МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»

Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой ЭВМ
_____ Д.В. Топольский
«___»_____ 2023 г.

Автоматическое обновление программного обеспечения устройств на базе Интернета вещей

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУРГУ-090301.2023.057 ПЗ ВКР

Руководитель работы,
к.пед.н., доцент каф. ЭВМ
_____Ю.Г. Плаксина
«___»_____2023 г.

Автор работы,
студент группы КЭ-406
_____Ю.А. Сендюков
«___»_____2023 г.

Нормоконтролёр,
ст. преп. каф. ЭВМ
_____ С.В. Сяськов
«___»_____2023 г.

Челябинск-2023

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

УТВЕРЖДАЮ
Заведующий кафедрой ЭВМ
_____ Д.В. Топольский
«___»_____2023 г.

ЗАДАНИЕ

на выпускную квалификационную работу бакалавра

студенту группы КЭ-406
Сендюкову Юрию Александровичу,
обучающемуся по направлению
09.03.01 «Информатика и вычислительная техника»

1. Тема работы: «Автоматическое обновление программного обеспечения устройств на базе Интернета вещей» утверждена приказом по университету от «25» апреля 2023 г. № 753-13/12

2. Срок сдачи студентом законченной работы: 1 июня 2023 г.

3. Исходные данные к работе:

– проект должен быть разработан на языке программирования C;

– микроконтроллер ESP32.

4. Перечень подлежащих разработке вопросов:

1. Обзор и анализ аналогичных проектов беспроводного обновления;
2. Выбор инструментария для разработки программной части проекта;
3. Проектирование архитектуры приложения;
4. Программная реализация проекта;
5. Проверка работоспособности разработанного проекта в реальных условиях.

5. Дата выдачи задания: 2 декабря 2022 г.

Руководитель работы _____/*Ю.Г. Плаксина* /

Студент _____/*Ю.А. Сендюков* /

КАЛЕНДАРНЫЙ ПЛАН

Этап	Срок сдачи	Подпись руководителя
Обзор и анализ аналогичных проектов беспроводного обновления	10.03.2023	
Выбор инструментария для разработки программной части проекта	10.04.2023	
Проектирование архитектуры приложения	21.04.2023	
Программная реализация проекта	04.05.2023	
Проверка работоспособности разработанного проекта в реальных условиях.	20.05.2023	
Компоновка текста работы и сдача на нормоконтроль	30.05.2023	
Подготовка презентации и доклада	01.06.2023	

Руководитель работы _____/*Ю.Г. Плаксина*/

Студент _____/*Ю.А. Сендюков* /

АННОТАЦИЯ

Ю. А.Сендюков Автоматическое обновление программного обеспечения устройств на базе Интернета вещей. – Челябинск: ФГАОУ ВО «ЮУрГУ (НИУ), ВШ ЭКН; 2023, 41 с., 7 ил., библиогр. список – 9 наим.

В рамках выпускной квалификационной работы производится обзор и анализ современных проектов беспроводного обновления устройств на базе Интернета вещей.

В ходе выбора инструментария были выбраны подходящие для выполнения поставленной задачи программные продукты и технологии.

В ходе проектирования архитектуры приложения был разобран процесс запуска микроконтроллера ESP32, а также сформулированы требования к каждому компоненту разрабатываемого программного обеспечения.

В ходе реализации был описан процесс разработки каждого компонента разрабатываемого программного обеспечения, а также были приведены функции фреймворка ESP-IDF использованные в разработке.

В ходе проверки работоспособности разработанного проекта в реальных условиях были протестированы функции разработанного программного обеспечения.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Интернет вещей – это система взаимосвязанных вычислительных устройств, способных передавать и собирать данные по беспроводной сети. Это направление очень активно развивается [1].

Неотъемлемой частью Интернета вещей являются микроконтроллеры. Микроконтроллер – это специальная микросхема, которая предназначена для управления различными приборами и электронными устройствами; позволяет не только контролировать их работу, но и обеспечивать взаимодействия между ними согласно заложенной программе [2].

Задача автоматического обновления программного обеспечения по беспроводной сети является значимой для Интернета вещей, так как не всегда есть возможность отправить технического специалиста для выполнения данной задачи. К тому же отправлять специалистов обновлять каждое устройство крайне нецелесообразно.

Микроконтроллеры являются неотъемлемой частью Интернета вещей, так как с их помощью осуществляется управление устройствами Интернета вещей. Поэтому выбор правильного микроконтроллера очень важен.

Компания Espressif в данный момент является лидером на рынке микроконтроллеров с интегрированными контроллерами Wi-Fi. Также компания Espressif разработала и поддерживает фреймворк ESP-IDF для разработки программного обеспечения для микроконтроллеров семейства ESP32, который значительно облегчает разработку программного обеспечения для микроконтроллеров семейства ESP32. Поэтому в данной работе программное обеспечение будет разрабатываться для микроконтроллера ESP32.

В данной работе используются следующие методы исследования:

- аналитический метод;
- метод моделирования;
- сравнительный метод.

Для разработки программного компонента для обновления программного обеспечения необходимо решить следующие вопросы:

– обзор и анализ аналогичных проектов беспроводного обновления;

– выбор инструментария для разработки программной части проекта;

– проектирование архитектуры приложения;

– программная реализация проекта;

– проверка работоспособности разработанного проекта в реальных условиях.

1 ОБЗОР И АНАЛИЗ АНАЛОГИЧНЫХ ПРОЕКТОВ БЕСПРОВОДНОГО ОБНОВЛЕНИЯ

Для анализа аналогичных проектов беспроводного обновления нужно определить критерии сравнения.

Критерии:

1) Должна быть возможность обновлять программное обеспечение с помощью WAN сети, чтобы иметь возможность скачивать новую версию программного обеспечения с сервера.

2) Должна быть возможность вернуться к заводским настройкам, если обновление произошло некорректно, чтобы повысить отказоустойчивость системы.

3) Должна быть возможность отслеживать количество обновившихся устройств – телеметрия.

4) Проверка программного обеспечения на целостность, чтобы иметь возможность вернуться к старой версии программного обеспечения в случае скачивания некорректного образа программного обеспечения.

5) Обновление должно проходить по расписанию или по запросу пользователя.

6) Новая версия программного обеспечения должна шифроваться на устройстве, чтобы защититься от утечки данных.

7) Должна быть возможность работать с двумя версиями программного обеспечения, чтобы повысить отказоустойчивость системы.

На сегодняшний день удалось найти в открытом доступе следующие проекты позволяющие автоматически обновлять программное обеспечение.. Для определения достоинств и недостатков рассмотрим следующие проекты:

1) CloudUpdate_Demo за авторством MoonFox2006 [3].

Проект позволяет обновлять программного обеспечения ESP8266 с использованием стандартных облачных сервисов Google.

Особенности:

– передача данных осуществляется через протокол HTTP;

– обновление программного обеспечения происходит через облачные сервисы Google;

– проверка на необходимость обновления осуществляется через сравнивание md5 программного обеспечения на устройстве и программного обеспечения на облаке;

– при необходимости обновления новая программное обеспечение скачивается и после перезагрузки устройства заменяет старую.

Недостатки:

– отсутствие шифрования при передаче данных;

– отсутствие шифрования прошивки на устройстве;

– отсутствие возможности удалённого контроля устройства;

– отсутствие возможности обновления по расписанию;

– отсутствие возможности отслеживать количество обновлённых устройств;

– отсутствие возможности работы в режиме LAN.

2) 30_https_ota за авторством lucadentella [4].

Проект обновления программного обеспечения ESP32 на платформе ESP-IDF.

Особенности:

– передача данных осуществляется через протокол HTTPS;

– проверка на необходимость обновления осуществляется через сравнение версии приложения и версии указанной в json-файле, который хранится на сервере.

Недостатки:

– отсутствие шифрования программного обеспечения на устройстве;

– отсутствие возможности удалённого контроля устройства;

– отсутствие возможности обновления по расписанию;

– отсутствие возможности отслеживать количество обновлённых устройств;

– отсутствие возможности работы в режиме LAN.

3) esp_ghota за авторством Fishwaldo [5].

Проект обновления программного обеспечения ESP32 через GitHub на платформе ESP-IDF.

Особенности:

– передача данных осуществляется через протокол HTTPS;

– для проверки на наличие новой версии используется SemVer.

– возможность настроить интервалы проверки обновлений, или вызвать обновление вручную;

– проверка на необходимость обновления осуществляется через сравнение версии приложения и версии указанной в json-файле, который хранится на сервере.

