Программная документация
к загрузчику для стенда-конструктора SDK-1.1M

Санкт-Петербург, 2023 г.

Содержание

Введение

Загрузчик является небольшой программой для встроенной системы (такой как микроконтроллер или система на кристалле), которая выполняется при включении или сбросе. Его первичная функция состоит в том, чтобы загрузить и выполнить прикладное приложение, которое обычно хранится в энергонезависимой памяти, обычно флэш-памяти или ROM. Загрузчик может также выполнить другие задачи, такие как проверка целостности прикладного приложения, конфигурирование аппаратных средств системы и обеспечение механизма для обновления программного обеспечения прикладного приложения. Загрузчик обычно хранится в части энергонезависимой памяти, которая защищена от записи, чтобы он не мог быть случайно перезаписан или стерт. Некоторые встроенные системы также имеют вторичный загрузчик, названный загрузчиком восстановления, который может использоваться для восстановления системы в случае, если основной загрузчик становится поврежденным.

Основная цель загрузчиков состоит в том, чтобы обеспечить возможность обновления прикладного приложения и конфигурирования аппаратных средств системы. Это важно для систем, к которым у разработчиков не будет доступа, поскольку это допускает исправление ошибок, добавление новых возможностей и настройку системы под конкретные требования приложения. Кроме того, встроенные загрузчики могут также использоваться, чтобы диагностировать и устранять проблемы с системой, а также выполнить такие задачи, как стирание или программирование энергонезависимой памяти.

1 Описание архитектуры

Загрузчик обычно находится в начале флэш-памяти, а образы двух приложений хранятся в разных местах энергонезависимой памяти (рис. 1). Каждое приложение имеет соответствующую конфигурацию. Загрузчик отвечает за загрузку кода приложения в память, проверку наличия корректного приложения и передачу управления коду приложения. Когда требуется обновление прошивки, новый образ загружается в неактивную область памяти. После проверки нового образа загрузчик может переключиться на новый образ и начать выполнение обновленной прошивки. Такой подход обеспечивает более надежный процесс обновления встроенного ПО, поскольку всегда имеется резервный образ на случай возникновения проблем с новым образом.

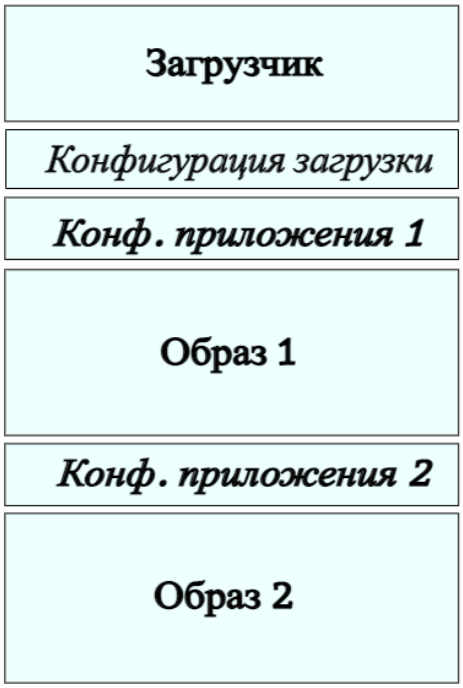

Рисунок 1 – Условное расположение загрузчика и приложений в памяти

Конфигурация приложения хранит 5 значений (табл. 1):

1) Значение для выбора приложения для выполнения. При загрузке новой прошивки это значение устанавливается загрузчиком на 1

меньше, чем в другой конфигурационной области. Таким образом, выполняется то приложение, значение у которого меньше.

2) Флаг первой загрузки. Изначально после стирания области конфигурации равен 1. Устанавливается загрузчиком в 0 при переходе из загрузчика в приложение. Используется для того, чтобы игнорировать флаг валидации, если приложение ещё не запускалось.

3) Флаг валидации. Изначально после стирания области конфигурации равен 1. Устанавливается приложением в 0 при его корректной работе. Если флаг первой загрузки равен 0, а флаг валидации равен 1, приложение считается поврежденным.

