Разработка USB устройства ввода в Linux

Михаил Белкин

23 сентября 2021 г.

Содержание

1. Введение **3**

2. Схемотехника. **4**
2.1. Выбор редактора схем. 4
2.2. Установка редактора и библиотек. 5
2.2.1. Установка редактора 5
2.2.2. Установка библиотек 5
2.3. Выбор элементной базы. 6
2.3.1. Микроконтроллер 6
2.3.2. Стабилизатор 7
2.3.3. Мелочь 8
2.4. Особенность схемотехники USB. 9
2.5. Схема устройства. 9

3. Программное обеспечение. **11**
3.1. Тулчейн. 12
3.2. Что нового в CMSIS5. 12
3.2.1. Скачивание CMSIS5 14
3.3. Компоновщик. 14
3.3.1. разметка памяти 14
3.3.2. точка входа 15
3.3.3. определения секций 15
3.4. startup файл 17
3.5. Makefile 18
3.6. Структура проекта. 18
3.7. Опрос кнопок. 18

4. USB (универсальная последовательная шина) **19**
4.1. Краткий обзор технологии USB 20
4.2. Периферия USB. 22
4.2.1. Включение 22
4.2.2. Буффер USB STM32 23
4.2.3. Ендпоинты 24
4.3. Дескрипоры USB. 24
4.3.1. Device дескриптор. 25
4.3.2. Configuration дескриптор. 25
4.3.3. Interface дескриптор. 25
4.3.4. Endpoint дескриптор. 25
4.3.5. String дескриптор. 25

4.3.6. Report дескриптор. 25
4.4. Стандартный протокол USB. 25
4.4.1. getStatus дескриптор. 26
4.4.2. clearFeature дескриптор. 26
4.4.3. setFeature дескриптор. 26
4.4.4. setAddress дескриптор. 26
4.4.5. getDescriptor дескриптор. 26
4.4.6. set Interface 26
4.5. USB HID 26
4.5.1. HID реквесты 26

5. Отладка. **28**

1. Введение

USB достаточно свежая технология (USB 1.1 - 1998г., USB 2.0 - 2000). Задумывалась как универсальная шина для компьютерной периферии. И в реальности так оно и оказалось, почти все устройства, даже те что раньше были исключительно встроенными теперь можно купить и в USB варианте, например, сетевую карту или звуковую карту. Да и вообще на сегодняшний день сложнее найти клавиатуру без USB разъема, чем с ним, хотя для клавиатур и мышек когда то задумывался отдельный стандартный разъем. Я же остановлюсь на чем попроще и соберу геймпад. В общем USB вытеснил практически все, его разъем даже больше запомнился как стандартный разъем питания на 5В, чем коаксиальный разъем.

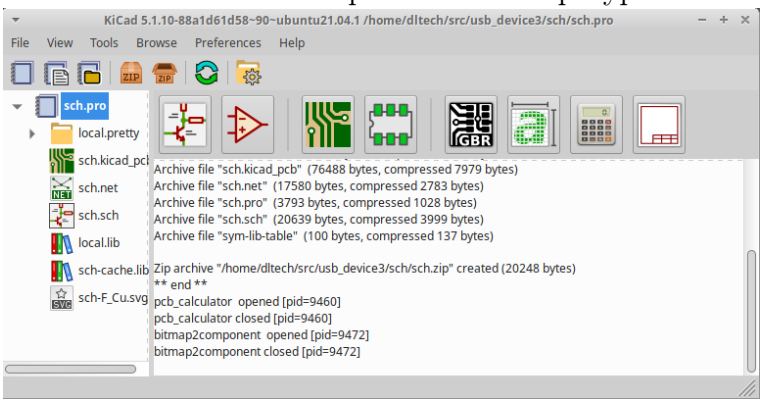
К сожалению, я использую всего лишь USB 2.0. В то время, как на данный момент уже вышла четвертая версия, чьи скорости превышают SATA в былые времена (40 Гбит/с). Но не стоит так сильно беспокоиться, ведь разъёмы до третьей версии USB обратно совместимы. Также нельзя не упомянуть и о существовании беспроводного USB, которого я практически нигде не видел, но рано или поздно мы попробуем и эту технологию.

