Test Report

Vector SLP3 Flash Bootloader

General Information

Component	
Test Person	Li Wenhe
Test Proceeding Date	2018-03-27
Test Report Recipients	Vector Informatik GmbH, Customer

Test Environment

Component		Version / Derivative
Target System	RH850	R7F701313EAFP
Generation Tool	GENy	01.04.50
Compiler	Green Hills	2015.1.7
Customer	Nexteer Automotive (Suzhou) Co.,Lt	
License	CBD1701035	
Project Number	5022666	
Test Database	DemoFBL_Vector_SLP3.dbc	

Component under Test

Component Type	Component Name	Version
CAN FBL (SLP3)	FblOem_Vector	07.03.01

Overall Test Result	OK
Details / hints	

Revision history

Author	Date	Version	Remarks
Achim Strobelt	2016-09-07	02.00.00	Converted to new template, removed tests which are done by automatic tests, added tests for pipelining options
Achim Strobelt	2017-03-15	02.01.00	Changed gap fill test value
Achim Strobelt	2017-08-14	03.00.00	Adapted tests for Vector SLP3 7.xx.xx

Test Report distribution

After finishing the tests, this test report must be stored in the _doc\30_TestReports directory. A copy of this document is included in the Bootloader delivery as PDF.

Glossary and Abbreviations

See chapter 6.

Contents

1 Introduction

This document serves as the Test Report for testing new integrations of the Vector SLP3 Flash Bootloader.

2 Test Environment

2.1 Module Versions, Compiler/Linker information

See file “CBD1701035_DeliveryDescription.html”.

2.2 Components

The test requires the following components:

- CAN Bootloader software on a target ECU/ board
- vFlash as Flash Tool
- CANoe
- Example ECU application
- HexView
- CANoe Automated Test Suite

This test report describes two sets of tests:

- Manual tests
- CANoe automated tests

2.3 Test Setup

The tests are based on a standard Bootloader configuration. The following base configuration is assumed:

- 8 segments per block
- 2 mandatory logical blocks
- Project State: “Integration”
- Flash Driver Usage: Download Driver
- Diagnostic Buffer Size: 4095 Bytes
- Security Class which was ordered for this Bootloader
- Hardware dependent settings matching the customer’s requirements
- Other settings defined by pre-config file

3 Configuration Tests

Configuration tests include white box tests and basic compile test for various features.

3.1 White Box Tests

This test is a verification of the source code before tests of the running Bootloader are performed.

Test-No.	Test Name	Test Entry	Expected Result	Result
00.01	**Flash Block Structure** This structure describes the mapping of flash segments on a specific controller	> The entries of the FlashBlock structure must correspond to the resource mapping of the supported controllers. > The end addresses of the blocks are the addresses of the last byte in the corresponding block.		OK

3.2 Compile Tests

The following tests are used to verify several Bootloader configurations compile without error.

Test-No.	Test Name	Test Entry	Expected Result	Result
01.01	**Project states**	Compile the Bootloader with each of the following project states > Integration > Production > Production (reduced checks)	Bootloader is compiled without compile errors.	**OK**
01.02	**Application task**	> Compile the Bootloader with and without Application Task	Bootloader is compiled without compile errors.	**OK**
01.03	**Watchdog**	> Compile the Bootloader with enabled and disabled watchdog support.	Bootloader is compiled without compile errors.	**OK**

4 Implementation Tests

4.1 Download Sequence Tests 1

The following tests are used to verify the standard download sequence. The following configuration will be used for these tests:

> 2 mandatory logical blocks (Application and Data)

> A running application has to be used for the application block.

> One of both blocks has to cross a flash block boundary.

Test-No.	Test Name	Test Entry	Expected Result	Result
02.01	**Program application from FBL**	> Download the application to the ECU.	> Application is downloaded correctly. The first logical block is validated after checksum/signature check. > The application is not started because the second logical block is missing.	OK
02.02	**Program data block from FBL**	> Download data block (second logical block)	> Data block is downloaded correctly. The second logical block is validated after checksum/signature check. > The application is started because both logical blocks are present.	OK
02.03	**Reprogram both blocks from application**	> The Bootloader is started from a running application. Both logical blocks are reprogrammed.	> Transition from application to Bootloader is working correctly. > Both logical blocks are erased and reprogrammed. > The application is running after reprogramming is finished.	OK

Test-No.	Test Name	Test Entry	Expected Result	Result
02.04	**Erase Application**	> Start another download of the application > Interrupt the download, e.g. by resetting the ECU.	> The application is not started again, resets start the Bootloader.	OK
02.05	**Reprogram Application from interrupted Download**	> Download the application to the ECU	> The application is started after a successful download.	OK
02.06	**Startup of valid application**	> Make sure a valid application is programmed. > Reset the ECU several times.	> Application is started after every reset.	OK

4.2 Download Sequence Tests 2

The following tests use a different configuration:

- 2 mandatory logical blocks, 1 optional logical block
- The optional block uses the multiple memory interface, e.g. an Eeprom driver.
- Flash Driver Usage: Optional Flash Driver Download
- Activate pipelined programming and pipelined verification (if ordered)[1].
- Please note: Test cases 03.02, 03.03 and 03.04 require no additional downloads.