4) Версия приложения. Указывается пользователем при загрузке новой прошивки.

5) Флаг корректности обновления. Изначально после стирания области конфигурации равен 1. Устанавливается загрузчиком в 0 после завершения записи прошивки во флэш-память. Если флаг корректности обновления равен 1, то приложение не подлежит дальнейшей проверке и запуску.

Таблица 1 – конфигурация приложения

Название	Размер, бит	Принимаемые значения
Убывающее значение	32	[0; 4294967295]
Флаг первой загрузки	32	1 – перед первым запуском 0 – после запуска
Флаг валидации	32	1 – корректность приложения не подтверждена 0 – приложение валидно
Версия приложения	32	[0; 4294967295]

Флаг корректности обновления	32	1 – произошла ошибка при записи приложения 0 – запись прошла успешно

Приложение считается валидным только в двух случаях:

1) Флаг корректности обновления равен 0, флаг валидации равен 0. В этом случае приложение корректно записалось, запустилось и установило флаг валидации.

2) Флаг корректности обновления равен 0, флаг валидации равен 1, флаг первой загрузки равен 1. В этом случае приложение корректно записалось, но ещё не запускалось.

Конфигурация загрузки хранит одно 32-битное значение – флаг требования обновления. Он используется при старте программы.

2 Работа загрузчика

При старте (сбросе) выполняется загрузчик. При запуске загрузчика выполняется механизм определения поведения программы – дальнейшее обновление или переход к приложению. Для этого в энергонезависимой памяти существует область конфигурации загрузки. Конфигурация загрузки содержит флаг требования обновления, указывающий, что будет запущено – загрузчик или прикладная программа.

Порядок выполнения программы после сброса (рис. 2):

1. Запуск загрузчика, инициализация оборудования.
2. Если флаг требования обновления равен 0, флаг устанавливается в 1, продолжает выполняться загрузчик.
3. Если флаг требования обновления равен 1, то проводится валидация. Если комбинация флагов в конфигурации приложения позволяет определить приложение как поврежденное, то информация о нем стирается.
4. Если во флэш-памяти есть валидное приложение, производится его запуск. Если нет, продолжается выполнение загрузчика.
5. Если запуск приложения прошел успешно, то продолжается выполнение приложения. Если нет, то после сброса информация о приложении стирается на этапе валидации.

При выполнении приложения пользователь может ввести команду требования обновления. Тогда флаг требования обновления устанавливается в 0, и выполняется программный сброс.