А данная статья, в отличии от остальных рассматривает процесс разработки всесторонне. Также нужно сказать что я не ставил цели написания подробной документации на все те программы, которые я использовал. Их вы можете найти по ссылкам в списке литературы, в данной статье только комментарии к коду.

2. Схемотехника.

2.1. Выбор редактора схем.

Разработка схемы производится в KiCAD, это очень легковесный и компактный opensource редактор. Но при этом, несмотря на его внешнюю простоту, редактор как будто бы кричит нам "Я ничем не хуже чем этот ваш Altium и уж тем более Eagle". Поддерживается редактирование многослойных плат, так же используется профессиональный подход, при котором схема устройства и печатная плата редактируются отдельно. Так же он очень нетребователен к ресурсам компьютера.



К редактору имеется собственная библиотека компонентов, которая включает в себя компоненты всех популярных производителей, всякие там ST, Ti, можете даже не искать. Но если вы принесли с китайского базара какую то экзотику, компонент придется разработать самостоятельно.
Вывод шаблона печатной платы возможен во всех удобный форматах, включая pdf. Так же и Gerber и векторный svg, последнее очень удобно для печати шаблона на принтере. Единственное неудобство это не возможность сразу выводить схему в растровом формате, приходится самостоятельно конвертировать из svg.
Логично предположить, что данный редактор используется в очень крутых проектах. Таких как rusEFI - универсальное ЭБУ для автомобилей, или olinuxino - открытый аналог малинки на дешевом китайском чипе от allwinner с кучей выведеннных портов на шилд.
В следующем разделе подробно рассказано об скачивании и установке редактора. А скачав и установив его вы можете ознакомится со схемой и платой моего геймпада, который находится в папке проекта.

2.2. Установка редактора и библиотек.

2.2.1. Установка редактора

Установка редактора в Ubuntu Linux производится очень просто, имеется отдельный ppa репозиторий с последней стабильной версией. В то время как наиболее полный набор библиотек компонентов можно скачать с гитлаба.
Набор команд для установки KiCAD:

```
sudo add-apt-repository --yes ppa:kicad/kicad-5.1-releases
sudo apt update
sudo apt install --install-recommends kicad
```

