Software Requirements Specification

for

Custom Bootloader

Version 0.0

Prepared by Nguyen The Anh

07/19/2024

Table of Contents

1. Introduction

1.1 Purpose

The purpose of this Software Requirements Specification (SRS) is to outline the functional and non-functional requirements, detailed description of the Custom Bootloader.

1.2 Product Scope

This document specifies requirements for Custom Bootloader application.

The Bootloader can allow users to:

- Receive and send data through UART terminal.
- Erase flash memory.
- Re-flash the application.
- Communication with user through the UART terminal.

The Bootloader runs offline by manual.

Note: The Bootloader functionality corresponds to V0.0 released in 2024.

2. Overall Description

2.1 Product Perspective

This product works on MKL46Z4 MCU and MKL46Z sub family. The Bootloader is a new application for updating new firmware without the need of using debugger by receiving data from user through the UART terminal.

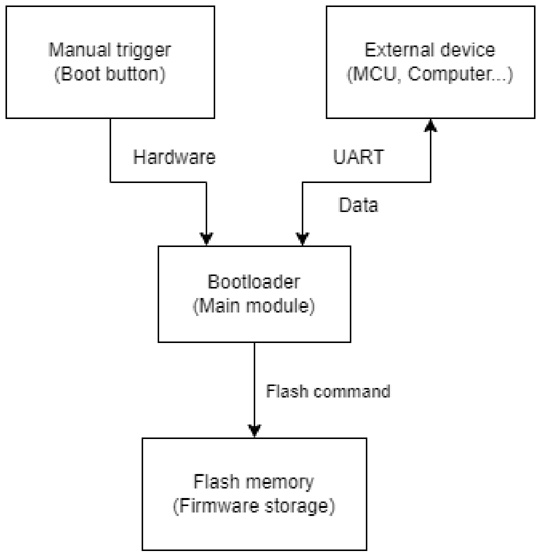

Figure 2.1: Overall system

2.2 Product Functions

The section describes the major functionalities and operations that the product will provide to its users. More detailed information for each function will be provided in section 3 & 4.

Major functionalities:

- Transfer data by UART protocol in the baud rate of 115200.
- Handle S-record file data.
- Checking Flash memory location for new firmware.
- Erase flash memory.
- Program the flash memory.
- Re-flash the application.
- Communication with user through the UART terminal.

2.3 Operating Environment

The Operating Environment for the Bootloader system is structured as follows:

- **Hardware platform:**
  - The Bootloader operates on MKL46Z4 microcontroller with 256KB flash memory and 32KB in RAM.

  - Interfaces with external devices via UART and GPIO for bootloader operation and firmware updates.

- **Software environment:**
  - **Bootloader firmware**:
    - + Developed in C language using Kinetis Design Studio IDE.
    - + Managing firmware updates and system boots processes.
  - **Application firmware**:
    - + Developed in C language using the same development environment.
    - + Provide application specific functionalities interacting with the bootloader for firmware updates.
- **Dependencies:**
  - **Development tools:**
    - + Require ARM Kinetis Design Studio IDE, ARM GCC compiler, and J-Link debugger for Bootloader development.
  - **Third-Party libraries:**
    - + Integrates with MKL46Z4 HAL library for microcontroller peripheral management.

2.4 Design and Implementation Constraints

- **Hardware Compatibility:** The custom bootloader must support a range of ARM Cortex-M microcontrollers, include MKL46Z4 and MKL46Z sub family. Compatibility testing will ensure proper operation with varying memory sizes (from 128KB to 256KB), different clock frequencies (up to 48MHz), and can communicate through peripheral UART.
- **Boot Time Optimization:** Target boot time should not exceed 500 milliseconds from reset to application launch under typical operating conditions. Performance optimizations will focus on reducing initialization overhead and optimizing flash memory access for faster execution.
- **Error Handling**: The Bootloader must include error detection to handle data and communication failures.
- **Compliance and Standards**: The bootloader will comply with industry standards such as the ARM Cortex Microcontroller Software Interface Standard (CMSIS) and relevant firmware update protocols (Motorola S-Record).

2.5 User Documentation

- KL46 Sub-Family Reference Manual.
- Cortex – M0+ Devices: Generic User Guide.