Недостатки:

– отсутствие шифрования программного обеспечения на устройстве;

– отсутствие возможности удалённого контроля устройства;

– отсутствие возможности отслеживать количество обновлённых устройств;

– отсутствие возможности работы в режиме LAN.

4) 08-LCD SimpleOTA за авторством mostafahk [6].

Проект обновления программного обеспечения ESP32 с использованием сервисов OTAdrive на платформе Arduino.

Особенности:

– передача данных осуществляется через протокол HTTP;

– обновление программного обеспечения программного обеспечения происходит через сервисы OTAdrive;

– контроль версий осуществляется через сервисы OTAdrive;

– проверка на необходимость обновления осуществляется через сравнение версии приложения и версии указанной в json-файле, который хранится на сервере.

Недостатки:

– отсутствие шифрования при передаче данных;

– отсутствие шифрования программного обеспечения на устройстве;

– отсутствие возможности удалённого контроля устройства;

– отсутствие возможности обновления по расписанию;

– отсутствие возможности отслеживать количество обновлённых устройств;

– отсутствие возможности работы в режиме LAN.

5) advanced_https_ota за авторством Esspressif [7].

Проект обновления программного обеспечения ESP32 на платформе ESP-IDF.

Особенности:

– передача данных осуществляется через протокол HTTPS;

– возможность работы с тремя разделами для приложений, эта возможность позволяет вернуться к старой версии программного обеспечения, если новая версия оказалась неисправной.

Недостатки:

– отсутствие шифрования программного обеспечения на устройстве;

– отсутствие возможности удалённого контроля устройства;

– отсутствие возможности обновления по расписанию;

– отсутствие возможности отслеживать количество обновлённых устройств;

– отсутствие возможности работы в режиме LAN.

Сведём результаты в таблицу 1.1.

Таблица 1.1 – Сравнение аналогов

Критерии	CloudUpdate_Demo	30_https_ota	esp_ghota	08-LCD SimpleOTA	advanced_https_ota
Защита от некорректного образа программного обеспечения	-	-	-	-	-
Защита от ошибок при передаче данных	-	-	-	-	-
Шифрование на устройстве	-	-	-	-	-
Возможность вернуться к заводской версии программного обеспечения	-	-	+	-	+
Возможность работы в WAN сети (DNS, IP, etc.)	+	+	+	+	+
Возможность работы в режиме LAN (автономный AP)	-	-	-	-	-
Работа с двумя версиями программного обеспечения	-	-	-	-	-

После обзора аналогичных проектов можно сказать, что во всех обозреваемых проектах отсутствует защита от копирования, отсутствует защита от некорректного образа программного обеспечения, отсутствует возможность вернуться к заводским настройкам, отсутствует возможность отслеживать количество обновлённых устройств.

После анализа аналогичных проектов были выявлены следующие требования:

- возможность работы в WAN сети;
- возможность работы в режиме LAN;

- возможность обновления по расписанию или по запросу пользователя;
- защита от копирования;
- защита от ошибок при передаче данных;
- защита от некорректного образа программного обеспечения;
- автоматическую проверку на наличие новой версии программного обеспечения;
- проверка программного обеспечения на целостность;
- автоматическое шифрование программного обеспечения;
- телеметрия;
- защищённость передачи;
- работа с двумя версиями программного обеспечения.

2 ВЫБОР ИНСТРУМЕНТАРИЯ ДЛЯ РАЗРАБОТКИ ПРОГРАММНОЙ ЧАСТИ ПРОЕКТА

2.1 Выбор фреймворка для разработки

Существует два основных фреймворка для разработки программного обеспечения для ESP32: Arduino и ESP-IDF. Для выбора фреймворка для разработки сравним оба варианта:

1) Arduino Core for ESP32.

Фреймворк созданный с целью интегрировать ESP-IDF в Arduino IDE.

Достоинства:

– низкий порог вхождения;

– простота написания кода;

– большое количество библиотек.

Недостатки:

– отсутствие поддержки низкоуровневых функций;

– (запоздание обновлений);

– отсутствие поддержки функций ESP-IDF.

2) Espressif IoT Development Framework.

Фреймворк разработанный компанией Espressif для разработки приложений для устройств Интернета вещей.

Достоинства:

– поддержка низкоуровневых функций;

– гибкость конфигурирования устройства;

– возможность конфигурирования разделов flash-памяти.

Недостатки:

– сложность написания кода.

На основе результатов сравнения можно сделать вывод, что Espressif IoT Development Framework подходит для разработки больше, так как для написания программного компонента для автоматического обновления программного обеспечения необходимы низкоуровневые функции ESP-IDF.

2.2 Выбор среды разработки

Для разработки приложений на платформе ESP-IDF существует множество сред разработки, наиболее популярными из них являются Visual Studio Code с расширением ESP-IDF и Espressif-IDE.

Расширение ESP-IDF создано для разработки приложений для устройств Интернета вещей с использованием платформы ESP-IDF в редакторе исходного кода Visual Studio Code.

Espressif-IDE – это официальная интегрированная среда разработки, разработанная компанией Espressif на основе Eclipse CDT для разработки приложений для устройств Интернета вещей с использованием платформы ESP-IDF. Это автономная, настраиваемая IDE, созданная специально для ESP-IDF. Espressif-IDE поставляется с подключаемыми модулями IDF Eclipse, основными подключаемыми модулями Eclipse CDT и другими сторонними подключаемыми модулями платформы Eclipse.

2.3 Выбор протоколов передачи данных

Для получения информации о новой версии программного обеспечения необходимо выбрать протокол передачи данных.

Наиболее популярными протоколами для передачи данных в Интернете вещей являются MQTT и CoAP. MQTT больше подходит для передачи информации о новой версии программного обеспечения, так как в отличие от CoAP, обеспечивает хранение сообщений на брокере.

Для загрузки новой версии программного обеспечения необходимо выбрать протокол передачи данных.

Наиболее популярными протоколами передачи данных для передачи программного обеспечения являются HTTP, HTTPS и FTP. HTTP и FTP не подходит по требованию защищённость передачи из главы 1, поэтому был выбран протокол HTTPS.

3 ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ПРИЛОЖЕНИЯ

3.1 Процесс загрузки ESP32

Перед проектированием архитектуры приложения необходимо рассмотреть процесс запуска приложения в EPS-IDF.

Процесс запуска приложения ESP-IDF можно разделить на три этапа:

1) загрузчик первой стадии – находится в однократно программируемом постоянном запоминающем устройстве и его невозможно изменить. Этот код выполняется при каждом перезапуске микроконтроллера и загружает образ загрузчика второй стадии в ОЗУ из flash-памяти;

2) загрузчик второй стадии – считывает таблицу разделов и находит исполняемые разделы factory и OTA. Если разделов OTA несколько, то загрузчик выбирает какой раздел запускать основываясь на данных из раздела ota_data. Код загрузчика второй стадии можно изменить;

3) запуск приложения – после завершения работы загрузчика второй стадии запускается один из разделов с приложениями. Всего предусмотрено три раздела с приложениями: factory, ota_0 и ota_1 [8].

Процесс запуска ESP32 изображён на рисунке 3.1.

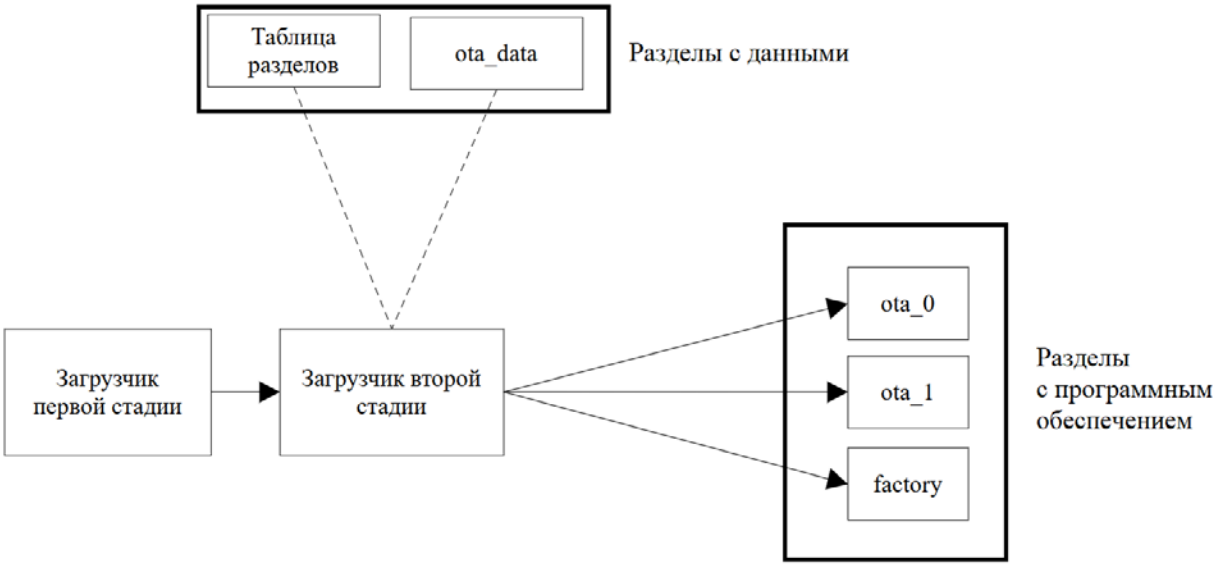

Рисунок 3.1 – Процесс запуска ESP32

3.2 Загрузчик первой стадии

Код загрузчика первой стадии находится в однократно программируемом ПЗУ и его невозможно изменить. Этот код выполняется при каждом перезапуске устройства и загружает образ загрузчика второй стадии в ОЗУ из flash-памяти.

