Published by :
http://www.ijert.org

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 10 Issue 05, May-2021

Software Update Test Bench for Smart Cards Usinga Simulator

Srinivas V N
MTech in Communication SystemsRV College of Engineering Bengaluru, India

Dr. Ramavenkateswaran Nagarajan
Asst Professor
Dept of Electronics and CommunicationRV College of Engineering Bengaluru, India

***Abstract*—The Smart Card evolved when it was embedded with a memory or processor chip, which allowed it to support multiple functionalities. Just like the smartphone is a commonly used device in today's world, that can perform various functionalities with the help of modules that are installed on it, called applications, the Smart Card evolved to have applets and a main software platform which manages these applications. Software updates are a common occurrence in our world, populated by technology and their software applications used to interact with them. These software updates can be in simplistic updates with bug fixes or complex with entire parts of the software changed. The Smart Card also requires updates to the main software platform to ensure that there are no errors or bugs which can reduce or stop functionality when it is handing applications or performing tasks. Before the software update can be released to devices, it must be tested first for faults that can cause errors of any magnitude. Thus, a test bench that can test the software updates for the smart card for different use-cases, while recreating the entire update procedure was required, and created. To ensure cost effective and safe testing for the use cases and the update itself, the entire smart card was emulated in a simulator which was used as the main object of testing. The created test bench extensively separated and modularized the software update steps for each use-case so that each aspect of the update can be tested individually.**

***Index Terms*—Smart Card, Software Update, ELF, Upgrade Session, ELF Upgrade Session Status**

I. Introduction

Technological progress in hardware demands equal progress from the software that it contains. Thus, the software is the only component that in the modern day is shared between the user and manufacturer as one utilizes the software while the other maintains it continuously. This is done via software updates to the applications or to the core software itself, the software updates can be in simplistic updates with bug fixes or complex with entire parts of the software changed. When updating software, one thing the developer needs to keep in mind is to ensure the update does not erase the user's data (or apps in case of encompassing software). In all cases, the update cannot be stopped in between as this can cause several issues in the software (even if the stoppage was unintentional).An Emulation of the actual software update procedure can be developed to validate the actual real-time software update without the risk of rendering one or several smart cards inoperable. The next step is to test this procedure for faults, and different use-cases. A simulator is a software that imitates the behaviour of a device or system, it effectively captures the system's operations and performs it independently and, in most cases, without the need of any additional peripherals or software.

A fundamental difference between classic software upgrades and smart card software updates is the applet instances that are installed onto a smart card. These instances belong to several different service providers and thus contain a lot of data (some of which can be sensitive). Deletion of this data, along with the applet during a software update can be seen as unfavourable and multiple software updates to a card, with the requirement of uninstallation of all applets multiple times can be further dissuading to service providers to install their applets onto a smart card. Thus, it is important to both, export or backup all forms of data (whether sensitive or not) not directly related to the smart card itself and to ensure the update or updates take place in a proper manner and the smart card is operable again, in the new version of the software, before the data is imported back. Should the applets also be updated here, then proper care must be taken to ensure that the applets are in their new version first before data is imported back.

The test bench was developed for an simulation of NXP's flagship smart card. NXP has a flagship smart card product that supports various features through the software installed on it. Like all other software, NXP's flagship software needs to be updated carefully to ensure no data is erased and update is properly applied (features are added/removed, and nothing is broken). NXP's smart card product also supports the installation of applications onto it, which are crucial forms of data that must be preserved during and after a software update.

A. Manual Software Update for a smart card on a simulator

The Software Update can also be manually applied to smart card by normally interfacing with it via a terminal card reader to which it is connected, with a host on the other side. With the simulator however, the process is much simpler, the simulator must simply be started and a proprietary software must be used to interact with simulator (one that translates predefined commands to APDUs for the simulator to read and process). Once this is done, the software can be used to interact with

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 10 Issue 05, May-2021

the simulator and send the necessary commands to start the Upgrade Session and the software update process respectively.

