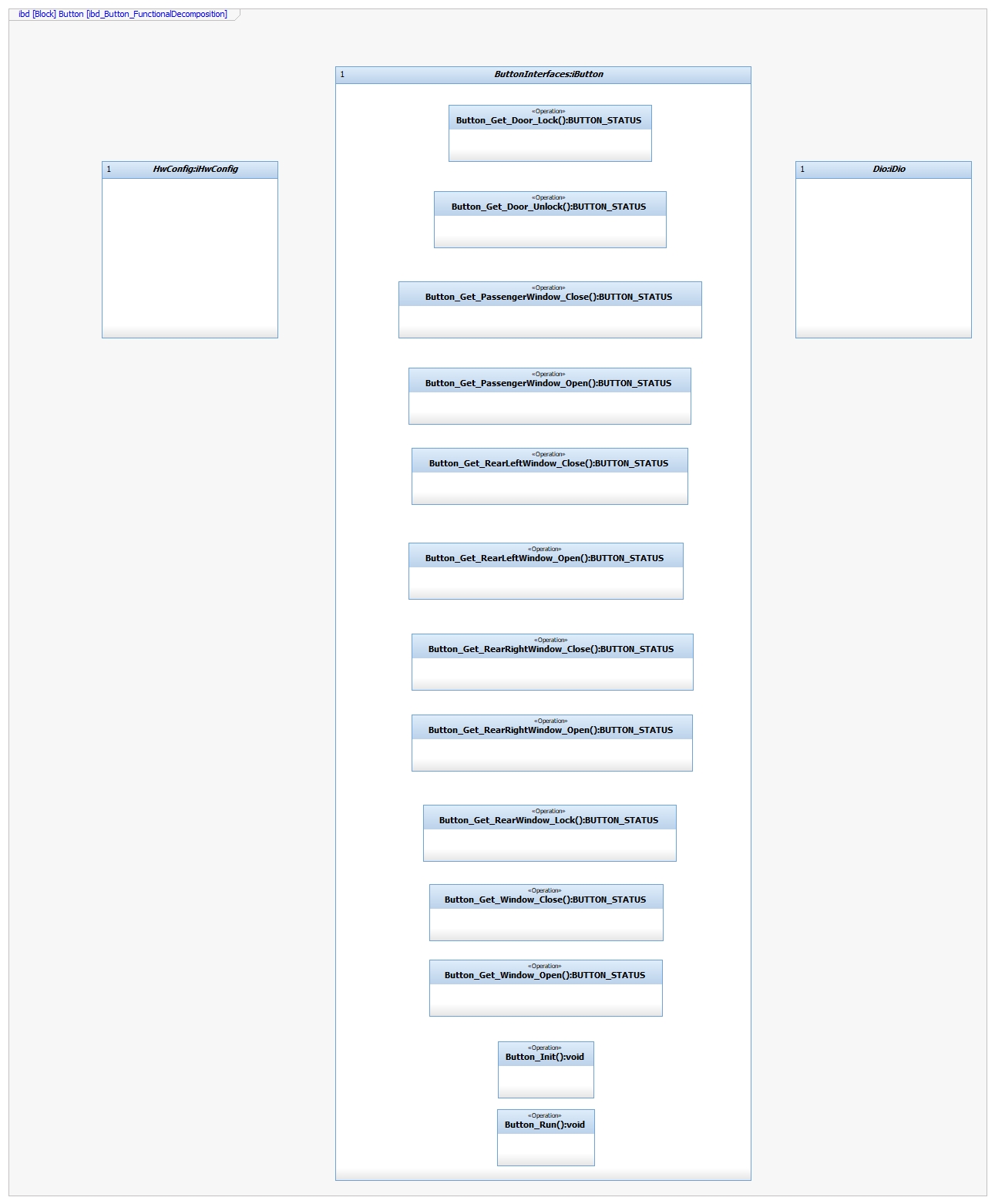
* BUTTON\_STATUS:
  + BUTTON\_NOT\_PRESSED = 0
  + BUTTON\_PRESSED = 1
  + BUTTON\_LONG\_PRESSED = 2
  + BUTTON\_STUCK = 3
* PIN\_VALUE:
  + DIO\_LOW = 0
  + DIO\_HIGH = 1

# SW Component internal breakdown

Button SW Component will not be break into sub-components. The component is not complex enough to be split into sub-components.