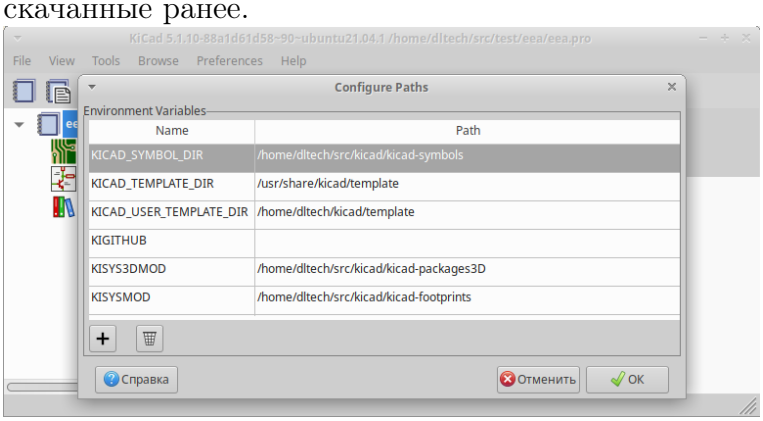
2.2.2. Установка библиотек

KiCAD и библиотеки ужасное сочетание. Те что в репозитории по умолчанию не самые полные, актуальное состояние, как всегда, отражает git проекта. Безусловно, их можно скачать и добавить в местном проекте, но вот на на еще одном компьютере, как всегда, будут проблемы. То есть универсального решения попросту нет. Ведь и самое главное что для каждого проекта символы то он кеширует, а вот футпринты почему то не хочет. Тем не менее, попытаемся написать инструкцию по добавлению библиотек с git.
Итак, до зимы 2020 года библиотеки базировались на гитхабе, сейчас переехали на гитлаб. И если у футпринтов формат не поменялся, то символы теперь уже адаптированы под еще не релизный шестой кикад. Поэтому символы качаем зимние, футпринты сегодняшние. Надеюсь устанавливать git вы умеете и про команду git clone вы тоже знаете. Вот ссылки на репозитории с библиотеками компонентов KiCAD:
`https://github.com/kicad/libraries/kicad-symbols`
`https://gitlab.com/kicad/libraries/kicad-footprints`
`https://gitlab.com/kicad/libraries/kicad-packages3D`
Скопируйте их все куда нибудь, что бы не мешались, например:

```
cd ~/src/kicad
git clone repo.git
```

Теперь, что бы поставить новые библиотеки так, что бы не мешались старые нужно удалить все содержимое папки /.config/kicad. При том условии, что кикад успел её создать. Дальше заменим пути до стандартных библиотек путями на новые. Глобальные пути редактируются через контекстное меню. Вкладка Preferenses->Configure Paths... Заменяем пути

в переменных KICAD_SYMBOL_DIR, KISYSMOD и KISYS3DMOD на пути до репозиториев kicad-symbols, kicad-footprints и kicad-packages3D, скачанные ранее.



А потом обновляем таблицы компонентов. При первом открытии вкладки Preferenses->Manage Symbol Libraries... он должен предложить меню, в котором можно выбрать добавление уже существующей таблицы (средний пункт), по кнопке ниже выбираем путь до файла /src/kicad/kicad-symbols/sym-lib-table. Для футпринтов аналогично: Preferenses->Manage Footprint Libraries..., /src/kicad/kicad-symbols/fp-lib-table.
Вот теперь вы можете создавать новые проекты и открывать существующие, имея самую полную библиотеку компонентов.

2.3. Выбор элементной базы.

2.3.1. Микроконтроллер

Для наиболее аккуратной реализации нужен современный микроконтроллер (МК) с полноценным аппаратным USB. Совершенно понятно, что таким микроконтроллером окажется STM32F103C8T6. Мощное ядро ARM Cortex-M3 с их фирменным вложенным векторным контроллером прерываний (NVIC) позволит с легкостью справиться с любой задачей. А с такой простой как USB геймпад уж тем более. На борту имеется 64 килобайта FLASH и 20 килобайт SRAM. И этого настолько много, что можно вовсе не думать об оптимизации. Теперь о стоимости, когда то я покупал такой за 60 рублей, сейчас цена приблизилась к 200, что по прежнему сравнимо по стоимости с остальными морально устаревшими микроконтроллерами. Так же в пользу данного микроконтроллера говорит наличие подробной документации. О том, почему именно F103, тут все просто, это самый дешевый МК с USB из тех что может предложить компания ST microelectronics. Продолжу перечислять его преимущества:

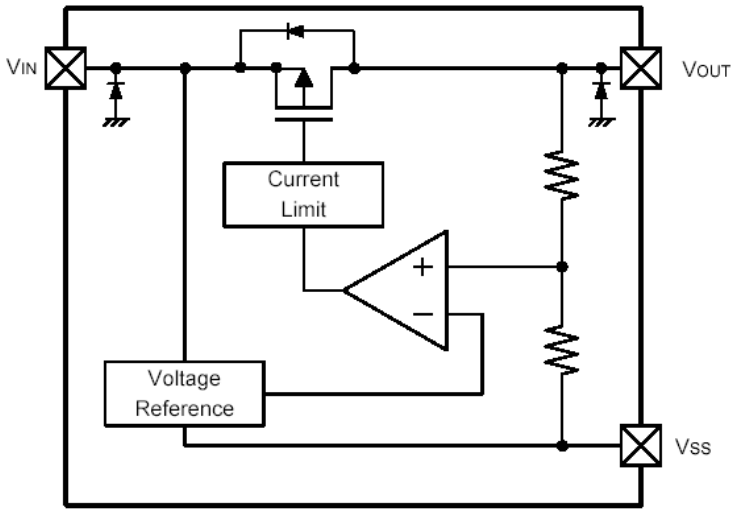
- Очевидно, 72 мегагерца на 32 разряда.
- Гибкая система тактирования, позволяющая отключать всю периферию для экономии батарейки.
- Поддержка интерфейса SWD и JTAG (прошивка по трем проводам).
- А также возможность установки бутлоадеров, один из которых позволяет обновлять прошивку по USB (поддержка протокола DFU).
- Прерывание, АЦП, выход таймера чуть ли не на каждом порте ввода вывода, что упрощает разводку ПП.
- DMA (direct memory access), по задумке должен упрощать работу с периферией, но на практике редко пригождается.
- Три 16 битных навороченных таймера счетчика.
- Возможность настройки таймеров как для аппаратной поддержки енкодеров, так и для различных шим режимов для шаговых двигателей.
- Часы реального времени, которые считают даже год.
- 2 АЦП со встроенным калиброванным источником опорного напряжения, и минимальным временем измерения в 1 мкс.
- А также встроенный датчик температуры.
- Много прочей стандартной периферии МК (SPI, I^2C, CAN, UART)

2.3.2. Стабилизатор

В шине USB, как известно, 5В, а номинальное напряжение питания МК 3.3В. Поэтому необходим понижающий стабилизатор напряжения. Я рассматривал три марки стабилизаторов. Они приведены в таблице ниже:

стаб	U_{inmax}	корпус	производитель	особенности
L78L33	30	SOT-89	ST microelectronics	
AMS1117-3.3	15	SOT-223	AMS semitech	термозащита
XC6206-33	7	SOT-23	TOREX	CMOS

И если первый давно знаком многим радиолюбителям. То последние два это стабилизаторы от китайских производителей, которые появились недавно. В целом гораздо больше доверия к старому 78l, как минимум из за его большого входного напряжения. К тому же AMS1117 мне попадались нерабочими, и очень легко пробивались от скачков напряжения, не спасая нагрузку. Но хотелось бы компактней и подешевле, к тому же компьютер сам по себе стабильный источник питания. Поэтому я выбрал XC6206. Довольно необычный новодел на полевых транзисторах, в то время как другие два на биполярных. Его КМОП структура улучшила такой параметр, как минимальное падение напряжения, которое всего 0,3В против 1,2В у транзисторных. Также в нем имеется встроенная защита по току, чип отключает нагрузку, когда ток превышает 200 мА. Ниже приведена его структурная схема, на которой видны еще и защитные антистатические стабилитроны. А вот защиты от перегрева у него не обнаружено, хотя 1117 что то подобное обещали.



Такие стабилизаторы не редкость и часто встречаются на дешевых модулях для ардуино.

2.3.3. Мелочь

На форумах можно услышать совет не ставить кварц в целях экономии. Но в случае с асинхронной шиной кварц нужен для стабильной работы устройства. Микроконтроллер можно настроить на работу от самого распространенного кварца на 8мГц. Не удивительно, что на алиэкспресс сразу же нашелся не только планарный, но и очень компактный вариант. Размером как 1206 чип резистор.
Резисторы размером 0402 я успел заказать заранее. А вот шунтирующие

конденсаторы пришлось выпаивать с донорских плат, потому они не такие компактные, как хотелось бы (0805).
Расчетный ток потребления десятки миллиампер, потому для стабилизации питания хватит и чип керамики, благо такая есть даже на 10 мкФ. На всякий случай установлю токоограничивающий резистор по питанию. Основная его цель обезопасить компьютер от случайного короткого замыкания. Хотя в дорогих флешках на его месте можно встретить чип предохранитель. Продолжу экономить и на разъемах, попросту ограничусь площадками под проводки.