Так как загрузчик первой стадии невозможно изменить, подробное рассмотрение его работы не требуется [8].

3.3 Загрузчик второй стадии

Flash-память микроконтроллера ESP32 может хранить множество приложений, а также различные группы данных. По умолчанию таблица разделов хранится в flash-памяти устройства по адресу 0x8000. Таблица разделов хранит информацию о том, где хранятся эти приложения и данные. Для выбора из какого раздела запускать приложение в ESP-IDF существует загрузчик второй стадии.

Загрузчик второй стадии находится в flash-памяти ESP32 по адресу 0x1000. Загрузчик второй стадии учувствует в шифровании и расшифровке программного обеспечения, а также в выборе какое программное обеспечение запускать.

После завершения проверки и загрузки загрузчика второй стадии загрузчик первой стадии делает безусловный переход на точку входа в двоичный образ загрузчика второй стадии.

Загрузчик второй стадии считывает данные из таблицы разделов, если в таблице разделов были найдены разделы OTA, то загрузчик второй стадии сверяется с информацией, записанной в разделе otadata для определения какой из разделов запустить. Если разделы OTA отсутствуют, то запускается раздел factory. После выбора раздела, загрузчик проверяет раздел на целостность [8].

Также в загрузчике второй стадии происходит шифрование и расшифровка программного обеспечения.

3.4 Запуск приложения

Для устройств Интернета вещей важно иметь возможность вернуться в изначальный режим работы, так как могут возникнуть проблемы во время обновления. Этим режимом работы в ESP-IDF является раздел factory.

По умолчанию в ESP-IDF раздел factory запускается, если нет разделов OTA или если разделы OTA повреждены. Также имеется возможность настроить вход в раздел factory, если зажата определённая кнопка.

Раздел factory не предназначен для изменения пользователем.

В разделы OTA будет загружаться новые версии программного обеспечения. В это программное обеспечение включается программный компонент для обновления программного обеспечения и основная логика работы приложения. В этой работе основная логика работы приложения разрабатываться не будет [9].

3.5 Архитектура

После рассмотрения процесса запуска ESP32 можно определить какие требования в каком компоненте программного обеспечения будут реализованы. Сведём эти результаты в таблицу 3.1.

Таблица 3.1 – Требования к разрабатываемым компонентам

Компоненты	Требования
загрузчик второй стадии	проверка программного обеспечения на целостность
	защита от копирования
	работа с двумя версиями программного обеспечения
	защита от некорректного образа программного обеспечения
	возможность вернуться к заводским настройкам
раздел factory	возможность работы в режиме LAN
разделы OTA	возможность работы в WAN сети
	защита от ошибок при передаче данных
	автоматическая проверка на наличие новой версии программного обеспечения
	телеметрия

3.6 Разработка алгоритма работы загрузчика второй стадии

Код загрузчика второй стадии будет переписан, чтобы удовлетворить следующие требования, определённые в главе 1:

- защита от некорректного образа программного обеспечения;
- защита от копирования;
- работа с двумя версиями программного обеспечения;
- возможность вернуться к заводским настройкам.

Защита от некорректного образа программного обеспечения будет реализована с помощью проверки контрольной суммы разделов внутри загрузчика второй стадии. Проверка целостности программного обеспечения во время работы загрузчика второй стадии позволит исключить возможность запуска некорректного программного обеспечения.

Защита от копирования будет реализована с помощью API ESP32, позволяющего шифровать и расшифровывать программное обеспечение.

Работа с двумя версиями программного обеспечения и возможность вернуться к заводским настройкам будут реализованы с помощью возможности загрузчика второй стадии выбирать какой из исполняемый разделов запускать.

Схема алгоритма представлена на рисунке 3.2.

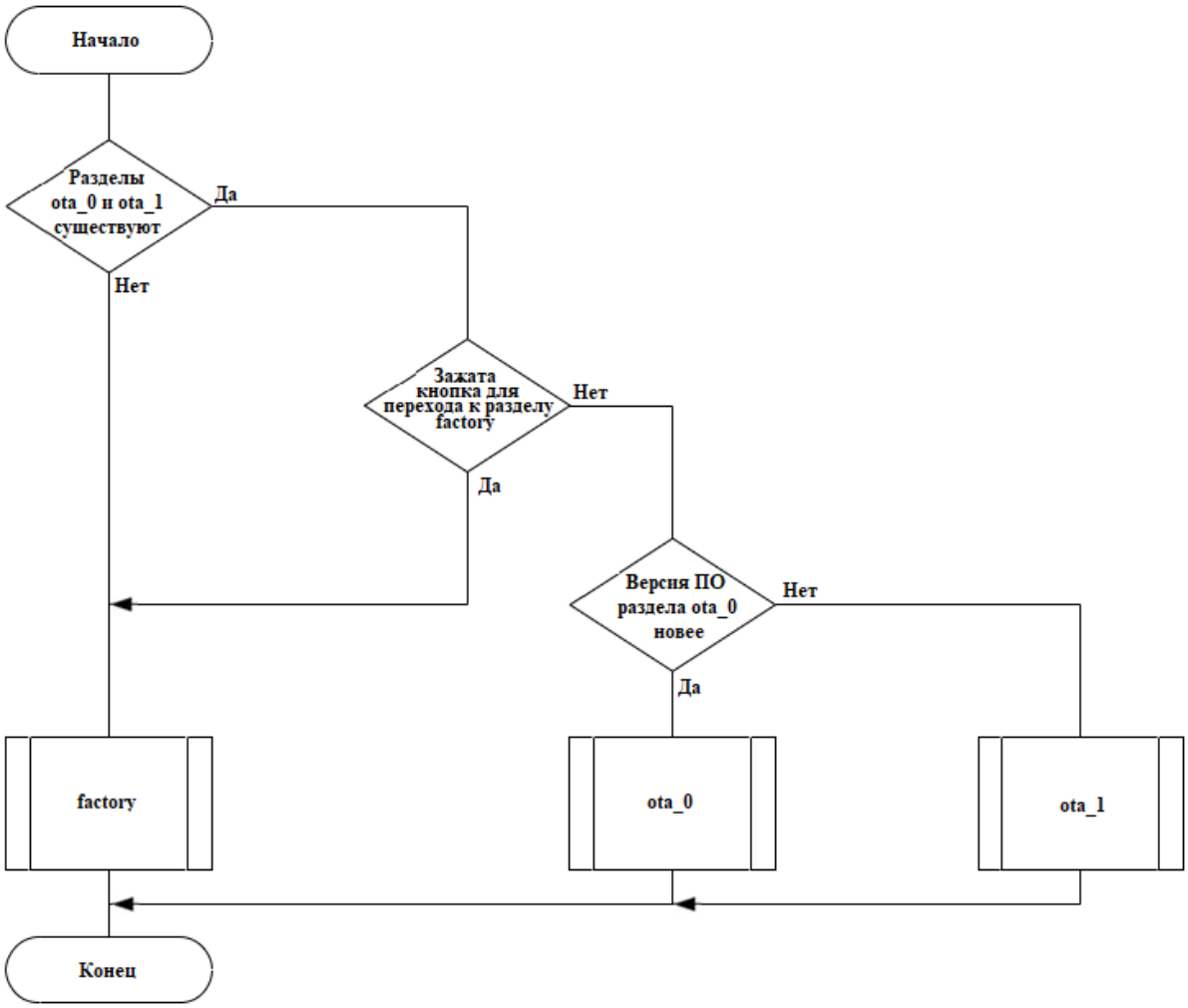

Рисунок 3.2 - Схема работы загрузчика второй стадии

3.7 Разработка алгоритма работы раздела factory

В разделе factory будут реализованы возможность обновления через LAN сеть и менеджер беспроводных соединений, в разделах OTA этого функционала не будет, так как этих разделов будет два и включать в них этот функционал будет затратно по памяти. Этот функционал позволяет выполнить следующие требования, определённые в главе 1:

– возможность работы в режиме LAN;

– возможность работать в WAN сети.