Here, the ELF Upgrade Session is manually started via entered commands and the proprietary software is used to translate these commands for each phase of the Upgrade Session to their corresponding MANAGE ELF UPGRADE Command APDUs to send to the simulator. The simplicity and linearity of this procedure allows back and forth communication quickly, however no advanced form of error detection beyond the error status word and abortion of the Upgrade Session will be present. The same applies to the software update with the additional requirement of wiping or resetting the simulator if any failure occurs during updating of system data. This leads to several trial and error runs which must be manually executed. This can cause a huge amount of delay, as the manual procedure is always more time-consuming. Another factor to consider here is that the process is prone to human error, which can invalidate either one of the times the update is executed or all based on the severity of the error. This adds even more delay to the process, while also masking the actual error(s) that may occur while the update is being performed. Furthermore, it is evident that this procedure is not very efficient both in terms of time taken to perform the update and finding and debugging errors. Another thing to keep in mind is that the simulator is wiped or restarted while debugging errors when it stops during a update. While this may be acceptable in a simulator, since this procedure is repeated with the little variation for an actual card, the trail and error may lead to the card becoming unresponsive till it is wiped completely. This is the main disadvantage of manual updation , being that the update cannot be stopped in between, and if it is stopped, say, due to an error, it renders the simulator unresponsive till it is reset. It is also very inefficient and time-consuming and also prone to human errors, which further complicate the testing process. All of these shortcomings pointed to the need of an automated, tested process of Software Update for the smart card.

B. ELF Upgrade Session for Applet Upgrades

An ELF Upgrade Session starts upon receipt of a MANAGE ELF UPGRADE [start] command. Once this command is received, the card will trigger preliminary checks and the Saving Phase [4]. If the ELF is pre-loaded during this Phase, then the WAITING_EXECUTABLE_LOAD_FILE status is skipped in the session. The MANAGE ELF UPGRADE [start] command can and shall be rejected with an error leading to abortion of ELF Upgrade Process if there is an ELF Upgrade Session that is already ongoing or the start command specifies a version number that does not match the version number of the old ELF version. It will also be rejected if there are any static dependencies that are preventing the old ELF version from being deleted or any Application instance created from the old ELF version is currently selected. If the new ELF version is already present on the card and it does not conform to requirements or some application instances have already been created from the new ELF version, then, this also is a cause for rejection. If multiple ELFs must be upgraded within the same ELF Upgrade Session, the session shall start with multiple MANAGE ELF UPGRADE [start] commands, each one specifying an ELF that shall be upgraded [4].

Once the ELF Upgrade Session has been started and till it is completed, no unrelated operations involving applications or ELFs involved in the process may occur, such operations will be instantly rejected. Such operations include selection, installation, or even deletion of applet instances involved in the ELF Upgrade process.

II. Software Update Test Bench

A. Software Requirements for the Software Update Test Bench

As the test bench is entirely based around a simulated version of the smart card. The first requirement is the simulator build itself. Both the older version and newer versions of the simulator are required. Next, it is important to create the skeleton of the test bench so that it can effectively test the software update for multiple use-cases. This is best done using an Integrated Development Environment (IDE). For developing the test bench, Eclipse IDE was used, Eclipse contains a base workspace and an extensible plug-in system for customizing the environment. Eclipse's primary use is for developing Java applications, and since the smart card uses Java Card, an extension of Java, Eclipse's environment for Java programming and testing was useful for quickly setting up the software test bench. To create the tests themselves, as well as the setting up and tearing down of each test case, the Java library JUnit, version 4.11, was utilized. JUnit is a unit testing framework for the Java programming language that can be used to create tests that are modularized and independent whilst sharing the same setup and teardown within a test suite. The test suites can also have their own setup and teardown, executed once per run, responsible for setting up variables or methods or classes required in general for all the cases.

The Eclipse IDE itself is run on Windows and the version of Java Card used in programming is version 3.1. The version of Java used was the Java Development Kit 1.8.

B. Flow of the Software Update in the Test Bench

1) The Main Software Update: Initially, before starting the update, the version number of the smart card's software, the one present on the simulator or card is compared to the one being applied to it. When checking if the version numbers of the old and new software do not match, it is important to ensure that the version being applied is much greater as anything lower constitutes a downgrade. The versions must not match either as this would simply qualify as the reapplication of an update. For the applet upgrade, the new ELF version is pre-loaded onto the simulator. Once the Upgrade Session is finished with the Saving and Loading Phases, the main procedure is started. NXP's smart card requires binary files that contain parameters obtained from the newer version's files (present alongside the simulator build). These files are used to trigger the main update after the ELF Upgrade Session's Saving and Loading Phases. The entire update procedure is

