Санкт-Петербургский государственный университет

Кафедра системного программирования

Кижнеров Павел Александрович

# Организация автоматизированного тестирования встраиваемого программного обеспечения

Магистерская диссертация

Научный руководитель:
д. ф.-м. н., профессор Терехов А. Н.

Рецензент:

Санкт-Петербург
2021

# Оглавление

# Введение

Данная работа выполняется в рамках разработки прошивки для новой версии фитнес-браслетов, разрабатываемых американской компанией-производителем для продажи на американском и мировом рынках. Аппаратная часть состоит из двух процессоров, модуля Bluetooth Low Energy (далее BLE), датчиков и дисплея. Такая связка на протяжении всего времени должна работать как единое целое, потребляя при этом минимум энергии и предоставляя максимальную степень удобства пользователю. Качество конечного продукта в значительной степени определяется качеством встраиваемого программного обеспечения, поэтому важно обеспечить процесс автоматического тестирования и мониторинга результатов. Работа с платформозависимым кодом, исполняемым на контроллере, имеет следующие специфические особенности:

- запуск кода возможен только после загрузки скомпилированного бинарного исполняемого файла в память контроллера; это увеличивает трудоемкость отладки и в целом снижает концентрацию внимания программиста, так как это монотонный и скучный процесс;

- мониторинг результатов осуществляется посредством чтения последовательного порта, через который контроллер подключен к инструментальной машине; это снижает наглядность отладки и усложняет процесс разработки.

# 1. Постановка задачи

В конечном итоге для автоматизации контроля качества продукта планируется разработка специального инструмента в составе тестового стенда (связка 2 контроллеров - коммуникационного и измерительного, модуля BLE и периферийных датчиков) , обладающего следующей функциональностью (пункты упорядочены по важности):

- автоматический запуск модульных тестов из инструментальной машины;
- автоматический запуск интеграционных тестов из инструментальной машины;
- мониторинг результатов выполнения любых тестов;
- взаимодействие с службой непрерывной интеграции;
- анализ результатов тестов;
- подсчет метрик кода;

# 2. Обзор

Для модульного и интеграционного тестирования проектов, написанных на языках C/C++, используются такие фреймворки как Gtest, Catch, Mettle, Boost.Test и все эти решения похожи между собой с точностью до особенностей конфигурации и синтаксиса тестирующих функций. Также в данных фреймворках есть возможность генерировать отчеты в формате XML, с которым работают службы непрерывной интеграции (прим. Jenkins). Связка тестирующего фреймворка и службы непрерывной интеграции обеспечивает автоматизацию процесса запуска тестов и мониторинга результатов, однако это применимо только если на сервере непрерывной интеграции есть возможность собирать и запускать тестируемый проект на целевой архитектуре и нет ограничений, связанных с потреблением вычислительных ресурсов.

Контроллеры обладают ограниченными вычислительными возможностями по сравнению с полноразмерными компьютерами, поэтому при разработке встраиваемого программного обеспечения особое внимание уделяется потреблению ресурсов. Также контроллеры имеют возможность переходить в различные режимы энергопотребления, динамически изменять аппаратные параметры, подключать / отключать периферийные устройства, что может повлиять на работоспособность встраиваемого программного обеспечения. Ввиду данных обстоятельств, тестирование аппаратнозависимых модулей как правило производится под управлением инструментальной машины - таким образом достигается приближенность к реальным условиям использования контроллера и расширение спектра сценариев тестирования.

В каждом проекте уникальные требования к разрабатываемому встраиваемому программному обеспечению, тестирующему окружению, аппаратой конфигурации тестового стенда, поэтому, часто готовых комплексных решений для автоматизации тестирования встраиваемого обеспечения нет и для каждого проекта оно либо полностью, либо частично реализовывается.

Как правило используются готовые инструменты генерации ком-

плексных отчетов. Популярные решения: TETware RT, OpenTest, autotestnet, DejaGnu, Robot Framework. TETware RT и OpenTest требуют UNIX окружение, в DejaGnu отсутствуют встроенные средства сверки ожидаемого и полученного результата, autotestnet требует windows окружение и имеет проблемы с доступом к документации, Robot Framework мультиплатформенный, но требует реализацию интерфейса коммуникации с контроллером (как и все остальные решения).

# 3. Реализация

## 3.1. Декомпозиция

### 3.1.1. Автоматический запуск модульных тестов из инструментальной машины

Предполагается что инициировать запуск модульного теста и собирать результат будет программное обеспечение на инструментальной машине. Для этого нужен надежный двунаправленный канал между тестовым стендом и машиной. На тестовом стенде есть BLE и три интерфейса UART.