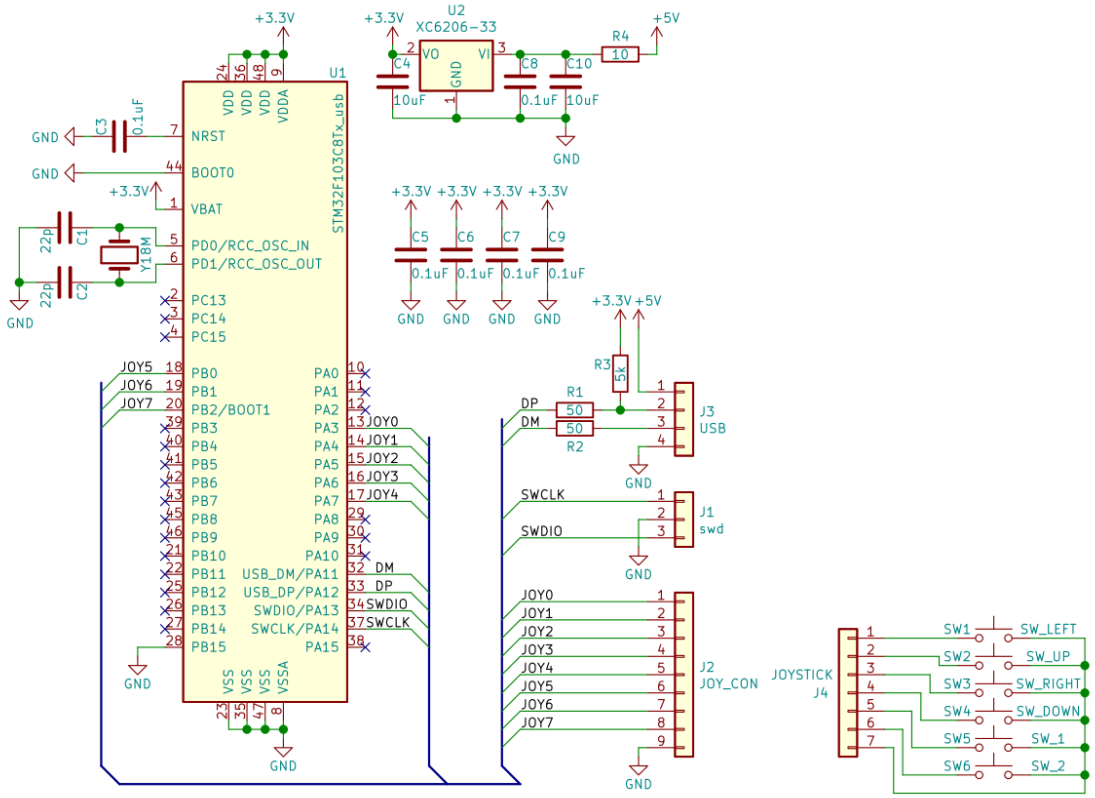
2.4. Особенность схемотехники USB.

На сайте можно скачать целый документ, посвященный вопросу распайки USB разъема. Основной вопрос заключается в возможности программного отключения устройства от ПК. По стандарту наличие USB устройства на шине определяется подтянутым к 3.3В портом DP. В версиях STM32 с USB OTG этот резистор встроенный, можно включать и отключать программно. У тех что просто USB Device встроенного резистора нету. Поэтому на отладочных платах ST устанавливают управление внешним подтягивающим резистором через транзистор, подключенный к порту ввода вывода. Моё же устройство будет всегда включено, поэтому подтяжка линии DP к питанию будет осуществляться внешним резистором. Также важно не забыть про защитные токоограничивающие резисторы, включенные последовательно на шинах DM и DP. Провод у меня используется готовый от клавиатуры, потому разъем на плате не нужен. Также в дорогих устройствах часто можно встретить защитные антистатические стабилитроны на шине USB. В моем случае экономим и на них.