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 10 Issue 05, May-2021

done via methods which themselves consist of the update commands used during the manual update procedure. These methods use variables and other sub-methods, overloaded or otherwise to store the update commands in a readable format, and execute them as such. The methods themselves are wrapped in a JUnit assertion to verify their success or failure, whichever is required. The assertions are applied on multiple levels to all methods and their respective sub-methods and sometimes even variables. These methods when called sequentially during the update, have their code translated into Java Byte Code by the JVM which the simulator, like the smart card it is programmed to simulate, reads and interprets as a set of commands (in the form of APDUs) to perform the Software Update. The commands themselves are proprietary and designed by NXP to perform the software update. The test bench's function is here is to ensure all this takes place automatically and seamlessly when the update is taking place. While all this can be also done manually, the test bench has the ability to modularize this process, by splitting the steps into their separate methods, which can then be asserted for failures at any point. This allows errors to be observed at whatever point they occur and traced back to the root cause. Furthermore, the update will cease at this point, making it much simpler to pinpoint the issue within a method (or methods) to identify which command is causing the update to fail, also since it is being performed on a simulator, there will be no issue in wiping it to reset it to its original state, a problem that can persist in an actual smart card. During each step of the update, the simulator is continuously checked for the updated values of the relevant parameters, as the smart card is a gradual process of updating of its internal domains, system data and applets that are internalized. For the unchanged parameters, the values are checked against the original values of the simulator (stored in separate variables in advance), and the changing parameters (including the version number) are constantly queried to see if they are at the final stage i.e the new version's parameter values. Once the main Software Update is complete, with respect to the data written to the card. The final step is to reupload and reinstall the applets that were initially saved off-card, leading to the resumption of the Upgrade Session.

Any failures will result in the roll back of the entire procedure to ensure that the system is still functional and thus not affecting user experience. The ELF Upgrade Session is stopped. If the update fails during one of the update steps, the newer version will not be reached. In the simulator this stops the update, causing it to become unresponsive to further commands until the older ELF is reapplied, via the Upgrade Recovery procedure and simulator is reset to older version. Fig. 1 shows the flowchart for the software Upgrade Session in a smart card.

2) Upgrading Applets in the Test Bench: Once the version numbers are checked and confirmed to different and the new software greater than the older one,applet upgradation can be started in the test bench for the older version. A 'Get Upgrade Session Status' Command is initially sent to the simulator to ensure that it is in the 'NO_UPGRADE_SESSION' state before proceeding with the applet update. The ELFs used here must also undergo a separate version number comparison to ensure that the minimum version number specified in the start command is the same as that of the older ELF version number, as any difference in the versions would mean that the data from the older instances cannot be restored, which is why the response to this would be an error. Once it is confirmed that the versions do not match and the Upgrade Session Status is the required 'NO_UPGRADE_SESSION' , the Upgrade Session is started by sending the required MANAGE ELF UPGRADE [start] command. Once the command is received by the simulator, the Saving Phase is started, and all applets and their data are backed-up 'offcard', for the simulator this means that the applets are backed-up in the test bench's directories. Once all applets and their data is backed-up, the applets themselves are deleted. Next the Loading Phase of the ELF is started with the new ELF already pre-loaded in the simulator before the start of the Session. The 'Get Upgrade Session Status' Command should indicate 'WAITING_RESTORE' during this phase showing that it is waiting for a command to resume the Upgrade Session. If the status is anything else, then there is a failure that must have occurred during pre-loading of ELF before the Session start and the test bench halts execution (does not proceed even with the main software update). Any cessation of the process in this stage can be traced back to issues with the new ELF or loading of the new ELF. As no applets can be updated until the software update procedure is complete, the Upgrade Session is paused in the test bench. Any errors until this point, after the start command has been sent, can be attributed to inconsistencies with the applets being updated or the upgrade session itself and will cause the test bench to stop the entire software update procedure and allow debugging of the issue. Once the main software update is complete, the ELF Upgrade Session can resume with the aptly named MANAGE ELF UPGRADE[resume] command to transition into the Restore Phase. Once the main software update is complete, the Upgrade Session can continue to finish the updation of applets that were present in the card. In the Restore Phase, the applets and their data which were backed-up offcard are restored. The applets are first reinstalled, then the data which was present in these applets is restored. Finally, consolidation of data required for each applet is performed. The applets, similar to how, during the Saving Phase, Deletion Sequence were uninstalled and deleted from the card, are ,during the Restore Phase, freshly installed with their versions being updated to the applet's version in the new software (only if any newer version exists). The Application IDs (AIDs) are unchanged for these old and new ELFs, and thus verification is only necessary at the end of the update. Any failures in the Restore Phase halts the test bench. The failures can occur during any of the three sequences which might indicate a fault in the parameters if it occurs during re-installation applets, or a impromptu change in the data if it fails during restore or consolidation sequences. The Upgrade Recovery procedure is started during teardown, if any Upgrade Session related failure

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 10 Issue 05, May-2021

occurs and is the reason for stopping the testbench, before starting the rollback and reset of the simulator.