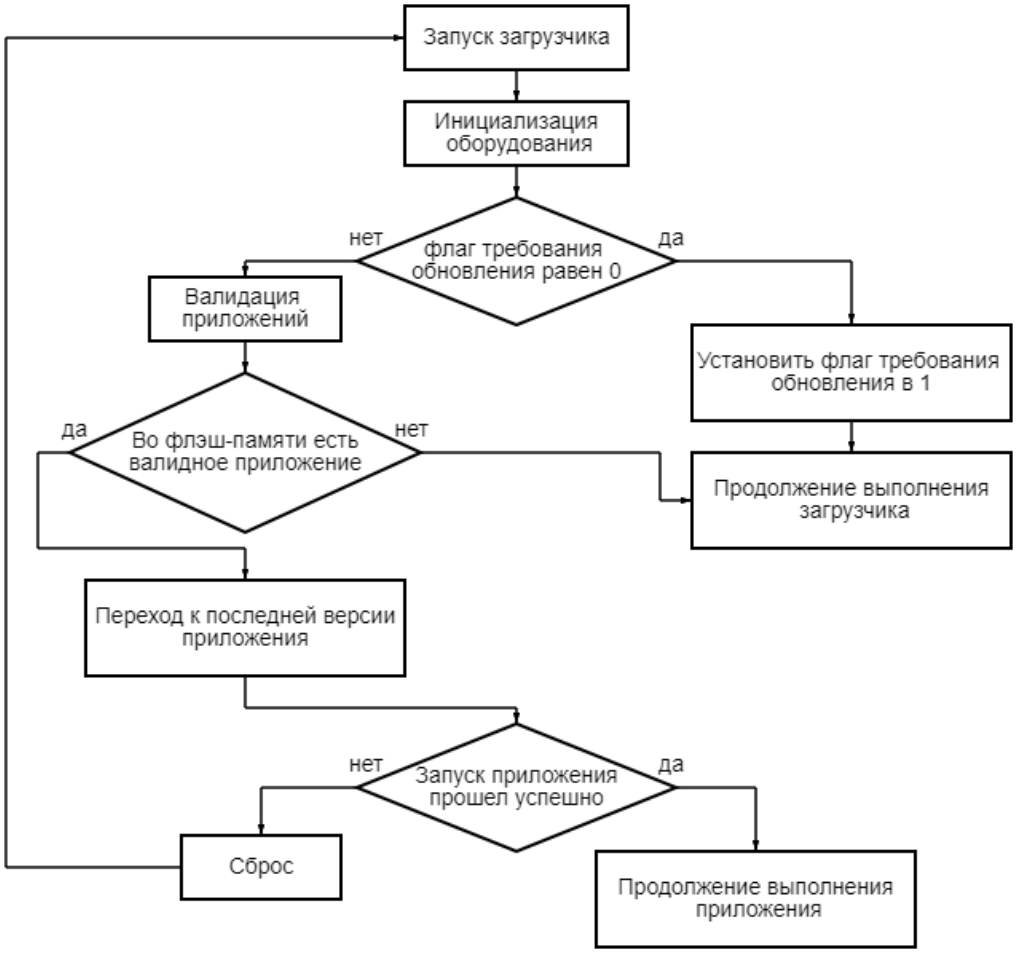

Рисунок 2 – Блок-схема поведения программы после сброса

Загрузчик поддерживает 5 команд: переход к приложению, обновление, получение версии, помощь, удаление приложений. Далее более подробно рассмотрена каждая из команд.

1. Переход к приложению (JUMP)

Осуществляет переход из загрузчика к последней актуальной версии приложения (рис. 3).

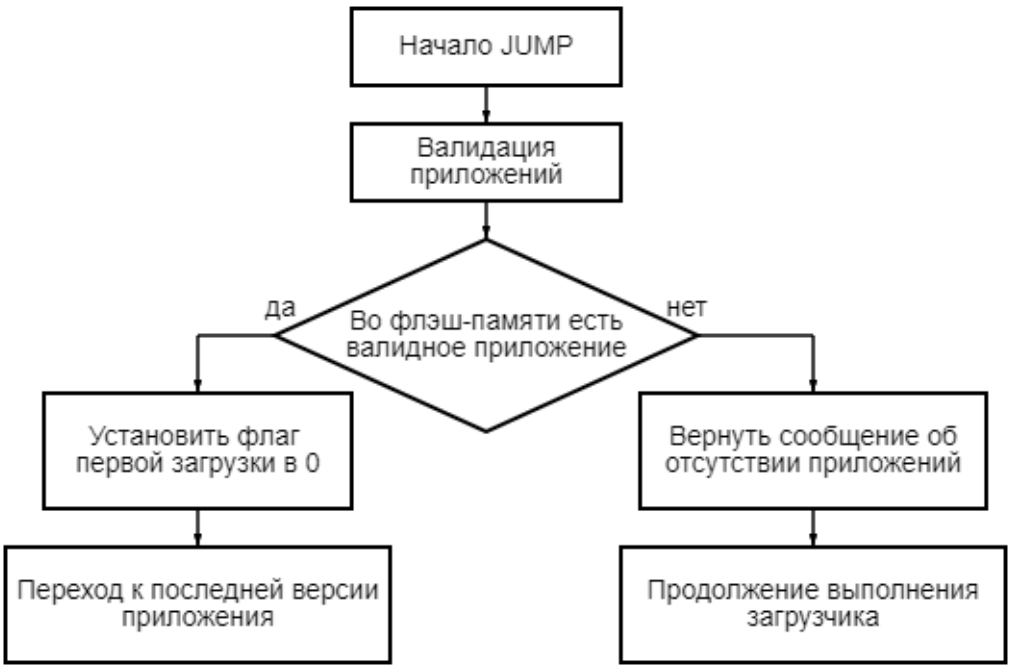

Рисунок 3 – Блок-схема перехода к приложению

Под валидацией приложений здесь и далее подразумевается стирание информации (конфигурации) о невалидных приложениях.