Возможность работы в режиме LAN будет реализована следующим образом: устройство создаёт точку доступа, запускает HTTP сервер, после чего пользователь подключается к точке доступа, переходит на веб-страницу обновления по LAN и загружает образ программного обеспечения, после чего устройство перезагружается.

Менеджер беспроводных соединений будет реализован следующим образом: устройство создаёт точку доступа, запускает HTTP сервер, после чего пользователь подключается к точке доступа, заходит на веб-страницу менеджера беспроводных соединений, выбирает в списке Wi-Fi сеть к которой хочет подключиться, вводит пароль и перезапускает устройство.

Схема алгоритма представлена на рисунке 3.3.

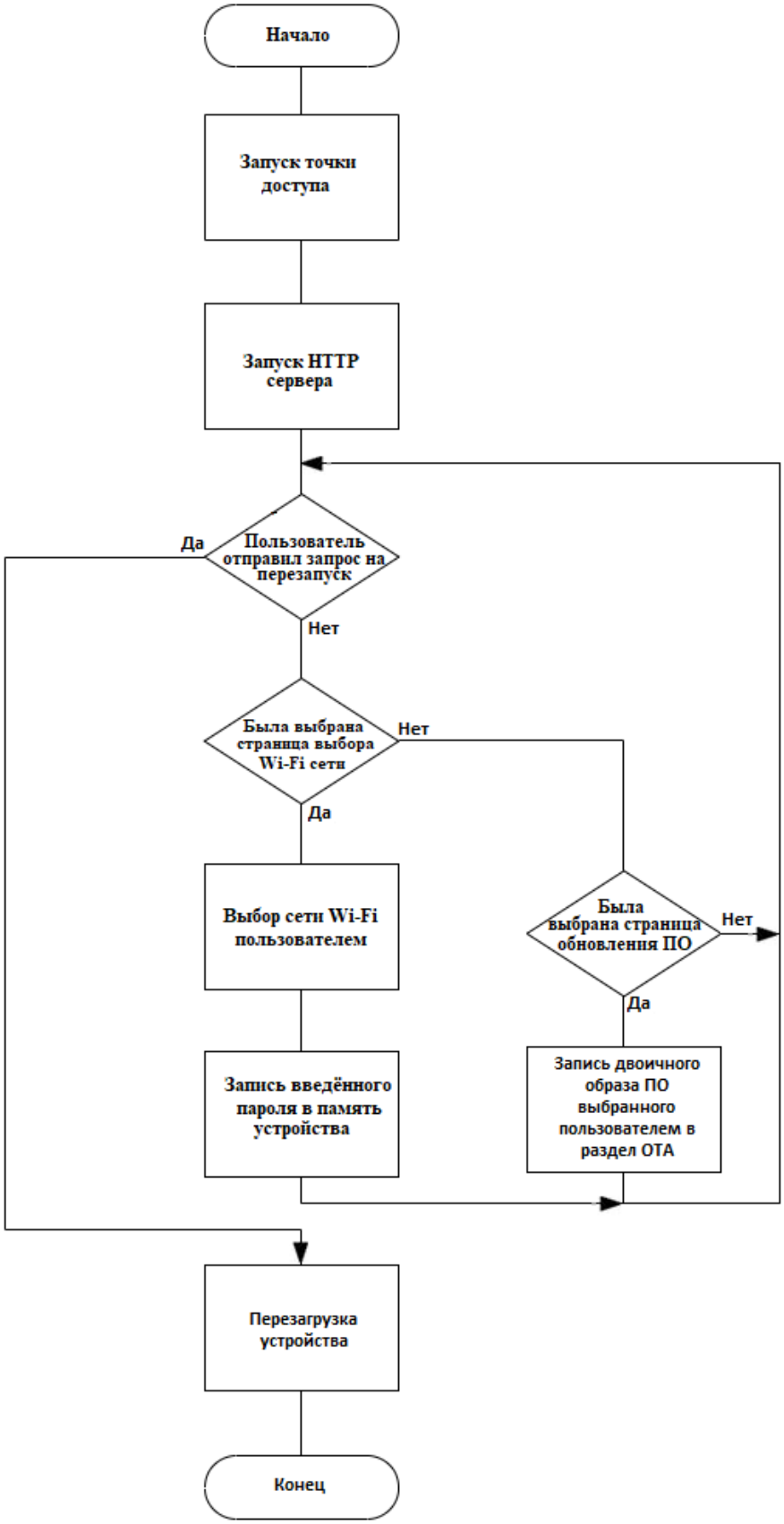

Рисунок 3.3 – Схема работы раздела factory

3.8 Разработка алгоритма работы программного компонента для обновления программного обеспечения

В программном компоненте для обновления программного обеспечения не будет реализована бизнес логика, так как реализация бизнес логики сильно зависит от задачи. Для примера будет использоваться мигание светодиодом. Код программного компонента будет записываться в разделы для OTA. Разделов OTA будет два, и чтобы загрузчик второй стадии понимал какой раздел запускать, программный компонент должен с помощью функций ESP-IDF записывать информацию о том, что он полностью функционирует. Если новый раздел после обновления не запустился, то функция, записывающая информацию о работе раздела, не вызывается и при следующем запуске загрузчик второй стадии запускает предыдущий раздел.

В программном компоненте для обновления программного обеспечения необходимо реализовать функционал:

1) получение информации о новой версии программного обеспечения;
2) обновление программного обеспечения по расписанию;
3) защита от некорректного образа программного обеспечения;
4) телеметрия.

Схема алгоритма представлена на рисунке 3.4.