2.6 User Characteristics

This product is used for all classes of user. The users have all privilege levels to the product.External Interfaces and Requirements

2.7 Hardware Interfaces

- **Microcontroller interface:**
  - **Description**: The custom bootloader interfaces directly with the microcontroller's hardware components, including memory (Flash and RAM), GPIOs, and communication peripheral UART
  - **Specifications**:
    - **UART:** Baud rate is at 115200, 8-bit data length, no parity bit, 1 stop bit.
    - **GPIO:**
      - PORT A: PTA1 – UART0_Rx pin, PTA2 – UART0_Tx pin.
      - PORT C: PTC12 – Switch 2 button pin.
      - PORT E: PTE29 – Red LED pin.
- **Reset and Boot Mode Button**:
  - **Description:** The Application will run in default. To enter boot mode, users have to hold the switch 2 then press reset button. After the boot, user need to press reset again to launch the new updated application.
  - **Specifications:** The switch 2 uses pull-up resistor so it has logic 1 in normal. When the switch is pressed, the logic will change from 1 to 0, causes the pin in port data input register to become 1. When user hold switch 2 then press reset, the application will reset and catch the boot button flag, then it will jump into the boot process to update new firmware.

2.8 Software Interfaces

- **Firmware update protocol:**
  - **Description**: The Bootloader communicates with external devices (Microcontroller, PC, UART terminal…) by UART protocol.
  - **Specifications**: The Bootloader will send requests and receive data from external devices by UART that has the following configuration:
    - **Speed:** Baud rate is at 115200.
    - **Data length:** 8 bits.
    - **Parity checking:** No.
    - **Stop bit count:** 1 stop bit.
    - **Start bit level logic:** 0.
    - **Stop bit logic level:** 1.
    - **UART clock frequency:** Up to 48MHz.
- **Configuration File Interface**:
  - **Description:** The Bootloader will receive file through the UART terminal or from external devices when in boot mode.

- **Specifications:**
    - **File format**: S-record file.
    - **Flash image**: Flash image of the application starting at address 0xA000 in flash memory.

3. System Requirements

The custom bootloader shall be designed for the MKL46Z4 microcontroller board and can be extended to the MKL46 sub family. It should provide the primary function of reliable firmware updates for embedded systems based on the MKL46Z4 microcontroller.

3.1 Functional Requirements

3.1.1 Initialization and Configuration:

Description: The Bootloader must initialize all required hardware peripherals automatically upon system startup, and configure the UART protocol.

Detailed process: When the Bootloader starts up, it should follow a structured sequence of steps below.

- Initialize the clock gating in the SIM (System Integration Module) module.
- Configure UART communication on UART0 module.
- Set the clock frequency for UART communication in the MCG (Multipurpose Clock Generator) module.
- Configure GPIO pins for components such as LEDs and switch buttons.

3.1.2 Communication Interface:

Description: The bootloader must support bidirectional communication via UART for firmware update operations.

Detailed requirement:

- The bootloader communicates with the user through the UART terminal, sending instructions and requesting actions.
- Flash images in the SREC file format are received by the bootloader via UART.
- In the event of a failed firmware update, the bootloader notifies the user of the failure state.

3.1.3 Data Handling:

Description: The bootloader must effectively manage and process data to ensure reliable firmware updates. This section details the requirements for receiving, validating, storing, and handling data throughout the firmware update process.

Detailed requirement:

- Data Reception and Parsing:
    - Shall initiate the reception of firmware images in SREC file format via UART communication.

- shall parse incoming data streams to extract firmware image data.

- Data Validation and Verification:
    - Upon reception, the bootloader shall verify the integrity of received firmware images using checksum and CRC (Cyclic Redundancy Check) mechanisms
- Storage and Management:
    - Validated firmware images shall be stored securely in designated flash memory sectors by the bootloader.
- Error Handling:
    - The bootloader shall detect and appropriately handle communication errors (e.g., UART errors, checksum failures) during data reception and processing.
    - Notify the user that the update process has failed.
- Performance Optimization:
    - The bootloader shall optimize data handling processes to minimize processing time and resource utilization, thereby enhancing overall system performance.
    - Critical data handling operations shall prioritize real-time responsiveness to meet system requirements and ensure operational reliability.

3.1.4 Boot Process:

Description: The bootloader must automatically initiate the boot process upon system reset when the user selects Boot mode, providing feedback on the boot progress through status indicators such as LEDs or serial console messages.

Detailed requirement:

- The application firmware will operate in default mode, and the user can access boot mode only by performing a specific hardware action.
- To enter Boot mode, the user must hold the boot button and then restart the system.
- A Red LED indicator will illuminate during the boot process.
- Ensure status updates and messages are communicated between each stage of the boot process (e.g., erase, program flash completion).
- Prompt the user to restart the system upon successful completion of the update.
- The Bootloader will initiate erasing the old application code in the boot process.
- The Bootloader will program the flash memory with the new updated firmware image.

3.2 Non-Functional Requirements

3.2.1 Performance:

- The Bootloader shall complete firmware updates within a maximum of 1 second after finishing the reception of the flash image.

- The bootloader shall handle firmware images up to 256 KB in size efficiently.

