|  |  |  |  |
| --- | --- | --- | --- |
| Title | Software Subsystem Test Plan <ServicePort Subsystem> | | |
| **Distribution** | Software Archive | | |
| **Author** | Hao Wu | **Date** | 2012-11-30 |
| **Review** | Paul Li  Xiaojin Kuang | **Date** | 2012-12-03 |
| **Approved** | Rock-RendongLi | **Date** | 2012-12-03 |
| **Remarks** |  | | |

Contents

[1 Introduction 3](#_Toc342291284)

[1.1 Scope 3](#_Toc342291285)

[1.2 Assumes 3](#_Toc342291286)

[1.3 Definitions, acronyms and abbreviations 3](#_Toc342291287)

[1.4 Objectives 3](#_Toc342291288)

[1.5 Tools 3](#_Toc342291289)

[2 Module Test 3](#_Toc342291290)

[2.1 How to Create Module Test 3](#_Toc342291291)

[2.2 Module Test Cases 4](#_Toc342291292)

[2.2.1 Initialization 4](#_Toc342291293)

[2.2.2 HMI Presence Pin 4](#_Toc342291294)

[2.2.3 Communication 5](#_Toc342291295)

[2.3 Module Test Summary 9](#_Toc342291296)

[2.3.1 Function Coverage 9](#_Toc342291297)

[2.3.2 Open Issues 9](#_Toc342291298)

[3 Integration Test 9](#_Toc342291299)

[References 9](#_Toc342291300)

[Revision Chart 9](#_Toc342291301)

[Review 10](#_Toc342291302)

**List of Figures**

xx

**List of Tables**

[Table 1: CMD Supported by Service Port 3](#_Toc342291531)

# Introduction

## Scope

The service port subsystem helps to fast configuration of the device. The protocol followed by the service port is same as that of HART, as defined by HCF. The subsystem supports the following HART commands.

Table 1: CMD Supported by Service Port

|  |  |  |
| --- | --- | --- |
| **Command** | **Description** | **Destination** |
| 1 | Read Object | CB or FE |
| 2 | Write Object |
| 3 | Read Memory |
| 4 | Write Memory |

The document provides the plan for the integration test of the subsystem. The test mainly focuses on requirement and communication. The intended audience is designers, developers, and testers of the software subsystem.

## Assumes

* None

## Definitions, acronyms and abbreviations

| **Term** | **Definition** |
| --- | --- |
| 2 Wire | ABB MP new 2 wire device platform based on UHTE |
| HMI | Subsystem providing Human Machine Interface for local user access via display and key input |
| FE | Front-end Board |
| HART | Highway Addressable Remote Transducer |
| CRC | Cyclic Redundancy Check |

## Objectives

The purpose of integration test is to test whether the subsystem works effectively, so need coverage of all functional requirement.

## Tools

|  |  |
| --- | --- |
|  |  |
| Environment | IAR C-SPY |
| Emulator Driver Version | 3.20.1.4 |
| Hardware |  |
| Date | 2012-09-17 |

# Module Test

## How to Create Module Test

1. Open the project in Test folder of the subsystem
2. Enable “**SERVICE\_PORT\_MODULE\_TEST**” macro definition, and some tests may need to enable another macro definition: “**FE\_DEFINE**”
3. The project includes subsystem: **T\_Data\_Obj**, **T\_Unit**, **embOS** and **System,** these are same with current 2Wire project (version 1.6.0).
4. Create stub subsystem: **HMI** and **p\_dp**

## Module Test Cases

The Module test includes three parts:

1. Initialization
2. HMI Presence Pin
3. Communication

### Initialization

Function: *servicePort.unit.Initialize (const T\_UNIT SLOW\*me, TUSIGN8 typeOfStartUp)*

The function initializes hardware, *INIT\_TASKS*, *INIT\_DATA* and *INIT\_CALCULATION*.

#### Test Case 1

* Input Invalid subsystem: *servicePort.unit.Initialize(NULL,INIT\_TASKS);*

The expected result is the function will call *assert()*.