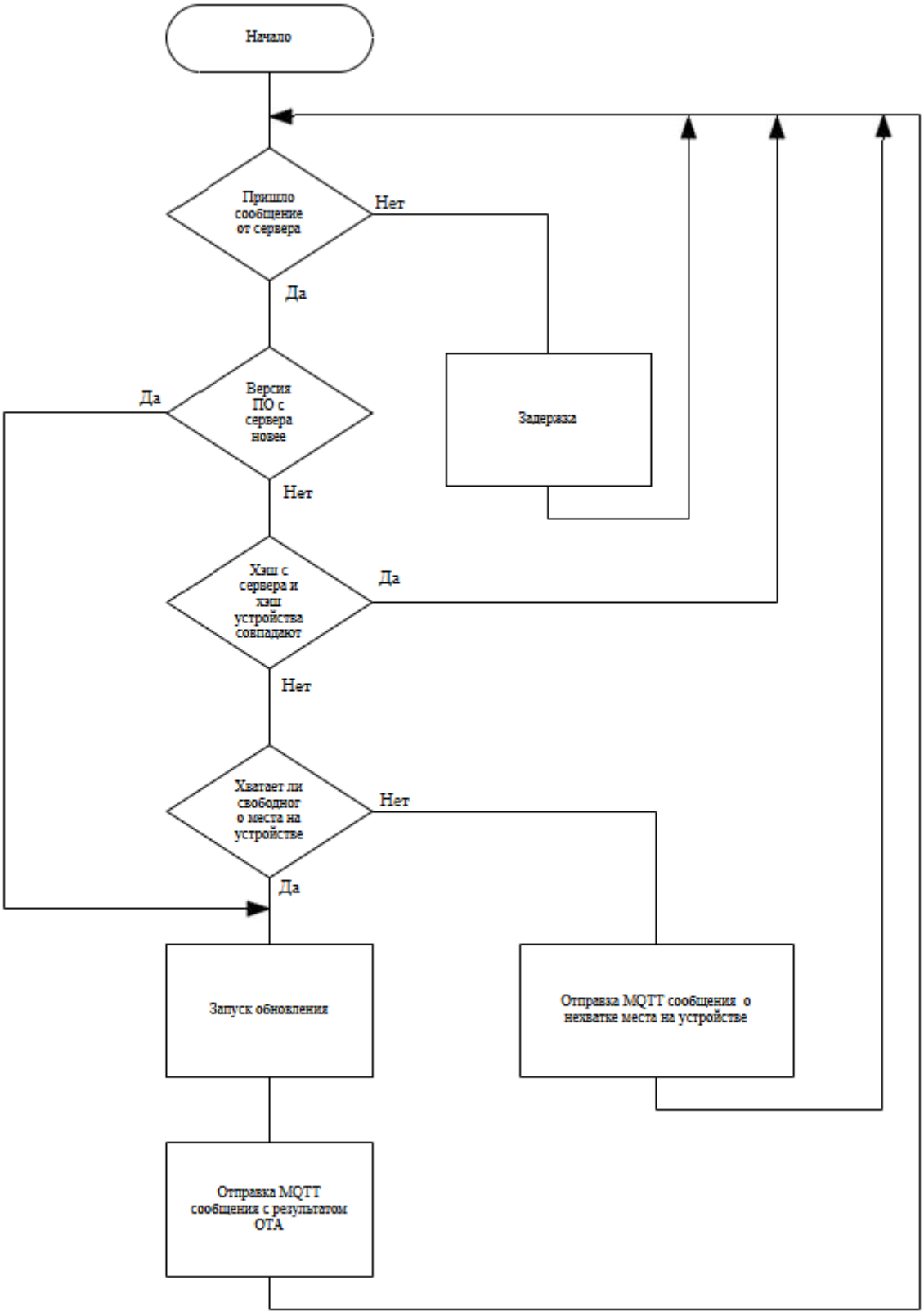

Рисунок 3.4 – Схема работы программного компонента для обновления программного обеспечения

4 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ПРОЕКТА

4.1 Реализация загрузчика второй стадии

Согласно главе 3.6 в загрузчике второй стадии необходимо реализовать проверку программного обеспечения на целостность и сброс до заводских настроек.

Сброс будет происходить, если при запуске пользователь зажмёт определённую кнопку на устройстве больше чем на 5 секунд.

Схема алгоритма представлена на рисунке 4.1.

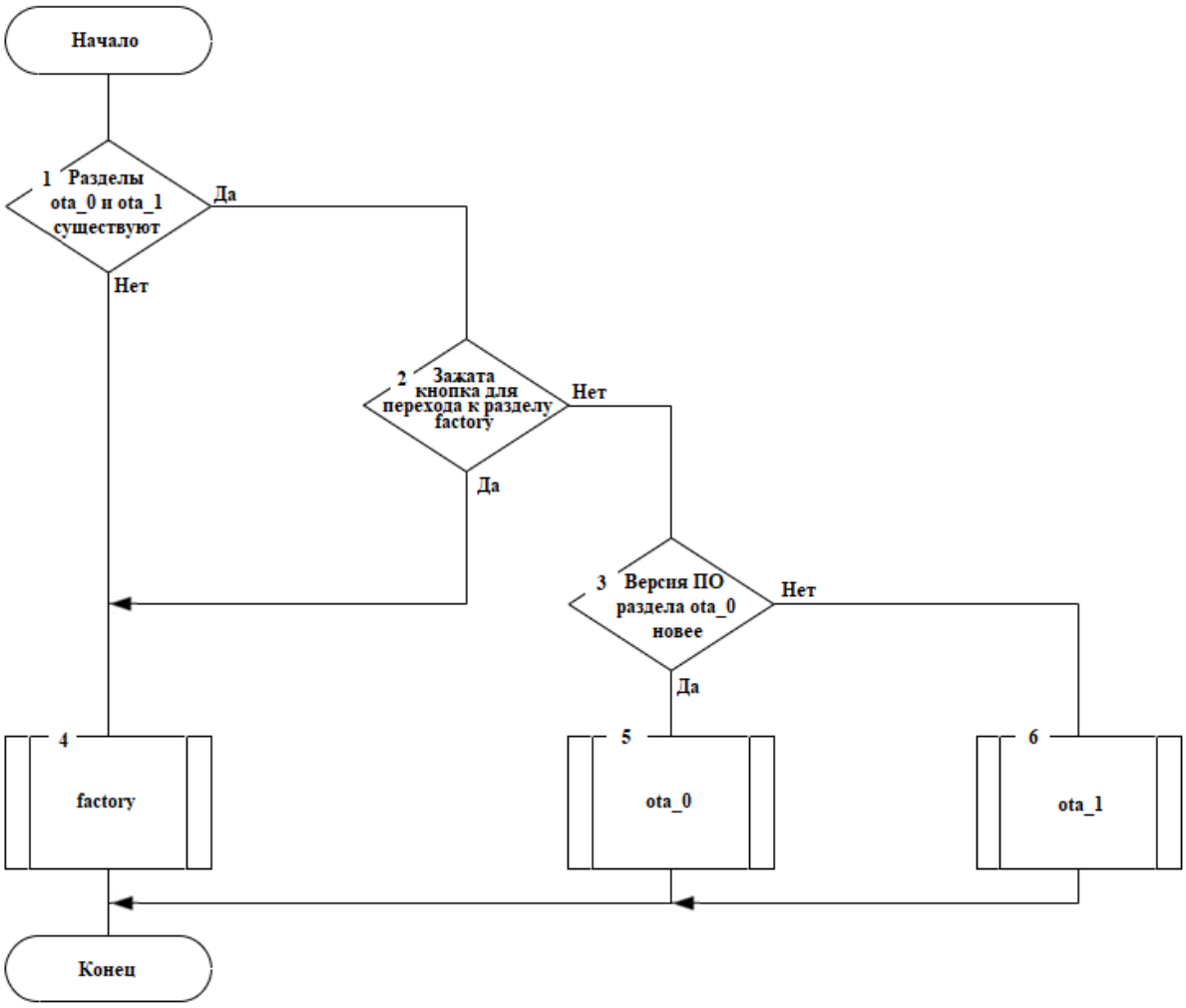

Рисунок 4.1 – Схема работы загрузчика второй стадии

Для реализации блока 2 необходимо сначала настроить конфигурацию проекта с помощью консольной утилиты idf.py. С помощью команды idf.py menuconfig необходимо перейти в меню конфигурации и в разделе bootloader config включить опцию GPIO triggers factory reset. Также в меню

конфигурации необходимо включить опцию шифрования программного обеспечения, чтобы загрузчик второй стадии мог шифровать и расшифровывать программное обеспечение. Для этого в разделе Security features включить опцию Enable flash encryption on boot.

Для реализации блока 1 необходимо воспользоваться функцией ESP-IDF read_otadata.

Для реализации блока 3 необходимо написать функцию которая считывает данные из раздела otadata с помощью функции bootloader_common_get_active_otadata.

4.2 Реализация раздела factory

Согласно главе 3.7 в разделе factory необходимо реализовать обновление через LAN сеть и менеджер беспроводных соединений.

Схема алгоритма представлена на рисунке 4.2.