2. Обновление прошивки (UPDATE)

Осуществляет прием и запись нового приложения с последующим переходом на него (рис. 4).

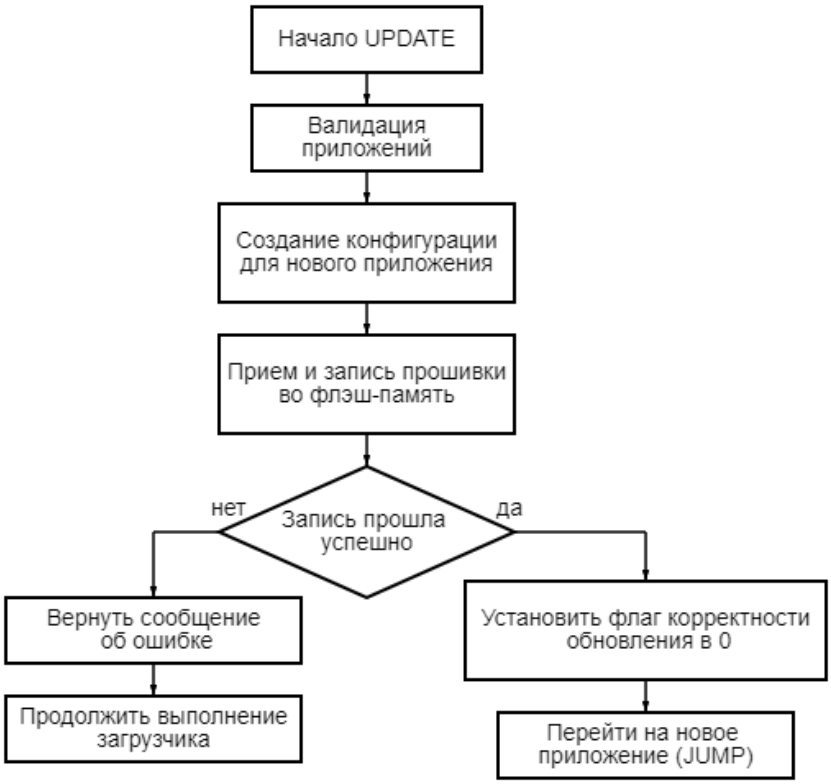

Рисунок 4 – Блок-схема обновления

Если уже есть два приложения во флэш-памяти, новая конфигурация записывается на месте конфигурации для более старого приложения.

3. Получение версии

Возвращает пользователю версию приложения, которая хранится в конфигурации приложения. Версия указывается пользователем при обновлении.

4. Помощь

Возвращает пользователю справку по доступным командам.

5. Удаление приложений

Стирает конфигурационные области всех приложений, таким образом удаляя информацию о них.

3 UART

Для передачи загрузчику команд на обновление, выполнение прикладного приложения, стирание приложений, получение версии приложений используются текстовые ASCII команды, пересылаемые по UART. Для передачи нового образа приложения используется XMODEM.

Пакет XMODEM содержит 133 байта: заголовок (1 байт), номер пакета (1 байт), дополнение к номеру пакета (1 байт), данные (128 байт), контрольная сумма (2 байта).

Пакет XMODEM обрабатывается следующим образом:

1. Проверяется заголовок пакета. В зависимости от заголовка либо происходит дальнейшая обработка данных, либо соединение закрывается.
2. Проверяется номер пакета. Если он не совпадает с нужным, посылается байт NACK, пакет передается заново.
3. Сравнивается переданная и рассчитанная контрольная сумма. Если они не совпадают, посылается NACK.
4. Данные (128 байт) передаются для записи прошивки. В случае успеха посылается байт ACK, передается следующий пакет.