3.2.2 Reliability:

- The bootloader shall have a mean time between failures (MTBF) of at least 1000 hours.
- It shall ensure a firmware update success rate of 99% under normal operating conditions.

3.2.3 Usability:

- The bootloader shall provide clear and concise user feedback through status indicators (e.g., LEDs, serial console messages) during and after firmware update processes.
- It shall include a user-friendly interface for initiating firmware updates.

3.2.4 Scalability:

- The bootloader architecture shall support future enhancements and extensions without major redesign.

3.2.5 Compatibility:

- The bootloader shall be compatible with the MKL46Z4 microcontroller family and associated hardware peripherals.
- It shall support firmware updates using the S-record file format.

4. System Scenarios

4.1 Use-case Diagram

4.1.1 Firmware Update Process:

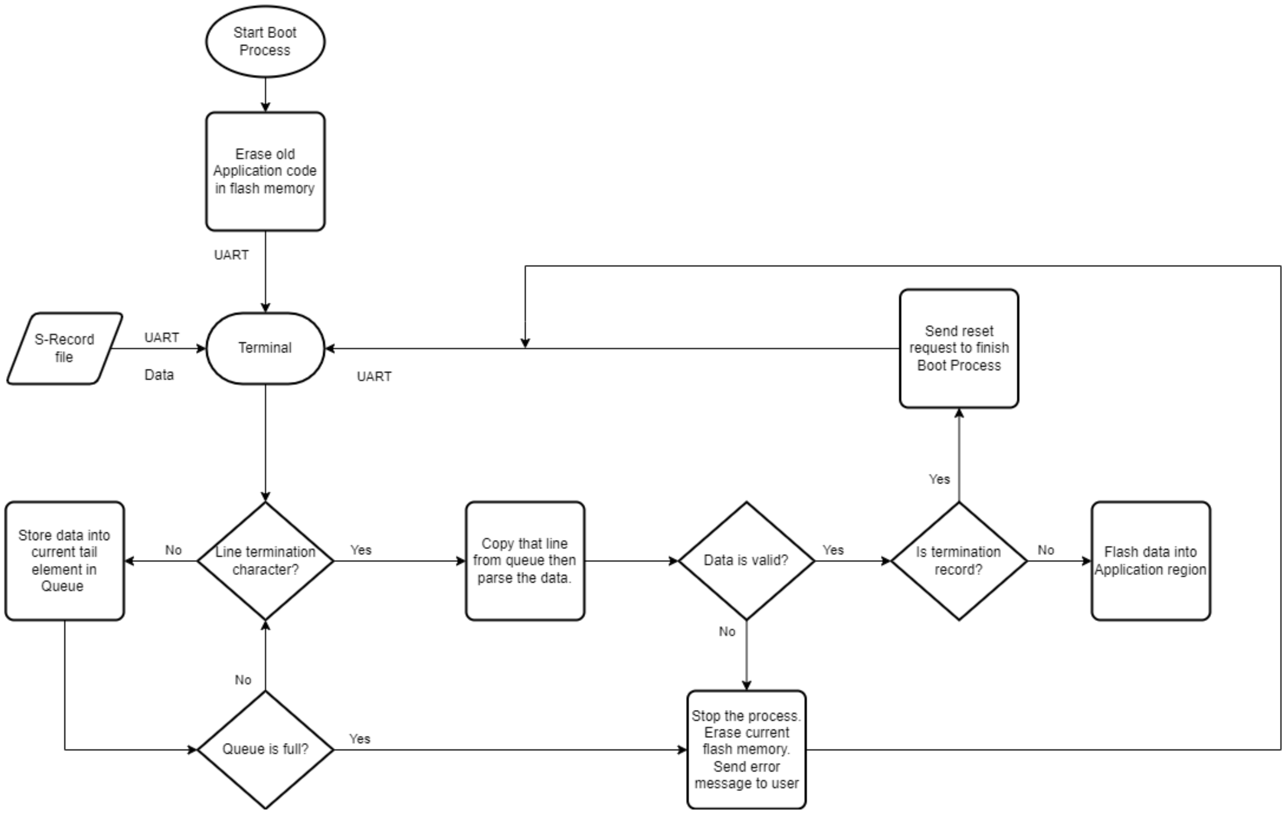
Start Boot Process
Erase old Application code in flash memory
UART
S-Record file
UART
Data
Terminal
UART
Send reset request to finish Boot Process
Yes
Store data into current tail element in Queue
No
Line termination character?
Yes
Copy that line from queue then parse the data.
Data is valid?
Yes
Is termination record?
No
Flash data into Application region
No
No
Queue is full?
Yes
Stop the process. Erase current flash memory. Send error message to user