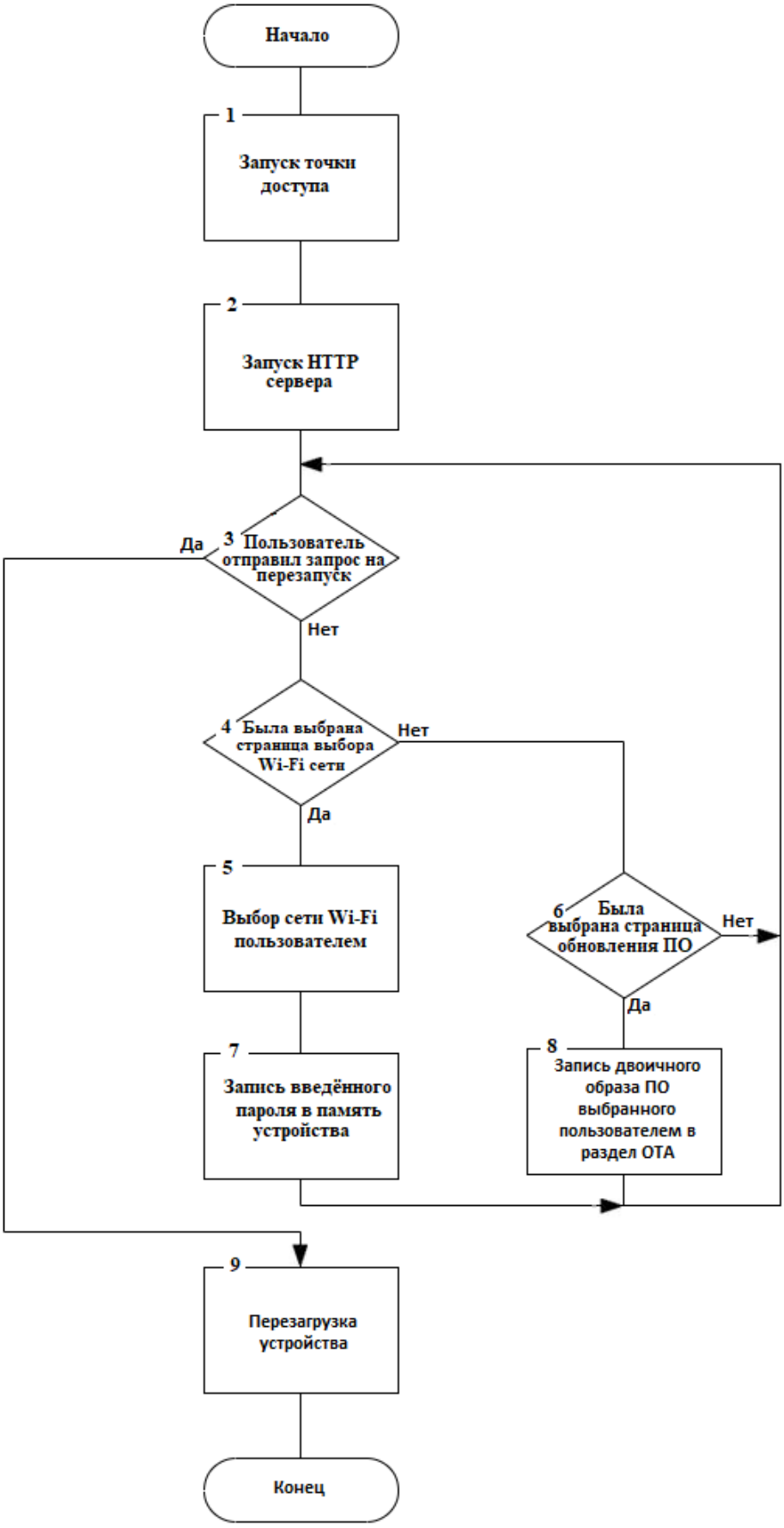

Рисунок 4.2 - Схема работы раздела factory

Для реализации блока 1 необходимо создать точку доступа с помощью функции esp-idf esp_netif_create_default_wifi_ap. Затем нужно установить режим работы WI-Fi как точки доступа с помощью функции ESP-IDF esp_wifi_set_mode. После этого необходимо настроить конфигурацию Wi-Fi сети с помощью структуры wifi_config_t. Необходимо настроить следующие поля: название точки доступа, пароль точки доступа и максимальное количество подключений к точке доступа. После настройки конфигурации точик доступа необходимо запустить точку доступа с помощью функции esp_wifi_start.

Для реализации блоков 2-8 необходимо реализовать HTTP сервер. Для создания HTTP сервера необходимо создать веб-страницу, с которой пользователь будет взаимодействовать. После создания веб-страницы необходимо реализовать методы GET и POST. После этого необходимо запустить сервер с помощью функции httpd_start.

4.3 Реализация раздела с основным программным обеспечением

Согласно главе 3.8 в программном компоненте для обновления программного обеспечения необходимо реализовать получение информации о новой версии программного обеспечения, обновление программного обеспечения по расписанию, защита от некорректного образа программного обеспечения и работа с двумя версиями программного обеспечения.

Схема алгоритма представлена на рисунке 4.3.

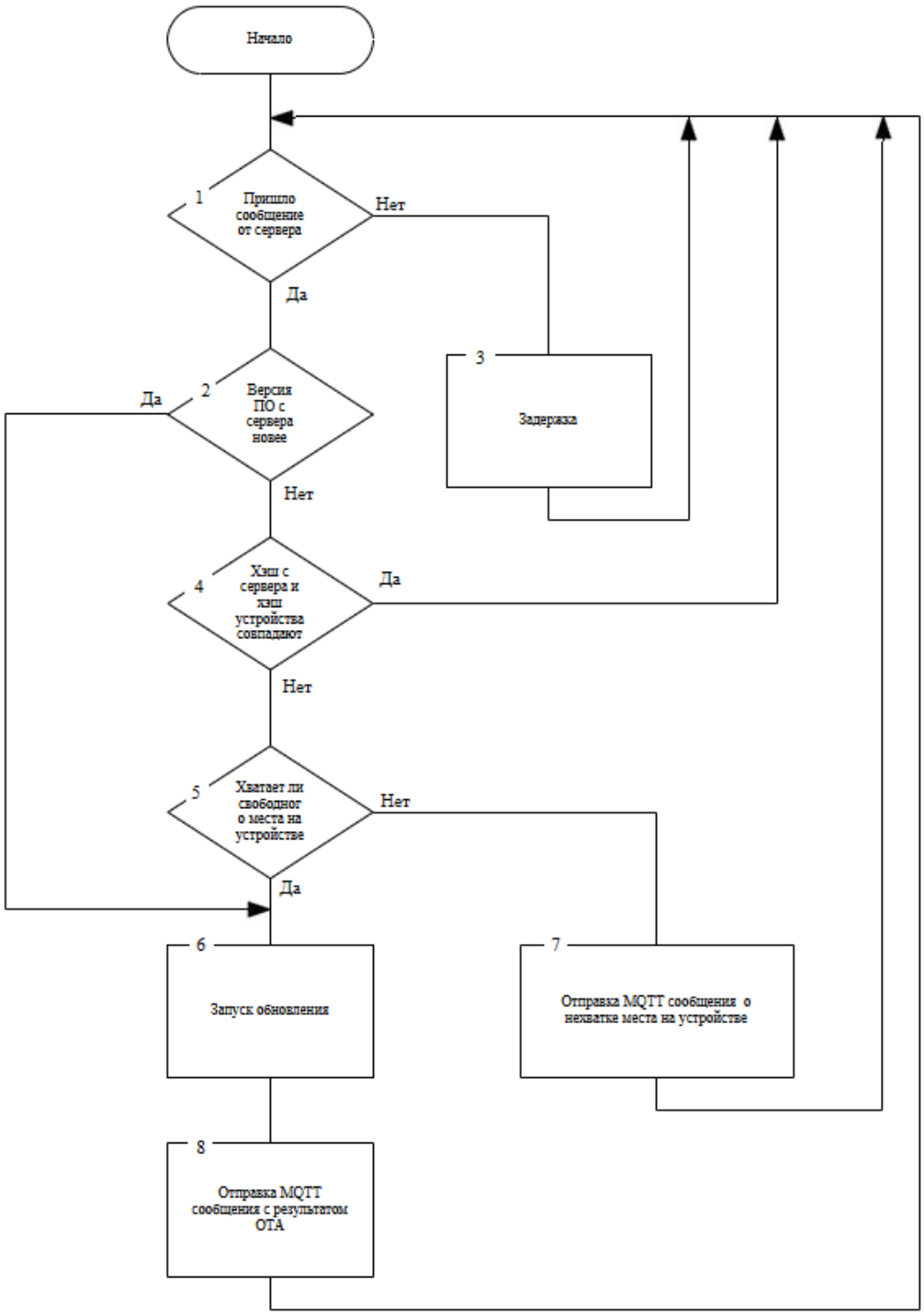

Рисунок 4.3 – Схема работы программного компонента для обновления программного обеспечения

Для реализации блоков 1, 7 и 8 необходимо написать функцию работы с MQTT сервером.

Для регистрации событий MQTT необходимо написать функцию обрабатывающую эти события.

Для обработки событий вызванных работой с протоколом MQTT будет написана функция mqtt_event_handler. События обрабатываемые функцией mqtt_event_handler:

1) MQTT_EVENT_CONNECTED – клиент успешно установил соединение с брокером. При возникновении данного события устройство подписывается на топик обновления программного обеспечения с помощью функции ESP-IDF esp_mqtt_client_subscribe.

2) MQTT_EVENT_DATA – клиент получил сообщение от сервера. При возникновении данного события полученное с сервера сообщение записывается в переменную строкового типа. После записи текста сообщения в переменную, на основании текста сообщения решается, нужно ли обновлять программное обеспечение. При необходимости обновления создаётся задача обновления программного обеспечения. После выполнения задачи обновления устройство отправляет на сервер сообщение с результатом обновления с помощью функции esp_mqtt_client_publish.