2.5. Схема устройства.

А вот и схема целиком, как видите, все шины питания подключены и заземлены фильтрующими конденсаторами. Кварц с нагрузочными конденсаторами в наличии, так же подтянут к земле и порт сброса. Все как советует официальная документация. Кнопки джойстика, как видно, подключены к портам напрямую, т.к. внутри МК уже имеются резисторы подтяжки к 3.3В. Плата компактная и помещается внутри корпуса, потому дополнительные шунты и защиты не нужны, просто порт. Так же не обошлось и без так называемого ”грязного хака“, для упрощения трассировки платы один из портов ввода-вывода (28 вывод) использован как вывод земли. Но в этом нет ничего плохого, ведь порты после сброса

находятся в состоянии с высоким входным сопротивлением.
Так же на схеме вы можете увидеть и стандартный разъем SWD для прошивки и отладки ПО. Ровно как и подключенный на землю вывод BOOT0 означает запуск прошивки из основной flash памяти.



3. Программное обеспечение.

Если по началу, когда эти чипы только появились. Выбора способов написания ПО для них было немного. Как правило это были коммерческие среды среды программирования под Windows, довольно тормозные и требовательные продукты. Которые студенты и любители скачивали в месте с так называемой таблеткой от жадности. Теперь же выбор бесплатных инструментов очень большой. Начиная с официальной среды от компании ST, STM32Cube. Которая может не только скачать сама все их библиотеки и примеры, но и имеет автоматический конфигуратор периферии (это такая штука, которая предлагает настраивать регистры, тыкая мышкой). И заканчивая сторонними продуктами, таким как AC6 system workbench. И эти новые инструменты, в отличии от старых, уже кроссплатформенные, и как это принято говорить на английском, легковесные. Например AC6 основан на eclipse. Так же на гитхабе полно простых устройств, собранных на одном Makefile. А ещё сообщество создало полноценную библиотеку libopencm3 с кучей примеров.
С прошивальшиком проблем нет, протокол задокументирован, под linux давно уже создана st-link utility. Можно даже в отладку, без чего, в случае возни с периферией не обойтись. По части периферийных библиотек, доступны как варианты от ST так и от проекта libopencm3, тут даже есть выбор.
Но исходя из того, что пример данного устройства довольно простой. Напишем весь код полностью самостоятельно. То есть просто представим, что сейчас 2007 год и мы только что обнаружили новый чип, на который нету ни сред ни примеров.
Можно сказать, что программа для ПК и для микроконтроллера ничем не отличается. Но на самом деле программы, которые возможно вы писали в школе на информатике ориентированы под запуск в операционной системе. И ОС значительно упрощает и саму программу и её запуск. В то время МК это просто процессор, у которого ничего такого нет. Поэтому вы должны самостоятельно описать для него все. Процесс запуска программы, разметку памяти, даже компилятор и тот придется запустить самостоятельно, попросить у него сохранить прошивку в правильном формате и самостоятельно доставить её в микроконтроллер. А также возможно вам придется самостоятельно взаимодействовать с ядром.

3.1. Тулчейн.

GCC, нисмотря на количество букв Си в этом названии, это не компилятор языка Си, а коллекция компиляторов проекта гну. То есть в этой коллекции есть много разных компиляторов, в том числе и компилятор языка фортран например. Когда то просто скачивание и установка gcc была поводом для отдельного поста. Я же отмечу только то, что gcc признает контроллеры на ядре Cortex M3. И для этих МК в ubuntu устанавливается вот такой вот командой:

```
apt install gcc-arm-none-eabi libnewlib-arm-none-eabi
apt install gdb-multiarch
apt install build-essential
```