#### Test Case 2

* Tasks Initialization, result = *servicePort.unit.Initialize(SERVICE\_PORT\_ME,INIT\_TASKS);*

The expected result is that result = OK

#### Test Case 3

* Data Initialization, result = *servicePort.unit.Initialize(SERVICE\_PORT\_ME,INIT\_DATA);*

The expected result is that result = OK

### HMI Presence Pin

1. *servicePort.ServPortUpdateActivationStatusSRV()*

The function updates status of **HMI** presence pin

1. servicePort.ServPortGetActivationStatusSRV()

The function returns the activation

#### Test Case 1

* Presence Pin startup check, read the status of Presence Pin

The expected result is that *PresencePinValue*==1;

#### Test Case 2

* Presence Pin disabled, *setPresencePinValue(0)*;

The expected result is that 1= *servicePort.ServPortGetActivationStatusSRV()* , ServicePort is enabled.

#### Test Case 3

* Presence Pin enabled, *setPresencePinValue(1);*

The expected result is that 0= *servicePort.ServPortGetActivationStatusSRV()*, ServicePort is disabled.

#### Test Case 4

* Service Port activation, *setPresencePinValue(0);*

The expected result is that 1= *servicePort.ServPortGetActivationStatusSRV()*, ServicePort is enabled.

### Communication

The following cases mainly focus on communication layer 2: data link layer (*void servicePortReceive(void)*

, work as receive state machine) and layer 7: application layer(*servicePort.ServPortCommandInterpreterEXE ()*, Interprets the command).

Create a stub subsystem **p\_dp** of which the write object test will execute.

#### Test Case 1

* Send a correct Frame (1) - Long Address
* Data frame: 255,255,255,255,255,130,26,7,0,0,0,1,8,0,0,0,4,0,0,255,255,146

The expected result: *SpLayer2flags.layer7requestedBit*=1, which means receive a valid frame and response frame has to be generated.

#### Test Case 2

* Send a correct Frame (2) - Short Address
* Data frame: 255,255,255,255,255,2,0,0,8,0,0,0,4,0,0,255,255,146

The expected result: *SpLayer2flags.layer7requestedBit*=1, which means receive a valid frame and response frame has to be generated.

#### Test Case 3

* Send a correct Frame (3) - Long Address & expansion byte
* Data frame: 255,255,255,255,255,162,26,7,0,0,0,1,1,8,0,0,0,4,0,0,255,255,146

The expected result: *SpLayer2flags.layer7requestedBit*=0, which means response code needn’t be generated. Because frame with expansion byte was only monitored and don't need to respond.

#### Test Case 4

* Send a correct Frame (4) - Short Address & expansion byte
* Data frame: 255,255,255,255,255,34,0,1,0,8,0,0,0,4,0,0,255,255,146

The expected result: *SpLayer2flags.layer7requestedBit*=0, which means response code needn’t be generated. Because frame with expansion byte was only monitored and don't need to respond.

#### Test Case 5

* Send a correct Frame (5) - Long Address & expansion byte
* Data frame: 255,255,255,255,255,130,26,0,0,0,0,1,8,0,0,0,4,0,0,255,255,146

The expected result: *SpLayer2flags.layer7requestedBit*=0, which means response code needn’t be generated. Because frame with expansion byte was only monitored and don't need to respond.

#### Test Case 6

* Send a correct Frame (6) - Long Address & number of bytes = 0
* Data frame: 255,255,255,255,255,130,26,7,0,0,0,1,0,158

The expected result: *SpLayer2flags.layer7requestedBit*=1, which means receive a valid frame and response frame has to be generated..

#### Test Case 7

* Send a valid Frame with UART error on preambles
* Data Frame: 255,255,255,255,255,130,26,7,0,0,0,1,8,0,0,0,4,0,0,255,255,146
* Just send the first byte with UART error.

The expected result: SpLayer2flags.layer7requestedBit=1, which means response code has to be generated. Because the receive handler checks UART error, if error happen then send the relative response with no regard to data frame.