Процесс приема прошивки с использованием XMODEM представлен в виде блок-схемы (рис. 5):

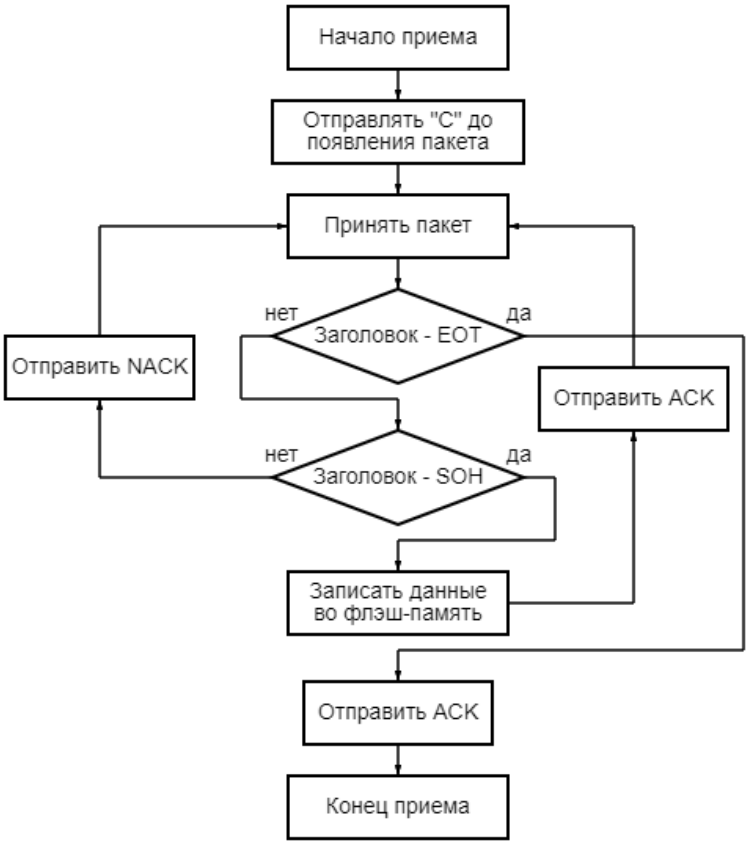

Рисунок 5 – Блок-схема приема прошивки с использованием UART

Для работы с загрузчиком по UART использовалась программа TeraTerm. Могут быть использованы и другие программы, поддерживающие работу через COM-порты и протокол XMODEM.

4 Ethernet

Для передачи загрузчику команд на обновление, выполнение прикладного приложения, стирание приложений, получение версии приложений и передачи новой прошивки используются шестнадцатеричные значения, пересылаемые в заголовках TCP пакетов.

При передаче команд клиент отправляет шестнадцатеричное значения и получает в ответ строку.

Поддерживаемые команды:

- JUMP (0x21) – переход к приложению
- UPDATE (0x63) – загрузить новое приложение и перейти к нему
- VERSION (0x02) – получить версии приложений
- HELP (0x10) – получить справку о командах
- CLEAR (0x43) – стереть приложения (конфигурации)

При передаче прошивки клиент отправляет пакеты размером 513 байт. Пакет обрабатывается следующим образом:

1. Проверяется заголовок пакета. Если значение заголовка равно EOT, происходит переход к приложению, соединение.
2. Данные (512 байт) передаются для записи прошивки. В случае успеха посылается байт ACK, иначе – NACK, принимается следующий пакет.