Для запуска MQTT клиента необходимо создать дескриптор MQTT клиента, а также зарегистрировать MQTT событие. Для создания дескриптора MQTT клиента в ESP-IDF есть функция esp_mqtt_client_init принимающая в качестве параметра структуру esp_mqtt_client_config_t содержащую конфигурацию MQTT клиента. В конфигурации MQTT клиента будет указан адрес брокера. Для регистрации MQTT события необходим дескриптор MQTT, тип регистрируемого события и обработчик MQTT событий. После регистрации события можно запускать MQTT клиент с помощью функции ESP-IDF esp_mqtt_client_start.

Для реализации блока 6 использована функция ESP-IDF esp_https_ota, которой передаётся параметр ota_config, содержащий URL для скачивания новой версии ПО и сертификат сервера.

Исходный код программного компонента для обновления программного обеспечения устройств на базе Интернета вещей представлен в Приложении.

5 ПРОВЕРКА РАБОТОСПОСОБНОСТИ РАЗРАБОТАННОГО ПРОЕКТА В РЕАЛЬНЫХ УСЛОВИЯХ

Для определения соответствует ли разработанное программное обеспечение требованиям, определённым в главе 1 был проведён ряд тестов. Перечень тестов приведён в таблице 5.1.

Таблица 5.1 – Перечень проведённых тестов

Тестируемая функция	Описание тестирования	Ожидаемый результат	Полученный результат
Обновление по LAN сети.	1. Запуск устройства. 2. Во время запуска устройства зажата кнопка на устройстве для перехода в раздел factory. 3. Перейти на веб-страницу созданную устройством. 4. На веб-странице нажать на кнопку “Обновление программного обеспечения”. 5. Выбрать файл с программным обеспечением. 6. Нажать на	В результате устройство должно обновится до новой версии.	Соответствует ожидаемому.

Продолжение таблицы 5.1

Тестируемая функция	Описание тестирования	Ожидаемый результат	Полученный результат
	кнопку перезагрузки устройства на веб-странице.		
Менеджер беспроводных соединений.	1. Запуск устройства. 2. Во время запуска устройства зажата кнопка на устройстве для перехода в раздел factory. 3. Перейти на веб-страницу созданную устройством. 4. На веб-странице нажать на кнопку “подключение к сети Wi-Fi”. 5. Выбрать сеть Wi-Fi и ввести пароль для подключения. 6. Нажать на кнопку перезагрузки	В результате устройство должно запомнить новую Wi-Fi сесть и автоматически к ней подключатся во время каждого запуска.	Соответствует ожидаемому.

Окончание таблицы 5.1

Тестируемая функция	Описание тестирования	Ожидаемый результат	Полученный результат
	устройства на веб-странице.		
Обновление по HTTPS серверу.	1. На MQTT сервер публикуется сообщение с флагом retain о текущей версии программного обеспечения. 2. Ожидается сообщение от устройство с результатом обновления.	В результате устройство должно обновится до новой версии и отправить сообщение о результате обновления на MQTT сервер.	Соответствует ожидаемому.

ЗАКЛЮЧЕНИЕ

В ходе выпускной квалификационной работы был разработан программный компонент для автоматического обновления программного обеспечения устройств на базе Интернета вещей.

Был проведён обзор и анализ аналогичных проектов беспроводного обновления, в ходе которого были сформулированы критерии сравнения аналогичных проектов, а также на основе результатов сравнения аналогов были выявлены требования к разрабатываемому программному обеспечению.

Был проведён выбор инструментария для разработки программной части проекта, в ходе которого был выбран фреймворк для разработки, выбор среды разработки, а также выбор протоколов передачи данных для получения информации о новой версии программного обеспечения и для загрузки новой версии программного обеспечения.

Было произведено проектирование архитектуры приложения, в ходе которого был рассмотрен процесс запуска микроконтроллера ESP32, а также была разработана архитектура для каждого компонента разрабатываемого программного обеспечения.

Было разработано программное обеспечение, в ходе разработки программного обеспечения был реализован функционал соответствующий определённым в ходе обзора аналогов требованиям.

Была проведена проверка работоспособности разработанного проекта в реальных условиях, в ходе которой был проведён ряд тестов для определения, соответствует ли разработанное программное обеспечение требования.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1 Что такое интернет вещей? [Электронный ресурс]. – URL: https://www.kaspersky.ru/resource-center/definitions/what-is-iot. (Дата обращения: 20.04.2023).

2 Микроконтроллер [Электронный ресурс]. – URL: https://x-logic.ru/articles/mikrokontroller. (Дата обращения: 20.04.2023).

3 CloudUpdate_Demo [Электронный ресурс]. – URL: https://github.com/MoonFox2006/CloudUpdate_Demo. (Дата обращения: 20.04.2023). обращения: 20.04.2023).

4 30_https_ota [Электронный ресурс]. – URL: https://github.com/lucadentella/esp32-tutorial/tree/master/30_https_ota. (Дата обращения: 26.02.2023).

5 esp_ghota [Электронный ресурс]. URL: https://github.com/Fishwaldo/esp_ghota. (Дата обращения: 26.02.2023).

6 08-LCD SimpleOTA [Электронный ресурс]. – URL: https://github.com/otadrive/ESP32-OTAdrive-Examples/blob/master/08-LCD%20SimpleOTA. (Дата обращения: 20.04.2023).

7 advanced_https_ota [Электронный ресурс]. – URL: https://github.com/espressif/esp-idf/tree/master/examples/system/ota/advanced_https_ota. (Дата обращения: 20.04.2023).

8 Application Startup Flow [Электронный ресурс]. – URL: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/startup.html