BLE нельзя использовать по нескольким причинам: инициализация BLE требует времени, а требуется тестировать модули, которые инициализируются гораздо раньше BLE; BLE подключен только к коммуникационному контроллеру, поэтому нельзя тестировать измерительный контроллер изолированно; предполагается тестирование самого BLE, а значит может быть взаимовлияние; в офисе много устройств, работающих на одной частоте с BLE, а значит канал связи подвержен интерференциям.

Для коммуникации подходит только UART, но использовать можно только один (тот что через USB) из за требования приближенности тестового стенда к конечному устройству и сочетаемости с отладочным интерфейсом. Поэтому для организации удаленной отладки необходимо необходимые данные среди всего потока байтов единственного последовательного порта.

Отдельной подзадачей является реализация способа инициализации запуска необходимых тестируемый функций

### 3.1.2. автоматический запуск интеграционных тестов из инструментальной машины

Эта задача является естественным расширением задачи автоматизации запуска и мониторинга модульных тестов, так как имея способ инициализации запуска необходимых тестируемый функций, можно писать

тестовые наборы, затрагивающие сразу несколько независимых модулей прошивки

### 3.1.3. мониторинг результатов любых тестов

В качестве каркаса ПО для инструментальной машины можно использовать один из распространенных фреймворков для тестирования, позволяющих генерировать комплексные отчеты по результатам модульных и/или интеграционных тестов. TETware RT, OpenTest, autotestnet, DejaGnu, Robot Framework - основные фреймворки для подобных задач. Вне конкуренции оказался Robot Framework ввиду простоты написания (язык - python), кроссплатформенности, автоматическим документированием и богатой инфраструктуры. Некоторые из других рассмотренных решений либо требуют unix окружение (TETware RT, OpenTest), либо в них отсутствует встроенное средство проверки результатов (DejaGnu), либо имеют слабую документацию.

### 3.1.4. интеграция с сервисом непрерывной интеграции

Ввиду кроссплатформенности выбранного фреймворка и популярности языка, написать скрипты для сценариев Jenkins (основное CI/CD средство инфраструктуры заказчика) не составит большого труда Декомпозиция оставшихся задач будет сделана после реализации плана-минимума (первые 4 пункта)

## 3.2. Реализация

### 3.2.1. Автоматический запуск модульных тестов из инструментальной машины

Для двунаправленной коммуникации используется интерфейс UART. По нему пересылаются пакеты определенного формата, которые легко определить в потоке.

```
typedef struct message
{
    uint8_t sig[4]; /**< Message signature */
    int32_t id; /**< Message id*/
    uint32_t payload_size; /**< Payload size */
    uint32_t checksum; /**< Crc32 checksum */
    uint32_t domain_receiver; /**< Receiver domain number */
    uint32_t action_receiver; /**< Receiver domain action */
    uint32_t domain_sender; /**< Sender domain number */
    uint32_t action_sender; /**< Sender domain action */
    uint8_t payload[0]; /**< Payload buffer */
} message;
```

В языке Си поля структуры располагаются в памяти последовательно, поэтому в структуре пакета первым полем добавлена преамбула (поле sig) - по ней пакет определяется среди потока байтов. Далее идут поля с метаинформацией для корректной обработки и маршрутизации пакетов. В самом конце непосредственно сама полезная информация. Приемом, передачей и маршрутизацией пакетов управляет брокер сообщений - сущность, реализованная как на стороне тестового стенда так и на инструментальной машине. Для минимизации количества ошибок в процессе разработки такая сущность должна быть идентичной на обоих концах коммуникации. Единственный язык реализации на стороне тестового стенда - Си. На стороне инструментальной машины язык реализации неявно навязан Robot Framework - python. Имея в виду сложившиеся обстоятельства, был реализован модуль на языке python,

который посредством библиотеки ctypes оборачивает реализацию брокера на Си. Также был реализован еще один модуль, предоставляющий высокоуровневый интерфейс коммуникации с тестовым стендом и использующий модуль-обертку.

В данный момент реализуется таблица соответствия тестируемых функций на тестовом стенде и вызывающих функций на стороне инструментальной машины. Также уже была опробована связка реализованных модулей с Robot Framework в контексте автоматического вызова некоторых тестируемых функций прошивки.

# 4. Планы на будущее

В рамках данной работы планируется реализовать:

- взаимодействие с службой непрерывной интеграции;
- анализ результатов тестов;
- подсчет метрик кода;

# Список литературы