#### Test Case 8

* Send a valid Frame with uart error on delimiter
* Data Frame: 255,255,255,255,255,130,26,7,0,0,0,1,8,0,0,0,4,0,0,255,255,146

The expected result: *SpLayer2flags.layer7requestedBit*=1, which means response code has to be generated. Because the receive handler checks UART error, if error happen then send the relative response with no regard to data frame.

#### Test Case 9

* Send a valid Frame with a slave to master request
* Data frame: 255,255,255,255,255,6,26,7,0,0,0,1,8,0,0,0,4,0,0,255,255,146

The expected result: *SpLayer2flags.layer7requestedBit*=0, which means response code need not be generated.

#### Test Case 10

* Send an a valid Frame with incorrect preamble
* Data Frame: 255,255,255,***254***,255,2,26,7,0,0,0,1,8,0,0,0,4,0,0,255,255,146

The expected result: *SpLayer2flags.layer7requestedBit*=0, which means response code need not be generated. Because the number of preamble is not enough.

#### Test Case 11

* Send an valid Frame with short address and uart error in the address status
* Data Frame: 255,255,255,255,255,2,0,0,8,0,0,0,4,0,0,255,255,146

The expected result: *SpLayer2flags.layer7requestedBit*=1, which means response code has to be generated. Because the receive handler checks UART error, if error happen then send the relative response with no regard to data frame.

#### Test Case 12

* Send a valid Frame with long address and uart error in the address status
* Data Frame: 255,255,255,255,255,130,26,7,0,0,0,1,8,0,0,0,4,0,0,255,255,146

The expected result: *SpLayer2flags.layer7requestedBit*=1, which means response code has to be generated. Because the receive handler checks UART error, if error happen then send the relative response with no regard to data frame.

#### Test Case 13

* Send an invalid Frame with short address and command = 1
* Data frame: 255,255,255,255,255,2,0,1,8,0,0,0,4,0,0,255,255,146

The expected result: *SpLayer2flags.layer7requestedBit*=0, which means response code need not be generated. Because the communication can only support short address when command=0.

#### Test Case 14

* Send a valid Frame with long address and uart error in the number of byte status
* Data Frame: 255,255,255,255,255,130,26,7,0,0,0,1,8,0,0,0,4,0,0,255,255,146

The expected result: *SpLayer2flags.layer7requestedBit*=1, which means response code has to be generated. Because the receive handler checks UART error, if error happen then send the relative response with no regard to data frame.

#### Test Case 15

* Send an invalid Frame with long address and buffer overflow

The expected result: *SpLayer2flags.layer7requestedBit*=1, which means response code has to be generated

#### Test Case 16

* Read value (1) - Attribute index WHOLE\_OBJECT
* Data frame: 255,255,255,255,255,130,26,7,0,0,0,1,8,0,0,0,4,0,0,255,255,146

The expected result: *SpLayer2flags.layer7requestedBit*=1, *ServPortTxBuffer.firstResp* = *SP\_RESP\_OK*

#### Test Case 17

* Read value (2) - Attribute index != WHOLE\_BOJECT
* Data frame: 255,255,255,255,255,130,26,7,0,0,0,1,8,0,0,0,4,0,0,0,0,146

The expected result:

1. *SpLayer2flags.layer7requestedBit*=1, means response code has to be generated
2. *ServPortTxBuffer.firstResp* = *SP\_RESP\_OK*, mean there is no specific command error.

#### Test Case 18

* Read value (3) - Invalid sybsystem index
* Data frame: 255,255,255,255,255,130,26,7,0,0,0,1,8,0,0,0,0,0,0,255,255,150

The expected result:

1. *SpLayer2flags.layer7requestedBit*=1, means response code has to be generated
2. *ServPortTxBuffer.firstResp* = *SP\_RESP\_WARN\_INVALID\_SELECTION*, mean code or index not allowed.