А наши makefile и скрипты прошивки будут их искать потом как arm-none-eabi-gcc, arm-none-eabi-objcopy и gdb-multiarch. В остальном даже флаги компиляции в случае с данными микроконтроллерами ничем особенно не отличаются. Слово arm в названии пакета, говорит о том, что компилятор версии для процессоров arm. А none-eabi означает, что компилятор для прошивок, а не для операционных систем.
Все же разберем стандартные для любого проекта флаги:
-ffunction-sections, -fdata-sections - позволит оптимизатору не добавлять в прошивку неиспользуемые функции
-Wall -Wextra -Werror -Wconversion -Wundef -Wformat=2 -Wformat-truncation -Wdouble-promotion -Wshadow -Wpadded -Wimplicit-function-declaration -Wredundant-decls -Wstrict-prototypes -Wmissing-prototypes - ворнинги, нужно включить все, какие только возможны, понятное дело, что в пяти килобайтах код должен быть вылизан полностью
-fno-common - выдает ошибку если одна и та же глобальная переменная объявлена дважды.
-Os - логично, флеша у нас не террабайт, экономим
-mcpu=cortex-m3 -mthumb - обязательно скажем, какой там процессор
-ffreestanding - не подключаем ненужные функции
-ggdb3 - данный параметр нужен для возможности отладки, перед релизом надо не забыть его удалить, иначе в прошивке будет много лишних данных

3.2. Что нового в CMSIS5.

CMSIS это слой абстракции, независимый от производителя. Это не просто библиотека, а стандарт взаимодействия с ядром ARM. В библиотеке определены главные интерфейсы для инструментов, а также она обеспечивает согласованную поддержку устройств.

CMSIS обеспечивает интерфейсы для процессоров и периферии, операционных систем реального времени, и компонентов промежуточного программного обеспечения. CMSIS включает в себя механизм доставки для устройств, плат, и программ, и позволяет комбинировать программные компоненты от разных поставщиков.
Важно заметить, что с появлением последней пятой версии она отличается от предыдущих, была дополнена и исправлена. Структура проекта CMSIS 5:

- Core(M) - стандартизированный API для процессоров Cortex-M и периферии. Включает в себя внутренние функции для Cortex-M4/M7/M33/M35P SIMD инструкций.

- Core(A) - стандартизированный API и базовая система выполнения для Cortex-A5/A7/A9 процессоров и периферии

- Driver - основные интерфейсы периферийных драйверов для промежуточного слоя. Соединяет периферию микроконтроллера со средним слоем что реализует, например, стеки связи, файловые системы или графические интерфейсы пользователя.

- DSP - коллекция библиотек цифровой обработки сигналов с более чем 60 функциями для различных типов данных: целочисленных (дробные форматы q7, q15 и q31) и для чисел с плавающей точкой одинарной точности (32 бита). Реализации оптимизированы для SIMD наборов инструкций для Cortex-M4/M7/M33/M35P

- NN - коллекция ядер эффективной нейронной сети разработанные для максимизации производительности и минимизации потребления памяти в процессорах Cortex-M

- RTOSv1 - основной API для операционных систем реального времени вместе с примером реализации основанной на RTX. Это включает компоненты программ, которые могут работать в нескольких системах RTOS.

- RTOSv2 - расширяет CMSIS-RTOSv1 поддержкой Armv8-M, динамическим созданием объектов, поддержку мультиядерных систем, бинарного совместимого интерфейса.

- SVD - периферийное объявление устройства, которое может быть использовано для создания периферийной осведомленности в отладчиках заголовочных файлах CMSIS-Core.

- DAP - прошивка для блока отладчика которая обеспечивает интерфейс к порту системы отладки CoreSight DebugAccess
- Zone - определяет методы для описания ресурсов системы и для разбиения на разделы этих ресурсов в нескольких проектах и областей выполнения.

3.2.1. Скачивание CMSIS5

Как ни странно, доступны на гитхабе, как обычно, не будем рисковать собой, а получим стабильную версию.

```
cd project_dir
git clone https://github.com/ARM-software/CMSIS_5
```

3.3. Компоновщик.