9 Over The Air Updates [Электронный ресурс]. – URL: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/ota.html

ПРИЛОЖЕНИЕ

Исходный код программного компонента для обновления программного обеспечения устройств на базе Интернета вещей

```
#include <stdio.h>
#include <stdint.h>
#include <stddef.h>
#include <string.h>
#include "esp_wifi.h"
#include "esp_system.h"
#include "nvs_flash.h"
#include "esp_event.h"
#include "esp_netif.h"
#include "protocol_examples_common.h"
#include "nvs.h"
#include <sys/socket.h>
#include "string.h"

#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/semphr.h"
#include "freertos/queue.h"

#include "lwip/sockets.h"
#include "lwip/dns.h"
#include "lwip/netdb.h"

#include "esp_log.h"
#include "mqtt_client.h"

#include "esp_ota_ops.h"
#include "esp_http_client.h"
#include "esp_https_ota.h"
#ifdef CONFIG_EXAMPLE_USE_CERT_BUNDLE
#include "esp_crt_bundle.h"
#endif

#define HASH_LEN 32

static const char *TAG = "MQTT_EXAMPLE";
extern const uint8_t server_cert_pem_start[] asm("_binary_ca_cert_pem_start");
extern const uint8_t server_cert_pem_end[] asm("_binary_ca_cert_pem_end");

char *OTA_url;

#define OTA_URL_SIZE 256

esp_err_t _http_event_handler(esp_http_client_event_t *evt)
{
    switch (evt->event_id) {
    case HTTP_EVENT_ERROR:
        ESP_LOGD(TAG, "HTTP_EVENT_ERROR");
        break;
    case HTTP_EVENT_ON_CONNECTED:
        ESP_LOGD(TAG, "HTTP_EVENT_ON_CONNECTED");
        break;
    case HTTP_EVENT_HEADER_SENT:
        ESP_LOGD(TAG, "HTTP_EVENT_HEADER_SENT");
        break;
    case HTTP_EVENT_ON_HEADER:
```

```
            ESP_LOGD(TAG, "HTTP_EVENT_ON_HEADER, key=%s, value=%s", evt->header_key, evt-
>header_value);
            break;
        case HTTP_EVENT_ON_DATA:
            ESP_LOGD(TAG, "HTTP_EVENT_ON_DATA, len=%d", evt->data_len);
            break;
        case HTTP_EVENT_ON_FINISH:
            ESP_LOGD(TAG, "HTTP_EVENT_ON_FINISH");
            break;
        case HTTP_EVENT_DISCONNECTED:
            ESP_LOGD(TAG, "HTTP_EVENT_DISCONNECTED");
            break;
        case HTTP_EVENT_REDIRECT:
            ESP_LOGD(TAG, "HTTP_EVENT_REDIRECT");
            break;
        }
        return ESP_OK;
    }

    void simple_ota_example_task(void *pvParameter)
    {
        ESP_LOGI(TAG, "Starting OTA example task");
    #ifdef CONFIG_EXAMPLE_FIRMWARE_UPGRADE_BIND_IF
        esp_netif_t *netif = get_example_netif_from_desc(bind_interface_name);
        if (netif == NULL) {
            ESP_LOGE(TAG, "Can't find netif from interface description");
            abort();
        }
        struct ifreq ifr;
        esp_netif_get_netif_impl_name(netif, ifr.ifr_name);
        ESP_LOGI(TAG, "Bind interface name is %s", ifr.ifr_name);
    #endif
        esp_http_client_config_t config = {
                .url = OTA_url,
    #ifdef CONFIG_EXAMPLE_USE_CERT_BUNDLE
            .crt_bundle_attach = esp_crt_bundle_attach,
    #else
            .cert_pem = (char *)server_cert_pem_start,
            .event_handler = _http_event_handler,
            .keep_alive_enable = true,
    #ifdef CONFIG_EXAMPLE_FIRMWARE_UPGRADE_BIND_IF
            .if_name = &ifr,
    #endif
        };

    #ifdef CONFIG_EXAMPLE_FIRMWARE_UPGRADE_URL_FROM_STDIN
        char url_buf[OTA_URL_SIZE];
        if (strcmp(config.url, "FROM_STDIN") == 0) {
            example_configure_stdin_stdout();
            fgets(url_buf, OTA_URL_SIZE, stdin);
            int len = strlen(url_buf);
            url_buf[len - 1] = '\0';
            config.url = url_buf;
        } else {
            ESP_LOGE(TAG, "Configuration mismatch: wrong firmware upgrade image url");
            abort();
        }
    #endif

    #ifdef CONFIG_EXAMPLE_SKIP_COMMON_NAME_CHECK
        config.skip_cert_common_name_check = true;
```

```
#endif

    esp_https_ota_config_t ota_config = {
        .http_config = &config,
    };
    ESP_LOGI(TAG, "Attempting to download update from %s", OTA_url);
    esp_err_t ret = esp_https_ota(&ota_config);
    if (ret == ESP_OK) {
        ESP_LOGI(TAG, "OTA Succeed, Rebooting...");
        esp_restart();
    } else {
        ESP_LOGE(TAG, "Firmware upgrade failed");
    }
    while (1) {
        vTaskDelay(1000 / portTICK_PERIOD_MS);
    }
}

static void print_sha256(const uint8_t *image_hash, const char *label)
{
    char hash_print[HASH_LEN * 2 + 1];
    hash_print[HASH_LEN * 2] = 0;
    for (int i = 0; i < HASH_LEN; ++i) {
        sprintf(&hash_print[i * 2], "%02x", image_hash[i]);
    }
    ESP_LOGI(TAG, "%s %s", label, hash_print);
}

static void get_sha256_of_partitions(void)
{
    uint8_t sha_256[HASH_LEN] = { 0 };
    esp_partition_t partition;
    partition.address   = ESP_BOOTLOADER_OFFSET;
    partition.size      = ESP_PARTITION_TABLE_OFFSET;
    partition.type      = ESP_PARTITION_TYPE_APP;
    esp_partition_get_sha256(&partition, sha_256);
    print_sha256(sha_256, "SHA-256 for bootloader: ");
    esp_partition_get_sha256(esp_ota_get_running_partition(), sha_256);
    print_sha256(sha_256, "SHA-256 for current firmware: ");
}


static void log_error_if_nonzero(const char *message, int error_code)
{
    if (error_code != 0) {
        ESP_LOGE(TAG, "Last error %s: 0x%x", message, error_code);
    }
}


static void mqtt_event_handler(void *handler_args, esp_event_base_t base, int32_t event_id, void
*event_data)
{
    ESP_LOGD(TAG, "Event dispatched from event loop base=%s, event_id=%d", base, event_id);
    esp_mqtt_event_handle_t event = event_data;
    esp_mqtt_client_handle_t client = event->client;
    switch ((esp_mqtt_event_id_t)event_id) {
    case MQTT_EVENT_CONNECTED:
        ESP_LOGI(TAG, "MQTT_EVENT_CONNECTED");

        esp_mqtt_client_subscribe(client, "ESP32_OTA", 0);
```

```
            break;
        case MQTT_EVENT_DISCONNECTED:
            ESP_LOGI(TAG, "MQTT_EVENT_DISCONNECTED");
            break;

        case MQTT_EVENT_SUBSCRIBED:
            ESP_LOGI(TAG, "MQTT_EVENT_SUBSCRIBED, msg_id=%d", event->msg_id);
            break;
        case MQTT_EVENT_UNSUBSCRIBED:
            ESP_LOGI(TAG, "MQTT_EVENT_UNSUBSCRIBED, msg_id=%d", event->msg_id);
            break;
        case MQTT_EVENT_PUBLISHED:
            ESP_LOGI(TAG, "MQTT_EVENT_PUBLISHED, msg_id=%d", event->msg_id);
            break;
        case MQTT_EVENT_DATA:
            ESP_LOGI(TAG, "MQTT_EVENT_DATA");
            printf("TOPIC=%.*s\r\n", event->topic_len, event->topic);
            printf("DATA=%.*s\r\n", event->data_len, event->data);
            OTA_url = event->data;
            xTaskCreate(&simple_ota_example_task, "ota_example_task", 8192, NULL, 5, NULL);
            break;
        case MQTT_EVENT_ERROR:
            ESP_LOGI(TAG, "MQTT_EVENT_ERROR");
            if (event->error_handle->error_type == MQTT_ERROR_TYPE_TCP_TRANSPORT) {
                log_error_if_nonzero("reported from esp-tls", event->error_handle->esp_tls_last_esp_err);
                log_error_if_nonzero("reported from tls stack", event->error_handle->esp_tls_stack_err);
                log_error_if_nonzero("captured as transport's socket errno",  event->error_handle-
>esp_transport_sock_errno);
                ESP_LOGI(TAG, "Last errno string (%s)", strerror(event->error_handle->esp_transport_sock_errno));

            }
            break;
        default:
            ESP_LOGI(TAG, "Other event id:%d", event->event_id);
            break;
        }
    }

    static void mqtt_app_start(void)
    {
        esp_mqtt_client_config_t mqtt_cfg = {
            .broker.address.uri = CONFIG_BROKER_URL,
        };
    #if CONFIG_BROKER_URL_FROM_STDIN
        char line[128];

        if (strcmp(mqtt_cfg.broker.address.uri, "FROM_STDIN") == 0) {
            int count = 0;
            printf("Please enter url of mqtt broker\n");
            while (count < 128) {
                int c = fgetc(stdin);
                if (c == '\n') {
                    line[count] = '\0';
                    break;
                } else if (c > 0 && c < 127) {
                    line[count] = c;
                    ++count;
                }
                vTaskDelay(10 / portTICK_PERIOD_MS);
            }
            mqtt_cfg.broker.address.uri = line;
```

```
        printf("Broker url: %s\n", line);
    } else {
        ESP_LOGE(TAG, "Configuration mismatch: wrong broker url");
        abort();
    }
#endif

    esp_mqtt_client_handle_t client = esp_mqtt_client_init(&mqtt_cfg);
    esp_mqtt_client_register_event(client, ESP_EVENT_ANY_ID, mqtt_event_handler, NULL);
    esp_mqtt_client_start(client);
}

void app_main(void)
{
    ESP_LOGI(TAG, "[APP] Startup..");
    ESP_LOGI(TAG, "[APP] Free memory: %d bytes", esp_get_free_heap_size());
    ESP_LOGI(TAG, "[APP] IDF version: %s", esp_get_idf_version());

    esp_err_t err = nvs_flash_init();
      if (err == ESP_ERR_NVS_NO_FREE_PAGES || err == ESP_ERR_NVS_NEW_VERSION_FOUND) {
          err = nvs_flash_init();
      }
      ESP_ERROR_CHECK(err);

      get_sha256_of_partitions();

    esp_log_level_set("*", ESP_LOG_INFO);
    esp_log_level_set("mqtt_client", ESP_LOG_VERBOSE);
    esp_log_level_set("MQTT_EXAMPLE", ESP_LOG_VERBOSE);
    esp_log_level_set("TRANSPORT_BASE", ESP_LOG_VERBOSE);
    esp_log_level_set("esp-tls", ESP_LOG_VERBOSE);
    esp_log_level_set("TRANSPORT", ESP_LOG_VERBOSE);
    esp_log_level_set("outbox", ESP_LOG_VERBOSE);

    ESP_ERROR_CHECK(nvs_flash_init());
    ESP_ERROR_CHECK(esp_netif_init());
    ESP_ERROR_CHECK(esp_event_loop_create_default());
    ESP_ERROR_CHECK(example_connect());

    mqtt_app_start();
}
```