Test-No.	Test Name	Test Entry	Expected Result	Result
03.01	**Download Sequence**	Try the following sequences with vFlash (FD flash driver, AP application, CA1 and CA2: calibration files) > FD, AP > FD, AP, CA1 > FD, AP, FD, CA1 > FD, AP, FD, CA1, FD, CA2 > FD, AP, CA1, CA2 > FD, AP, CA1, FD, CA2 > AP > AP, FD, CA1, CA2	> The application is started if both mandatory blocks (AP and CA1) are present.	OK
03.02	**Multiple memory devices**	> Flash driver selection is done by multiple memory interface.	> Verify data is downloaded to CA2 (e.g. by Debugger).	OK

[1] These features have to be ordered explicitly and are not tested if they are not ordered.

Test-No.	Test Name	Test Entry	Expected Result	Result
03.03	**Pipelined programming**	> Verify pipelined programming feature	> Download should be OK > Transfer Data Requests should be answered without a response pending messages[2].	N.a.
03.04	**Pipelined verification**	> Verify pipelined verification feature	> Checksum/Signature calculation should be OK. > Routine Control/Check Memory should be processed without response pending message.[3]	N.a.

[2] If response pending messages are still observed, the response times should be shorter than without pipelined programming.

[3] If response pending messages are still observed, the response times should be shorted than without pipelined verification.

4.3 Special Configurations

The following tests are used to verify special Bootloader features are working correctly.

Test-No.	Test Name	Test Entry	Expected Results	Result
04.01	**Multiple Nodes**	Configure the Bootloader to use a multiple node configuration. > Configure the Bootloader to use node “Demo2” > Start the Bootloader	> The Bootloader communicates using the CAN identifiers of “Demo2”. > The CAN identifiers of node “Demo” are ignored.	OK
04.02	**Sleep Mode**	> Enable Bootloader Sleep Mode > Send a CAN message, verify communication is working. > Stop communication	> `ApplFblBusSleep()` is reached after the configured sleep time expired.	OK
04.03	**Sleep Mode**	> Enable Bootloader Sleep Mode > Send functional tester present messages (e.g. every 2500ms)	> `ApplFblBusSleep()` is not reached.	OK
04.04	**Flash Driver from ROM**	> Configure the Flash Driver Usage to “Use Flash Driver from ROM”. > Download application without flash driver. > Download application with flash driver.	> The application can be flashed without a flash driver download. > The flash driver download is rejected by the bootloader.	OK
04.05	**Changed download sequence**	> Deactivate the following options: Control DTC option Record and Check Programming Preconditions Service. Set Communication Control Type to “Disable Rx and Tx”. > Use the same settings in vFlash	> Download of first logical block should succeed.	OK

Test-No.	Test Name	Test Entry	Expected Results	Result
04.06	**Response after reset**	> De-activate “Response after Reset”. > Set the Bootloader to programming session and send a default session request (10 01) to the Bootloader. > Send a reset request (11 01) to the Bootloader.	> Both requests are responded at once.	OK
04.07	**Response after reset**	> Re-activate response after reset. > Set the Bootloader to programming session and send a default session request (10 01) to the Bootloader. > Send a reset request (11 01) to the Bootloader.	> Both requests are not responded at once. > The Bootloader should send a response pending message, issue the reset and send the positive response after the reset.	OK
04.08	**Force Boot Mode**	> Enable “Stay In Boot” in GENy > Download an invalid application software (e.g. random data) without working CAN/Diagnosis to the ECU. > Start a download using the “Force Boot Mode” feature of vFlash.	> The Bootloader can be started if a corrupt application has been validated. > A working application is flashed onto the ECU. > After flashing the correct application, the application can be reprogrammed using the normal sequence.	OK
04.09	**Use of ApplFblStart()**	> Enable “ApplFblStart Function” in GENy. > Adapt your application to use ApplFblStart. > Start download from running application.	> Programming session can be started from a running application without using NV-memory. > Reprogramming of application is possible.	OK