## Functional Decomposition

Overview of functions and their dependencies shown by a Static Function Tree

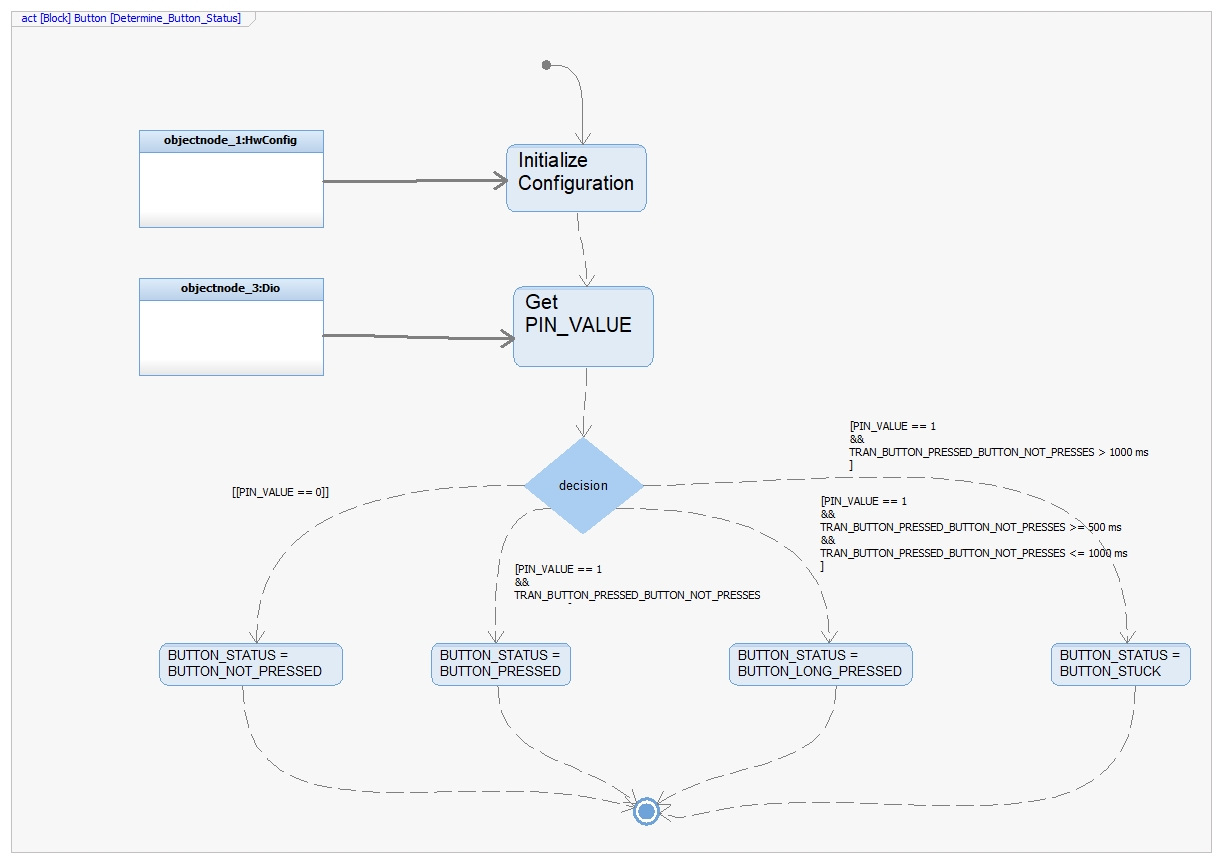


**Function Description and Dynamic Behavior**

## **Function** void Determine\_Button\_Status(BUTTON\_STATUS, PIN\_VALUE)

|  |  |
| --- | --- |
| **Description** | This function initializes and runs Door component and read PIN\_VALUE from Dio Component to determine BUTTON\_STATUS. |
| **Parameter 1** input | PIN\_VALUE |
| **Parameter 2**  output | BUTTON\_STATUS |
| **Return Value** | void |
| **Precondition** | PIN\_VALUE shall transitions from the previous state. |
| **Post condition** | Depending on the value of PIN\_VALUE, BUTTON\_STATUS shall be considered as Not Pressed, Short Press, Long Press or Stuck. |
| **Error Conditions** |  |
| **Requirements** | DCU\_SWR\_1  DCU\_SWR\_2  DCU\_SWR\_3  DCU\_SWR\_4  DCU\_SWR\_66  DCU\_SWR\_67  DCU\_SWR\_68  DCU\_SWR\_69  DCU\_SWR\_70  DCU\_SWR\_71  DCU\_SWR\_72  DCU\_SWR\_73  DCU\_SWR\_74 |

**Dynamic Behavior**

  
The activity diagram describes Determine\_Button\_Status() Operation. This is the only operation needed in order to achieve functionalities described for Button Component. The operation starts reading the variant from HWConfig. This configuration defines which Door and Window the ECU is controlling. Thereafter, Button reads buttons and switches values from Dio component. It is recommended to perform this operation at least every 10 ms. When all the PIN\_VALUES for buttons and switches are read, Button component defines the status for each button and switch. Theses values will be read from DoorApp to perform Debounce operation and control Manual operation.