Software requirements specification for project “STM32-bootloader”

1. Authors

- Bortnikov Pavel
- Tarasov Artem
- Vasilev Pavel
- Patrushev Boris

2. Introduction

The main idea of this project is to develop a bootloader and a high-level bootloader utility program for the STM32, where the end user can choose a file with a new version of the program and then send it to the STM32. Then STM32 receives the program and stores it in the flash memory. For security reasons the transferred program is encrypted (it will be decrypted by the bootloader on the STM32 chip). Also the high-level program will be able to upload the bootloader to the device (for the updates to devices that don’t have it yet). In other words, the end user will have to just drag and drop the bootloader/program file to the utility program on their PC, choose the upload options.

3. Glossary

Bootloader - the program on the STM32 microcontroller that can download or upload data (programs or any other data) and that transfers the control to the firmware.

Developer’s server - Developer’s server that interacts with the flasher.

Device - an embedded system that has an STM32 chip to run it’s main program.

Device’s firmware - the main program on the device that’s supposed to run on it.

Device’s ID - an identification number/code that developer has provided to the end users.

Flasher - the utility program on the end user’s PC.

Flash Drive - a storage device that is easily available for the end user and the developer (e.g. a thumb drive).

Project - developer’s firmware project.

Pre-build script - script that adjusts the project so it aligns with the proposed memory layout.

4. Actors

4.1. **Name:** End user
Goal: Having the up-to-date authentic device program
Responsibilities: Literally follow instructions and click the buttons on the PC

4.2. **Name:** Device developer
Goal: Get the updates to the end user and protect their own intellectual property (namely the program)
Responsibilities: they have to provide the end user with the device’s firmware (and its updates) and the flasher.

4.3. **Name:** Flash Drive
Goal: nope
Responsibilities: Transmit updates to the flasher and receive data from it

4.4. **Name:** Developer's server
Goal: Ensure the correct operation of the server that was created for distribution of firmware updates
Responsibilities: Transmit updates to the flasher and receive data from it

4.5. **Name:** Project
Goal:
Responsibilities:

4.6. **Name:** End user without the bootloader (EUWOBL)
Goal: Install the bootloader and be up-to-date
Responsibilities: Register his device on the developer website, get the flasher and firmware, and install it. Bootloader will be installed automatically via flasher.

5. Functional requirements

5.1. Strategic Use-cases

5.1.1. Use-case "Reduce personnel requirements"

Actors: End user(s), devices' developers

Goal: Reduce requirements to the end users and device developer there will be no need for a programmer and special skills for doing firmware upgrade

Main success scenario:

1. device developer installs pre-build script
2. Connects their program to the pre-build script
3. Sends the final program to the user
4. Users updates their device (see the "Firmware update" actor's case)

5.1.2. Use-case "Confidential operating principle of firmware"

Actors: Device developer

Goal: Not to reveal how the firmware works

Main success scenario:

Nobody can read the code starting from the build process to the installation process on the device

5.1.3. Use-case "Save device's correctness"

Actors: Device developer

Goal: Data can not be corrupted accidentally during transmission.

Main success scenario:

No one can accidentally remove some data from the device.

5.2. Use-cases for End user

5.2.1. Use-case “Firmware update”

Actors: End user, device developer

Goal: To update the device’s firmware

Precondition: End user must have a PC and flasher installed on it

Trigger condition: Outdated device firmware

Extensions:

Main success scenario:

1. Device developer notifies end users that there is a new version of the device's firmware
2. End user must download the device’s firmware from the internet or “choose the file on the PC”.
3. Then they open the flasher and choose the downloaded file (if he downloads it from the internet).
4. Also they need to connect the device to the computer.
5. After all they need to click the button “upload” and wait for the firmware to be uploaded to the device.

5.3. Use-cases for Device developer

5.3.1. Use-case “Upload firmware to developers”

Actors: End user, device developer
Goal: Get the program data
Precondition/Context: End user must have a PC and the flasher installed on it
Trigger condition: Request from developer to end user
Extensions:
Main success scenario:

1. Device developer asks user to send program
2. End user connects device to PC
3. Opens the flasher
4. Presses the button “upload program to developer” or “save program data”.
5. Waits until program will be sent successfully

Alternative scenario Internet connection error (if user uploads by internet)**:**

End user must fix the internet connection and try to send again.

Alternative scenario Connection with device error**:**

Contact developers, report them error code and get additional information.

5.3.2. Use-case “Modificate the build”

Actors: Device developer, Project
Goal: Make it so that that firmware built from the project could be used with the flasher and the bootloader.
Precondition/Context: Developer wants to upload his project on the device, but he needs the memory layout etc, without which he can’t upload the project.
Extensions:

Main success scenario:

1. Device developer enters the parameters in the modificator.
2. The modificator returns the modified project files which can be passed to the building process.
   //will be refactored

5.3.3. *Use-case* "Get memory layout"

Actors: Device developer

Goal: Check the memory layout with other memory

Precondition/Context: Device's firmware

Trigger condition: Necessary to get layout (for example, change it)

Extensions:

Main success scenario:

1. Developer uses a pre-build script: he runs it with a special flag that means that he wants to get the memory layout of the current firmware. Also he needs to pass firmware as arguments.
2. Get the layout

6. System-wide functional requirements

none?

7. Non-functional requirements

7.1. Environment

Any STM32 chip with any interaction interface (e.g. UART or I2C). The interaction interface is specified by the developer.

Windows 10 on PC.

7.2. Performance

The bootloader transfers control to the firmware in a short amount of time.

The bootloader takes up a small amount of flash memory on the chip.

7.3. Reliability

- If the program is not completed, then we restore the previous version of the firmware and execute the restored version
- bootloader will control checksums to be confident that program was downloaded successfully

7.4. Extensibility

7.5. Other

1. Data format that contains the device name, ID, and other data to adjust the installation process (for example, not to override the program if it has an up-to-date version of firmware)