Процесс приема прошивки с использованием Ethernet представлен в виде блок-схемы (рис. 6):

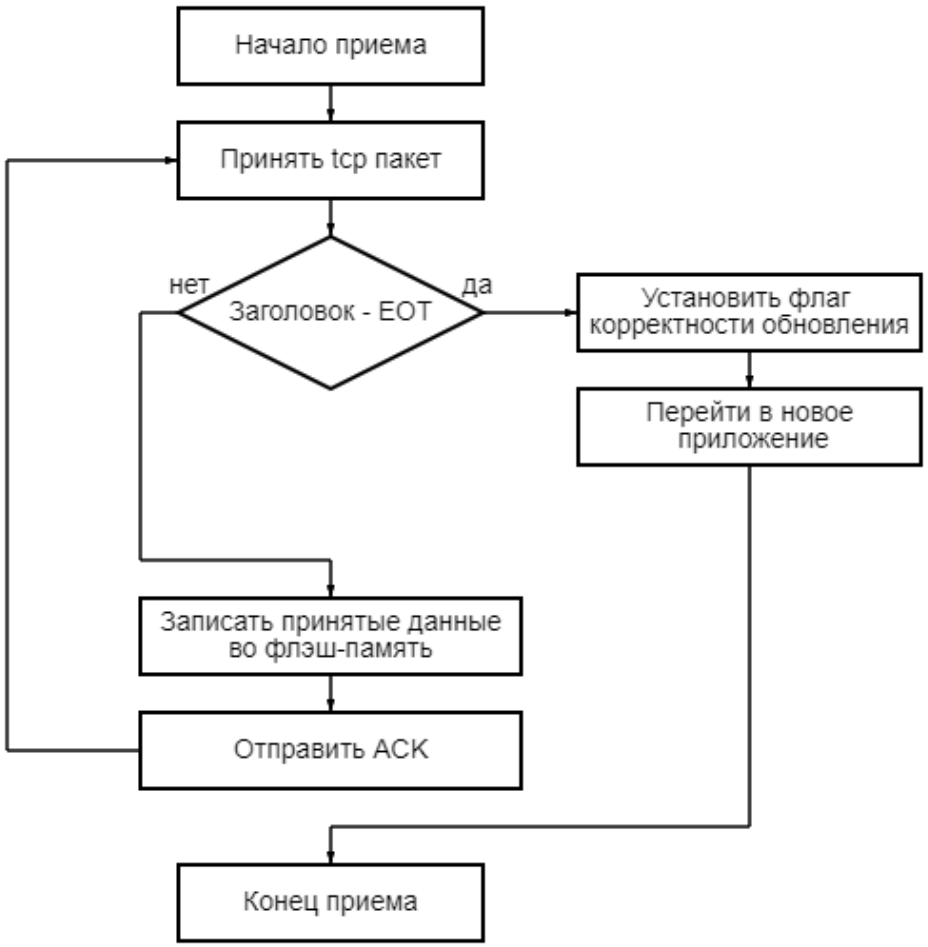

Рисунок 6 – Блок-схема приема прошивки с использованием Ethernet

Для работы с загрузчиком по Ethernet был разработан Python-клиент. Он устанавливает соединение с SDK-1.1M и в зависимости от введенных пользователем команд посылает пакеты с шестнадцатеричными кодами в заголовках. В теле пакетов может присутствовать дополнительная информация или передаваемая прошивка. Исходный код доступен по ссылке https://github.com/ales961/bootloader_eth_client.

5 Повторное использование

Для повторного использования разработанная программа должна быть использована в качестве шаблона. Тогда она может быть изменена и адаптирована другими пользователями. Исходный код доступен по ссылке https://github.com/ales961/bootloader.

В программе присутствует четыре изменяемых блока:

1. Архитектура

Файлы boot_config.c/.h содержат адреса расположения конфигурационных областей и приложений, а также логику их взаимодействия. Для изменения архитектуры программы необходимо редактировать эти файлы.

2. Поддержка интерфейсов для передачи данных

Файлы uart.c/.h, usart.c/.h, buffer.c/.h, command.c/.h, menu.c/.h, xmodem.c/.h используются для передачи данных по UART. Также часть функций для обработки принятых команд находится в main.c. Эти файлы необходимо редактировать для изменения работы по UART. При исключении поддержки UART будет сэкономлено 11.4 Кб флэш-памяти.

Файлы lwip_tcp.c/.h, tcp_protocol.c/.h используются для передачи данных по Ethernet. Их необходимо редактировать для изменения логики передачи пакетов или работы протокола для передачи данных по Ethernet. При исключении поддержки Ethernet будет сэкономлено 55.9 Кб флэш-памяти.