C. Verification of Software and Applet Updates

While the main Software Update can be verified by checking the version numbers, it is important to ensure the application data on the card is also unaffected during and after the update. To ensure application data is unmodified and also does not affect the upgrade procedure, the applets are backed-up and saved off-card before the update, as mentioned above, and deleted from the card. However, after the update it is important to ensure that the applets are, first, in the newer version (should one exist) and also, more importantly, during the last two sequences of the Restore Phase, i.e the Restore Sequence and the Consolidation Sequence, the applet is able to restore and retain all of the old data present on it before the update. To test these requirements, two test applets were added to the ELFs, (i.e the applets were installed 'onto' the simulator and were programmed to implement the Upgrade API). One of the applets, say applet 'A', had a newer version (with a new AID) implemented in the new ELF version while the other had the same AID as before. However, the other applet, called applet 'B', it was programmed to hold a known default value. In the test bench, before the start of the Upgrade Session, the second applet, applet 'B', was installed as a contactless applet which had its data changed to another known value, and this modified value was recorded in a variable. Now, during Data Saving Sequence in the Saving Phase it was expected that the value to be saved would be the modified one. The first applet, applet 'A', (installed without the contactless property) which had two different versions of its AID but no change to any of its data, had only its old AID recorded beforehand. After the Upgrade Session was complete, the applets were separately checked for their respective properties, the AID or the data, which were expected to either change (for applet 'A') or remain the same (for applet 'B'). For applet 'A', it was expected that the applet would be installed with the new AID version as a part of the new ELF, the AID was once again recorded and compared to ensure that it did not match that of the old ELF's applet AID. For applet 'B', the data in the applet was expected to be the changed value made before the update started, as this was what was saved and thus the same was expected to be restored to the applet as well. Should the applet's data have been the default known value, then it would have shown that the last two sequences of the Restore Phase had failed as the applet was installed with default values from the new ELF version in the Installation Sequence. The value of the applet's data is compared to the recorded value (the modified value) and an equality suggests that the applet is been imported unmodified as expected from the Upgrade Session. Both applets were selected, one using the regular interface and one using 'wireless' interface to ensure that they were functioning normally. This case seeks to validate the entire Upgrade Session by adding a custom input to the defined process with different requirements as expectations and a success in meeting the requirements further shows the effectiveness of the implementation of a Software Update for a smart card in a test bench.

III. Conclusions

In this paper, an approach to integrate testing and automation of the Smart Card Software Update has been proposed and implemented. The procedure is an alternative to the manual, time-consuming update procedure for a Smart Card Software Update prone at anytime to human error. The Test Bench can perform the entire software update and applet upgrade procedure automatically when run, ensuring there are no interruptions in the update procedure with the exception of errors, which is what the test bench is designed to find. The modularization of the process alongside the IDE environment allows simpler error debugging with steps split into methods allowing for proper tracing. The whole process is therefore far more efficient, allowing for quicker updating when there are no errors and more importantly, faster debugging when errors are present.

References

[1] Global Platform Card Specification, v2.3.0, GlobalPlatform, 2015,pp 1-237
[2] GlobalPlatformCard Technology Contactless Services Card Specification Amendment C, Version 1.2, GlobalPlatform, 2015, pp 1-128
[3] Smart Card Handbook, 4th. Ed, Wiley Publishing,2010, pp 1-1043.
[4] GlobalPlatform Technology Executable Load File Upgrade Card Specification v2.3 Amendment H, Version 1.1, GlobalPlatform, 2018, pp 1-52
[5] GlobalPlatformCard Technology Contactless Services Card Specification Amendment C, Version 1.2, GlobalPlatform, 2015, pp 1-128
[6] X. Wu, Y. Chen and S. Li, "Contactless Smart Card Experiments in a Cybersecurity Course," presented at 2018 IEEE Frontiers in Education Conference (FIE), San Jose, CA, USA, 2018, pp.1-4
[7] Gupta, Brij B. and Shaifali Narayan. "A Survey on Contactless Smart Cards and Payment System: Technologies, Policies, Attacks and Countermeasures." Journal of Global Information Management vol28, no.4, 2020, pp 135-159
[8] N. Jesani, N. Gupta, S. Bhatt, P. Singh and A. Saxena, "Smart Card For Various Application In Institution," 2020 IEEE International Students' Conference on Electrical,Electronics and Computer Science (SCEECS), Bhopal, India, 2020, pp. 1-5
[9] T. Adiono, A. Alfaruq and S. Fuada, "Hardware Design of Transaction Device based on Contact and Contactless Smart Card," 2019 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS), Bangkok, Thailand, 2019, pp. 277-280
[10] R. K. Pal, "Authenticated Encryption Schemes on Java Card," 2019 22nd Euromicro Conference on Digital System Design (DSD), Kallithea, Greece, 2019, pp. 238-245
[11] F. Mancini, "Using Smart Cards to Enhance Security of Android Smartphones in Tactical Scenarios," 2018 IEEE International Conference on Communications Workshops (ICC Workshops), Kansas City, MO, 2018, pp. 1-6.
[12] S. mohammed, S. KURNAZ and A. H. Mohammed, "Secure Pin Authentication in Java Smart Card Using Honey Encryption," 2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA), 2020, pp. 1-4