Test-No.	Test Name	Test Entry	Expected Results	Result
04.10	**Use of Presence Patterns**	> Enable “Presence Pattern” in GENy > Enable Application Validity Flag 1. Generate, re-build and program the Bootloader, the application area is erased. 2. Check the mask/pattern area at the end of the logical blocks (e.g. with the debugger). 3. Download a valid application (two blocks). 4. After the download has finished, check the mask/pattern area at the end of the second logical block. 5. Set a breakpoint in function `FblDiagRCStartErase-MainHandler()`, directly behind the call of function `ApplFblInvalidateBlock()`. 6. Start another download of the application file (first logical block). After the breakpoint has been hit, check the mask/pattern area. 7. Resume the debugger and wait until the (positive) response for the erase request(s) has been received. 8. Stop the debugger and check the mask/pattern area of the first logical block. 9. Restart the bootloader and make another download of the first logical block.	> General: The size of masks and pattern is an integer multiple of the device’s flash segment size. The write pattern size is extended automatically to match the device’s flash segment size. > Step 2: All bytes should equal the (device specific) flash deleted value. > Step 4: The application is started. The mask should have flash deleted value, the pattern area should show the completely written application valid pattern as defined in the fbl_apnv file “kFblPresencePattern={0x73, 0x6A, 0x29, 0x3E}”. > Step 6: The mask of the second logical block should show the inverse value of the application valid pattern written in Step 4. > Step 8: The mask/pattern area has been erased. That is, mask and pattern shall show the flash erased value again. > Step 9: The valid pattern has to be set at the first logical block.	OK

Test-No.	Test Name	Test Entry	Expected Results	Result
04.11	**Startup application validation**	> De-activate “Application Validity Flag”. > Flash a valid application (2 logical blocks).	> The result of the Check Programming Dependencies service is not stored any more. > During startup, ApplFblIsValidApp is used to check if all mandatory logical blocks are present. > `ApplFblCheckProgDependencies()` is called during startup to establish compatibility between both logical blocks. > Application is started.	OK
04.12	**Gap Fill**	> Enable “Gap Filling” > Set “Fill Code” to 0xC5 > Download a file which doesn’t start at the beginning of a logical block, includes a gap between to segments and doesn’t end at the end of the logical block.	> The areas which aren’t occupied by the downloaded data are padded with 0xC5.	OK
04.13	**Data Compression**[4]	> Activate data compression and download compressed files: > Worst case compression (random data). File should be bigger than uncompressed file. > Best case compression (short pattern) > Application software > Flash driver	> All files must be downloaded and flashed correctly.	N.a.

[4] This test is optional and only done if data compression has been ordered.

Test-No.	Test Name	Test Entry	Expected Results	Result
04.14	**Erase Function**	> Configure on logical block which includes the complete flash memory > Start another download (complete flash memory). > Stop the download after erase.	> Verify all memory has been erased, e.g. by setting a breakpoint at the end of `FblEraseRoutine()` and verifying the erase with the debugger.	**OK**

4.4 Download Error Tests

The following tests create some defined error conditions and verify if these errors are processed correctly by the Bootloader.

Test-No.	Test Name	Test Entry	Expected Results	Result
05.01	**Segment Number Check**	> Configure “Max segments per block” to 1 > Download a file which contains more than 1 block.	> The Bootloader must reject the download.	OK
05.02	**Remove Vfp**[5]	> Remove Vfp from the flash device. > Start a download.	> The Bootloader must reject the download.	OK
05.03	**Misaligned Address Region**	> Try to download a (dummy) file where the start address of the address region is not aligned to the flash segment size.	> The Bootloader must reject the download.	OK
05.04	**Bootloader memory overlap**	> Try to download the Bootloader Hex file	> The Bootloader must reject the download. > The Bootloader must be working after this test.	OK
05.05	**Programming Attempt Counter**	> Set the number of allowed programming cycles to 1 and erase the NV-memory of the Bootloader > Program the application > Reprogram the application	> The first programming attempt must succeed and the application should start. > The second programming attempt must be rejected by the Bootloader. The application should start again.	OK

[5] This test is only applicable on some hardware platforms, e.g. V850.

4.5 Test of Security Features

The following tests are used to verify the security features ordered with this Bootloader (e.g. AES encryption or signature verification) are working.

Test-No.	Test Name	Test Entry	Expected Results	Result
06.01	**Signature verification (Security class C and CCC)**[6]	Test signature verification > Security class C. > Security class CCC. > Make one download with correct checksums/signatures and one download with incorrect checksums/signatures.	> Signature verification is working for all requested classes. > Application can be downloaded if correct checksums/signatures are used. > Applications with incorrect checksums/signatures are rejected.	N.a.
06.02	**Encryption**[7]	> Test AES encrypted files if available (Security class AAA).	> Downloading of encrypted flash driver and application is possible. > The application is running correctly.	N.a.

[6] Optional feature – only tested if ordered. Security class DDD is verified by the automatic tests.