Загрузчик с поддержкой UART и Ethernet занимает 81.1 Кб флэш-памяти.

Для добавления поддержки нового интерфейса нужно написать драйвер для этого интерфейса, описать логику передачи данных и добавить вызов функции обработки для нового интерфейса в главный цикл.

3. Обработка принятой прошивки

На данный момент поддерживается обработка прошивки, полученной в формате Intel HEX. Для того, чтобы изменить логику разбора и записи прошивки, необходимо редактировать файлы hex_parser.c/.h.

5.1 Структура разработанной программы

Связи между модулями разработанной программы представлены в виде диаграммы (рис. 7).

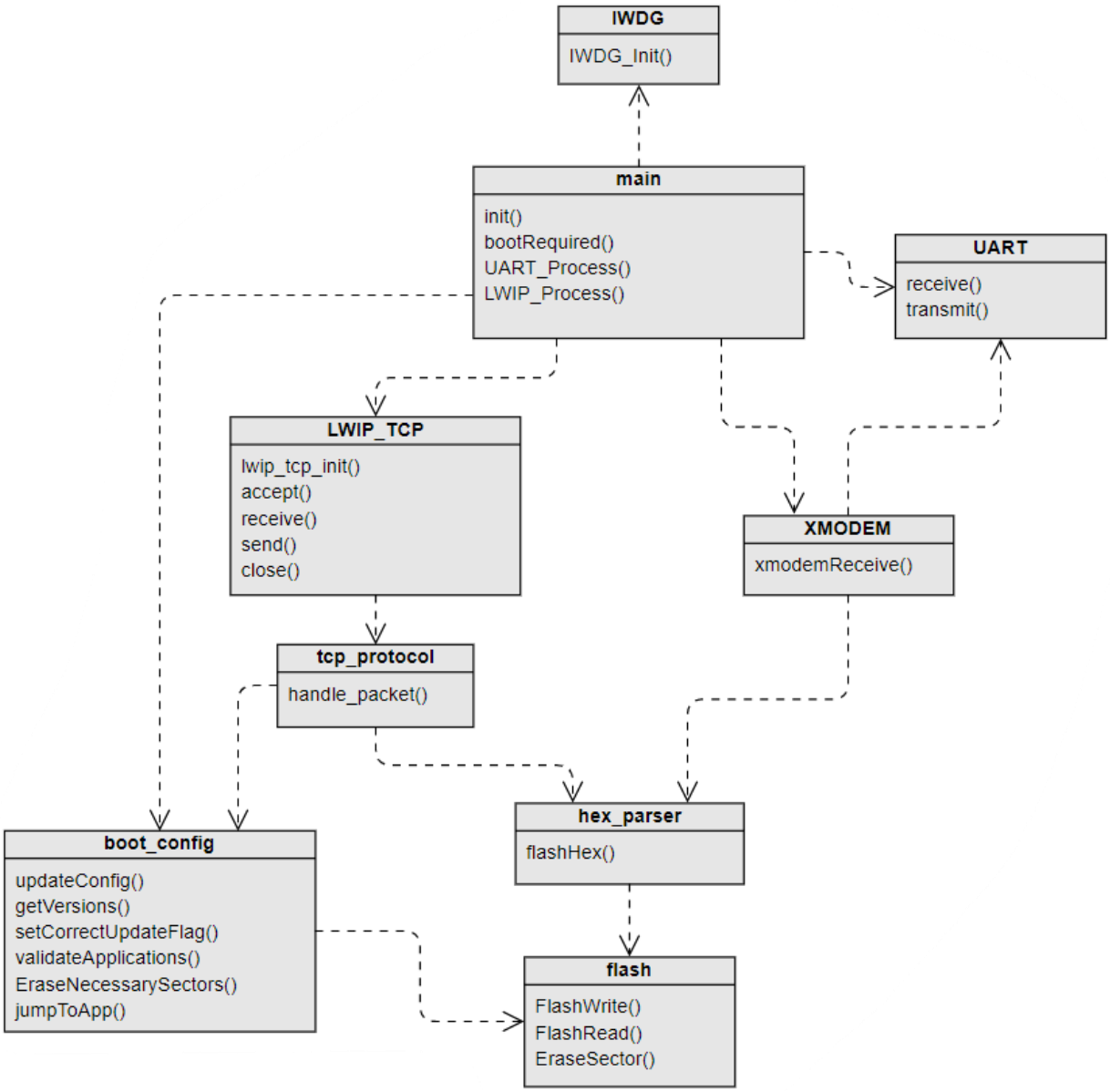

Рисунок 7 – Связи между модулями программы

1. Модуль main

Отвечает за инициализацию оборудования, выбор исполняемой программы (загрузчик или приложение) и обработку команд и пакетов, полученных по прикладным интерфейсам.

2. Модуль IWDG

Отвечает за инициализацию сторожевого таймера. Сторожевой таймер нужен для сброса в случае зависания прикладной программы.

3. Модуль UART

Используется для приема и передачи данных по UART.

4. Модуль XMODEM

Отвечает за прием прошивки по UART с использованием протокола XMODEM.

5. Модуль hex_parser

Реализует разбор файла формата Intel HEX, записывает прошивку в энергонезависимую память.

6. Модуль flash

Отвечает за работу с флэш-памятью (чтение, запись, стирание).

7. Модуль boot_config

Содержит информацию о размещении приложений, конфигурации приложений и конфигурации загрузки во флэш-памяти. Используется для создания/удаления конфигураций, получения информации о приложениях, валидации приложений и перехода к приложению.

8. Модуль LWIP_TCP

Отвечает за работу с TCP-соединением, передачу данных по Ethernet.

9. Модуль tcp_protocol

Реализует протокол для приема команд и прошивки по Ethernet.

5.2 Требования к прикладной программе

При разработке логики взаимодействия загрузчика и приложений было введено несколько механизмов для обеспечения бесперебойной работы. Использование этих механизмов подразумевает введение дополнительных требований к приложению.