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 10 Issue 05, May-2021

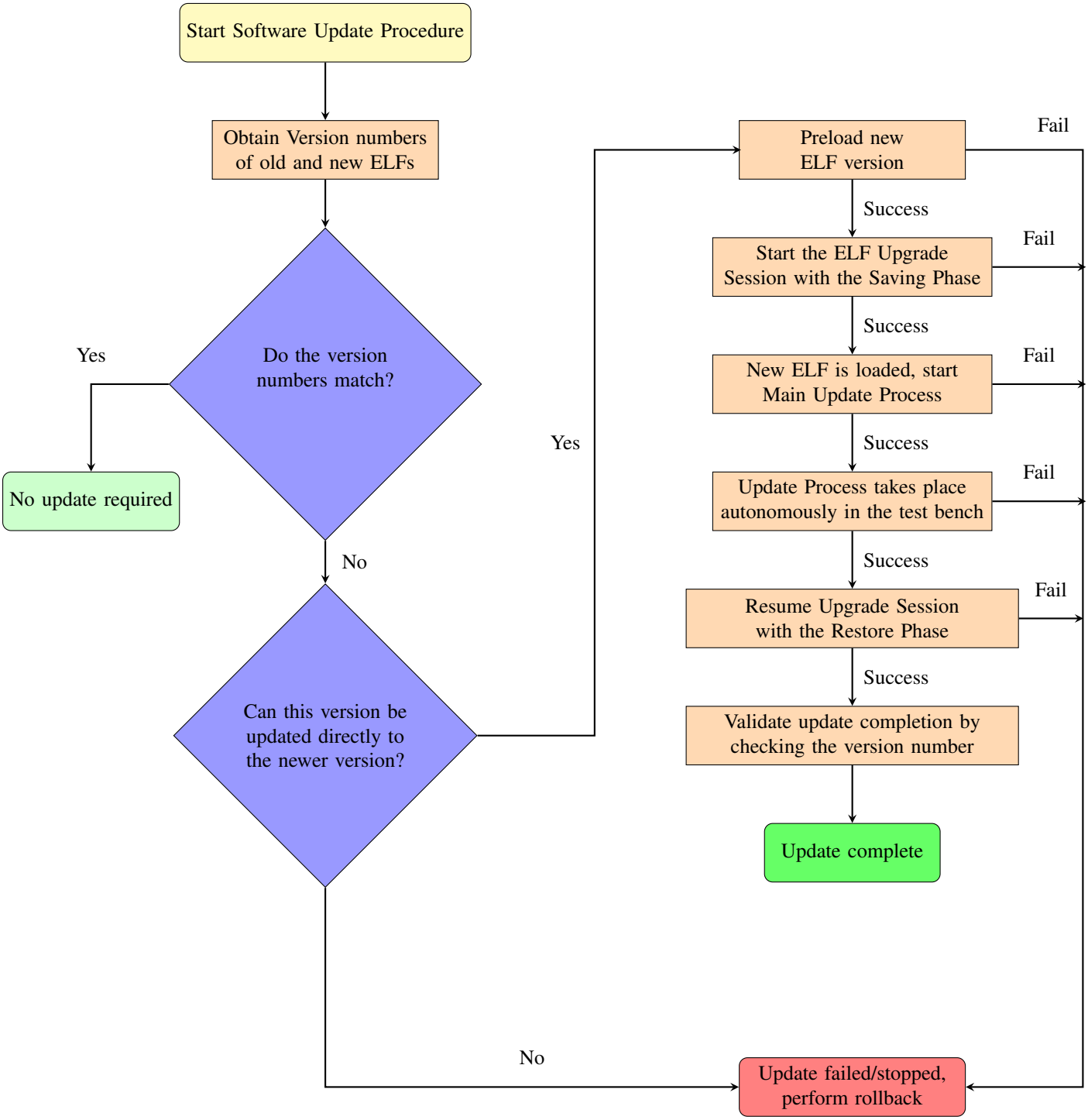

Fig. 1: Flowchart of the Software Update in a test bench