#### Test Case 19

* Read value (4) - Invalid object index
* Data frame: 255,255,255,255,255,130,26,7,0,0,0,1,8,0,0,0,4,0,2,255,255,144

The expected result:

1. *SpLayer2flags.layer7requestedBit*=1, means response code has to be generated
2. *ServPortTxBuffer.firstResp* = *SP\_RESP\_WARN\_INVALID\_SELECTION*, mean code or index not allowed .

#### Test Case 20

* Read value (7) - Read on an incorrect Board index
* Data frame: 255,255,255,255,255,130,26,7,0,0,0,1,8,2,0,0,4,0,0,255,255,144

The expected result:

1. *SpLayer2flags.layer7requestedBit*=1, means response code has to be generated
2. *ServPortTxBuffer.firstResp* = *SP\_RESP\_WARN\_INVALID\_SELECTION*, mean code or index not allowed for cmd

#### Test Case 21

* Write value (1) - Attribute index WHOLE\_OBJECT
* Data frame: 255,255,255,255,255,130,26,7,0,0,0,2,12,0,0,0,4,0,0,255,255,0,0,0,0,149

The expected result:

1. *p\_dP\_Get(P\_DP\_IDX\_pdpUnitValue, WHOLE\_OBJECT, &testValue)*, testValue=0
2. *ServPortTxBuffer.firstResp* =*SP\_RESP\_OK*

*p\_dP\_Get()* is stub function for the created stub subsytem

#### Test Case 22

* Write value (2) - Invalid sybsystem index
* Data frame: 255,255,255,255,255,130,26,7,0,0,0,2,12,0,0,0,0,0,0,255,255,0,0,0,0,145

The expected result:

1. *SpLayer2flags.layer7requestedBit* =1
2. *ServPortTxBuffer.firstResp* =*SP\_RESP\_WARN\_INVALID\_SELECTION*

#### Test Case 23

* Write value (3) - Invalid object index
* Data frame: 255,255,255,255,255,130,26,7,0,0,0,2,12,0,0,0,4,0,2,255,255,0,0,0,0,151

The expected result:

1. *ServPortRxBuffer.firstResp* = *SP\_RESP\_OK*
2. *ServPortTxBuffer.firstResp* =*SP\_RESP\_WARN\_INVALID\_SELECTION*

#### Test Case 24

* Write value (5) - Write on an incorrect Board index
* Data frame: 255,255,255,255,255,130,26,7,0,0,0,2,12,2,0,0,4,0,0,255,255,0,0,0,0,151

The expected result:

1. *ServPortRxBuffer.firstResp* = *SP\_RESP\_OK*
2. *ServPortTxBuffer.firstResp* =*SP\_RESP\_WARN\_INVALID\_SELECTION*

#### Test Case 25

* Read memory (1) - Correct reading
* Data frame: 255,255,255,255,255,130,26,7,0,0,0,3,5,0,0,0xAA,0xAA,0,153

The expected result:

1. *SpLayer2flags.layer7requestedBit* =1
2. *ServPortTxBuffer.firstResp* = *SP\_RESP\_OK*

#### Test Case 26

* Read memory (2) - Fe board reading
* Data frame: 255,255,255,255,255,130,26,7,0,0,0,3,5,1,0,0xAA,0xAA,0,152

The expected result:

1. *SpLayer2flags.layer7requestedBit* =1
2. *ServPortTxBuffer.firstResp* =*SP\_RESP\_WARN\_INVALID\_SELECTION*

#### Test Case 27

* Read memory (3) - Buffer overflow
* Data frame: 255,255,255,255,255,130,26,7,0,0,0,3,5,0,0,0xAA,0xAA,33,0xb8

The expected result:

1. *SpLayer2flags.layer7requestedBit* =1
2. *ServPortTxBuffer.firstResp* =*SP\_RESP\_WARN\_INVALID\_SELECTION*

#### Test Case 28

* Write memory (1) - Correct writing
* Data frame: 255,255,255,255,255,130,26,7,0,0,0,4,5,0,0,0xAA,0xAA,0,0x9e

The expected result:

1. *SpLayer2flags.layer7requestedBit* =1
2. *ServPortTxBuffer.firstResp* = *SP\_RESP\_OK*