Figure 5.1: Boot Process Diagram

4.1.2 Boot Mode Selection:

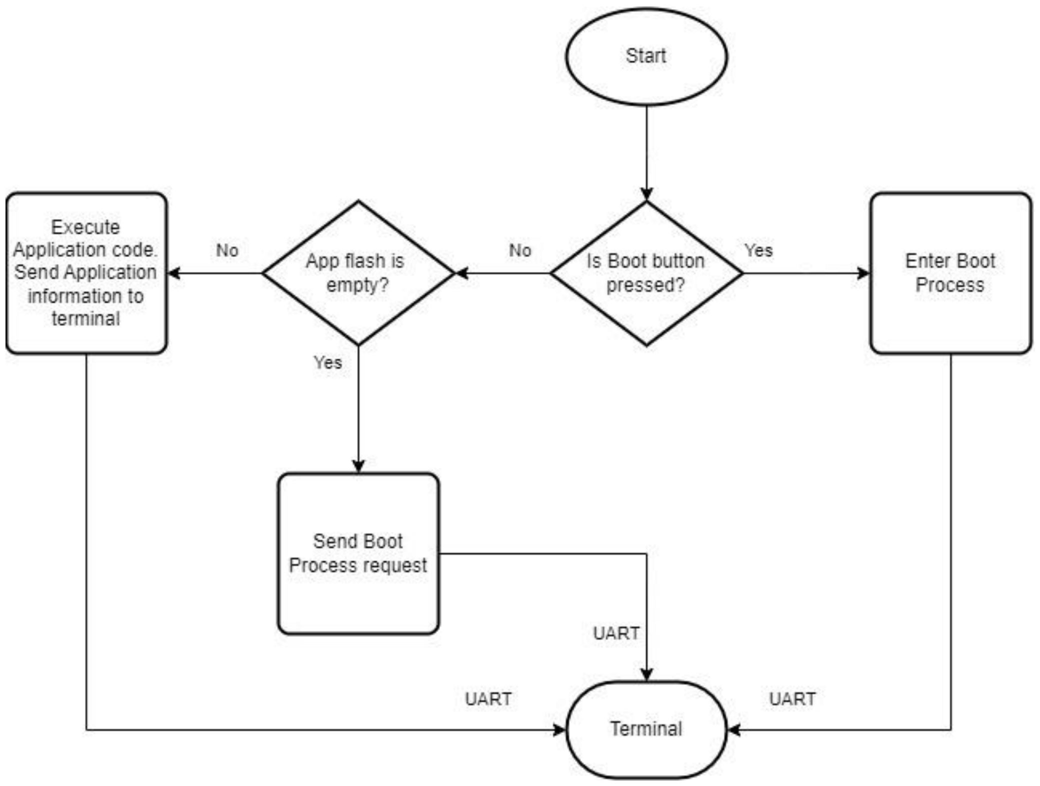

Figure 5.2: Boot Mode Selection diagram

4.2 Scenarios

4.2.1 SCN-1: Firmware Update Process

Scenario Description: Initiating and completing a firmware update using the bootloader.

- **Actor:** User.
- **Preconditions:**
  - The MKL46Z4 board is powered and connected to a host system.
  - The Bootloader is operational and waiting for instructions.
- **Trigger:** User send an S-Record format file through terminal or another microcontroller.
- **Main flow:**
  - The Bootloader will erase the old application flash memory to ready for flashing new firmware.
  - User transfer the new firmware flash image by uploading an S-Record format file.
  - The Bootloader receives and verifies the integrity of the firmware image.
  - Upon successful verification, the Bootloader programs new firmware into the App region in flash memory (Start at address 0xA000).

- Status indicators such as LEDs or serial console messages provide feedback on the progress of each stage (erase, programming…).

- **Postconditions:**
  - The firmware update is successfully completed.
  - The system automatically initiates the new firmware execution upon reset.
- **Alternate Flow:**
  - If the firmware image fails the verification, the Bootloader will not proceed the update and notify the user as well as erase the current written flash memory.

4.2.2 SCN-2: Boot Mode Selection

Scenario Description: Entering Boot mode or run the application firmware.

- **Actor:** User.
- **Preconditions:**
  - The MKL46Z4 board is powered and connected to a host system.
  - The system is in normal operating mode.
- **Trigger:** User performs a specific hardware action. (Hold down boot button then press reset.)
- **Main flow:**
  - User hold down the boot button then press reset.
  - The Bootloader detects boot mode request and initializes the boot mode.
  - Boot mode will request user to send S-Record file to update new firmware.
- **Postconditions:**
  - The system remains in Boot mode until the user completes the desired action or reset the system manually.
- **Alternate Flow:**
  - If the user do not trigger the boot mode, the application will be execution in default system operation.