[7] Decryption is an optional feature. The test is omitted if no decryption module is ordered.

4.6 CAN Bootloader Specific Tests

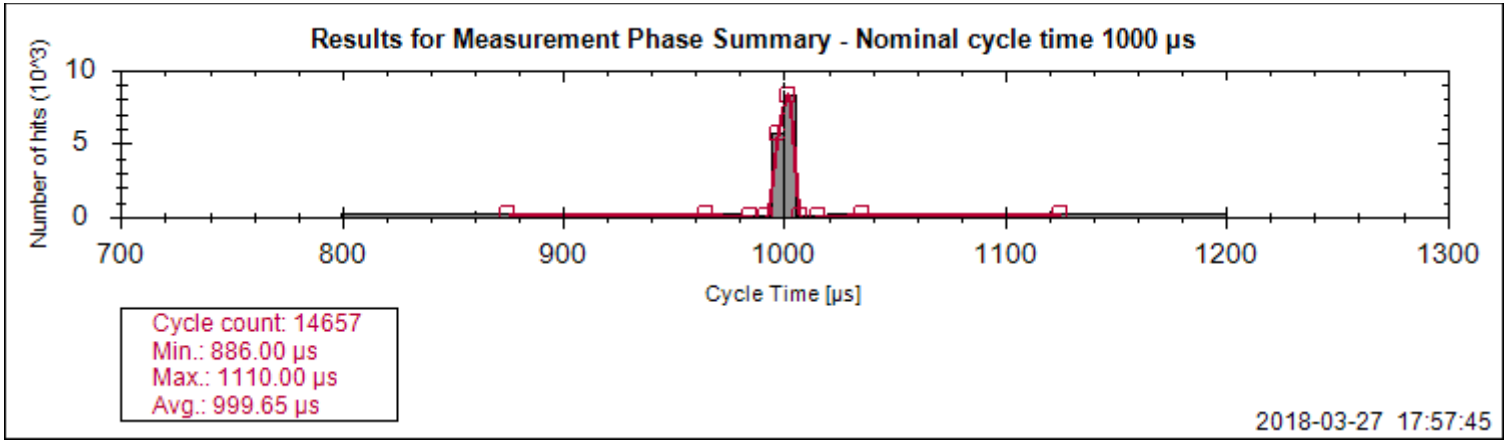
These tests are used to create special bus conditions and to verify CAN bus error handling.

Test-No.	Test Name	Test Entry	Expected Results	Result
06.01	**Bus Off Handling**	> Set a breakpoint in `ApplFblCanBusOff` > Create a bus off condition by disturbing the ECU's Tx message using CANstress.	> The breakpoint should be hit. > When the bus off condition is removed, the Bootloader should be able to restore communication.	OK
06.02	**Bootloader Startup with dominant Rx**	> Use CANstress to generate a dominant level on the bus. > Start ECU.	> ECU must be running – no endless loop in CAN hardware initialization.	OK
06.03	**High Bus Load**	> Generate high bus load with an IG in CANoe (bus load >= 75%). > Perform a download.	> The download must be successful.	OK
06.04	**CAN message reception (configuration dependent)**	Verify CAN message reception for special configurations (29 Bit IDs, NormalFixed or Extended Addressing). Normal Addressing with 11 Bit IDs is verified by the automatic test suite. > Send diagnostic messages from the receive message range to the ECU	> Only messages which should be received by the ECU should be processed. > Other messages should be discarded.	OK

4.7 Watchdog Tests

Test-No.	Test Name	Test Entry	Expected Results	Result
07.01	**Watchdog Refresh Rate**	> Activate watchdog handling, set watchdog time to 1ms. > Add pin trigger to function `ApplFblWdTrigger()`. > Use ERICle.Wdg to measure the trigger timings. > Perform a download.	> The actual trigger period should be close to the period specified in the Bootloader throughout the whole download. > Insert ERICle.Wdg protocol below.	OK

Measurement Result: Refresh Rate 1ms



5 Automated Tests

The following tests are used to verify the diagnostic implementation.

Please use the following configuration:

- 2 mandatory logical blocks
- Security class DDD
- Other settings should be set to default configuration

Test-No.	Test Name	Result
08.01	**Diagnostic Frame Test**	**OK**
08.02	**Diagnostic Sequence Test**	**OK**
08.03	**Half-Automatic Test**	**OK**

6 Glossary and Abbreviations

Abbreviation	Description
OK	The obtained result was in accordance to the expected.
N. a.	Not applicable. The selected test configuration excludes this test.
N. t.	Not tested. The test was not proceeded.
FAILED	The obtained result was **not** in accordance to the specified.

7 Contact

Visit our website for more information on

- News
- Products
- Demo software
- Support
- Training data
- Addresses

www.vector.com