1. Возможность перехода к загрузчику из приложения с использованием UART и Ethernet

При выполнении приложения должна быть возможность обновления. Для этого нужна возможность перехода к загрузчику по поддерживаемым прикладным интерфейсам. При этом нужно установить флаг требования обновления в 0, по умолчанию он находится по адресу 0x08104000.

2. Сброс сторожевого таймера

Для предотвращения зависания в приложении используется сторожевой таймер, который генерирует сигнал сброса по истечении определенного промежутка времени. Необходимо периодически сбрасывать сторожевой таймер. Достаточно добавить `WRITE_REG(IWDG->KR, 0x0000AAAAU)` в главный цикл.

3. Запись нужного значения во флаг валидности

При первом запуске приложение помечается как невалидное. По истечении определенного промежутка времени приложение должно записывать 0 по адресу флага валидности в конфигурации этого приложения. Флаг валидности по умолчанию находится по адресу 0x08108008 для первого приложения, 0x0810C008 – для второго.

4. Два варианта файла прошивки с разными начальными адресами

По умолчанию hex файл прошивки генерируется под адрес начала флэш-памяти. Для того, чтобы приложение могло записаться по нужному адресу, перед генерацией в файлах линковки нужно изменить начальный адрес на тот, который указан в boot_config.h. Таким образом, получится два файла под каждую из областей хранения приложений. По умолчанию для получения нужного файла прошивки перед генерацией изменяется поле FLASH ORIGIN секции MEMORY в файлах линковки на 0x8020000 (для первой области приложения) или 0x8110000 (для второй).

5. Размер

По умолчанию приложение может занять всю флэш-память. Поскольку во флэш памяти хранятся три конфигурационные области, загрузчик и два приложения, появляется ограничение на размер приложения. В файлах линковки (поле FLASH LENGTH секции MEMORY) нужно изменить размер доступной флэш-памяти на 912 Кб (для размера флэш-памяти 2Мб).

Примеры прикладных программ доступны по ссылке https://github.com/ales961/bootloader_app_examples.