#### Test Case 29

* Write memory (3) - Buffer overflow
* Data frame: 255,255,255,255,255,130,26,7,0,0,0,4,5,0,0,0xAA,0xAA,33,0xbf

The expected result:

1. *SpLayer2flags.layer7requestedBit* =1
2. *ServPortTxBuffer.firstResp* =*SP\_RESP\_WARN\_INVALID\_SELECTION*

#### Test Case 30

* Frame correct, too few bytes for a read command
* Data frame: 255,255,255,255,255,130,26,7,0,0,0,3,4,0,0,0xAA,0xAA,0x98

The expected result:

1. *SpLayer2flags.layer7requestedBit* =1
2. *ServPortTxBuffer.firstResp* = *SP\_RESP\_ERR\_TOO\_FEW\_DATA*

#### Test Case 31

* Command not implemented
* Data frame: 255,255,255,255,255,130,26,7,0,0,0,5,5,0,0,0xAA,0xAA,0,0x9f

The expected result:

1. *SpLayer2flags.layer7requestedBit* =1
2. *ServPortTxBuffer.firstResp* = *SP\_RESP\_ERR\_CMD\_NOT\_IMPLEMENTED*

#### Test Case 32

* CRC error
* Data frame: 255,255,255,255,255,130,26,7,0,0,0,5,5,0,0,0xAA,0xAA,0,0xbf

The expected result:

1. *SpLayer2flags.layer7requestedBit* =1
2. *ServPortTxBuffer.firstResp* = *SP\_CHECKSUM\_ERROR*

## Module Test Summary

### Function Coverage

The module test of the ServicePort subsystem includes three parts: initialization, HMI presence port and communication. The chapter *2.2 Module Test Cases* provides enough cases to cover every situation so as to achieve module test objective.

### Open Issues

The Layer1 mainly involve with hardware, could not be tested in this step, this could be tested in integration test.

# Integration Test

The ServicePort subsystem integration test will be integrated into system integration test.

For more details, please refer to integration test plan.

References

|  |  |
| --- | --- |
| **Ref.** | **Document** |
| [1] |  |
|  |  |

Revision Chart

|  |  |  |  |
| --- | --- | --- | --- |
| **Rev.** | **Description of Version/Changes** | **Primary Author(s)** | **Date** |
| 00 | First Release | Hao Wu | **2012-11-30** |

Review

**First Review**

|  |  |
| --- | --- |
| **Document Revision:** | 00 |
| **Review Date:** | 2012-12-03 |

**Review-Participant:**

|  |  |  |
| --- | --- | --- |
| *Dept.* | *Name* |  |
| R&D ShangHai | Paul Li  Xiaojin Kuang | 2012-12-03 rev.00 |

**Decision of the Review:**

|  |  |  |
| --- | --- | --- |
|  | *Decision* | *next steps* |
| **X** | Inspection passed ***without restrictions*** | Phase finished |
|  | Inspection passed ***with restrictions*** | some changes must be done |
|  | Inspection ***not*** passed | Inspection must be repeated |

**Changes Are Proved:**

The Reviewer confirms that all changes are done:

|  |  |  |
| --- | --- | --- |
| proved Rev: | Date: | Reviewer: |
| 00 | 2012-12-03 | Paul Li  Xiaojin Kuang |

**Check list:**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | yes | no |
| 1. | The Module Tests are passed successful and documented in a proper way. | **X** |  |
| 1. | The Integration Tests are passed successful and documented in a proper way. |  | **X** |
| 2. | The Code Reviews are passed successful and documented in a proper way. | **X** |  |
| 3. | All safety requirements (Safety Function/Integrity/Measures) are considered. |  | **X** |
| 4. | The accomplished Test Procedures are sufficient to start the Product System Test. | **X** |  |
| 5. | Are all open issues transferred to the defects table? | **X** |  |

**Remarks:**

**Defects**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No. | Checkpoint | Description | Major Defect | Done date |
| 1 | 3 Integration Test | The ServicePort subsystem integration test will be integrated into system integration test. | N | 2012-12-03 |