Компоновщик это что входит в состав тулчейна gcc, и занимается эта утилита сборкой исполняемого модуля из объектных модулей, полученных в результате компиляции.
Устройство довольно простое, почему бы по новой не написать целиком все. Итак, не совсем целиком. Если игнорирование официальной библиотеки от производителя это уже немного глупо, то игнорировать библиотеки создателей процессора просто нельзя. Итак, выбран процессор с ядром ARM Cortex M3, значит и будем опираться на файл `CMSIS/Device/ARM/ARMCM3/Source/GCC/gcc_arm.ld`.
Итак, скрипт компоновщика состоит из:

1) разметка памяти
2) определение входной точки
3) определения секций

3.3.1. разметка памяти

Компоновщик по умолчанию разрешает разметить всю доступную память, включая FLASH и SRAM. Для того, чтобы определить область памяти, которая будет использована компоновщиком, и избежать использования каких либо других областей памяти, существует команда MEMORY. Она обозначает расположение и размер блоков памяти. В

шаблоне она уже написана, от нас требуется занести адреса в переменные. Итак, микроконтроллер STM32F103C8T6 имеет 64кБ FLASH, которые начинаются с адреса 0x08000000. Значит __ROM_BASE = 0x08000000, __ROM_SIZE = 0x00010000. Оперативная память начинается с адреса __RAM_BASE = 0x20000000, размер 20кБ __RAM_SIZE = 0x00005000. Размеры стека и кучи оставим без изменений, доверимся примеру.

3.3.2. точка входа

Команда ENTRY используется для определения первой исполняемой инструкции. Синткасис команды: ENTRY(symbol), где symbol это переменная, которая обычно позже переопределяется в коде. В нашем файле программа стартует с адреса сброса - Reset_Handler.

3.3.3. определения секций

В скриптах компоновщика переменная точка (.) хранит текущий адрес в памяти, в то время как память разделена на секции. Команда SECTIONS контролирует то как размечаются input секции в output, а также порядок output секций в памяти. Итак, команда SECTION как правило используется для определения секции (section definition), которая определяет свойства output секций, такие как: ее расположения в памяти, выравнивание, содержание, шаблон заполнения, и целевую область памяти. Синтаксис команды следующий:

```
SECTIONS {
...
secname start BLOCK(align) (NOLOAD) : AT (ldadr)
    { contents } >region :phdr =fill
...
}
```

secname имя секции,
start определяет начальный адрес, с которого будет загружаться секция
BLOCK(align) выравнивание
(NOLOAD) обозначает невозможность загрузки секции во время выполнения программы
AT (ldadr) определяет адрес загрузки секции как ldadr
>region объявляет секцию как определенную область памяти
Служебные слова secname и contents требуются для определения секции, остальные опциональны.
Давайте рассмотрим секцию .text для примера.

```
.text :
{
  KEEP(*(.vectors))
  *(.text*)

  KEEP(*(.init))
  KEEP(*(.init))

  /* .ctors */
  *crtbegin.o(.ctors)
  *crtbegin?.o(.ctors)
  *(EXCLUDE_FILE(*crtend?.o *crtend.o) .ctors)
  *(SORT(.ctors.*))
  *(.ctors)

  /* .dtors */
  *crtbegin.o(.dtors)
  *crtbegin?.o(.dtors)
  *(EXCLUDE_FILE(*crtend?.o *crtend.o) .dtors)
  *(SORT(.dtors.*))
  *(.dtors)

  *(.rodata*)

  KEEP(*(.eh_frame*))
} > FLASH
```

.text это имя секции, в секции расположен код. KEEP(*(.vectors)) используется для того, чтобы секция векторов прерываний не была оптимизирована (удалена), похожий метод применен для секций init, fini, eh_frame. Напомним, что переменная '.' хранит текущий адрес. Таким образом скрипт извлекает из переменной vectors конечный адрес векторов, используя переменную точки, и располагает секцию vectors в текущей области памяти. Так же располагаются секции dtors и ctors по адресам, полученным из crtbegin.o и crtbegin?.o.
Секция ctors это список конструкторов (инициализационных функций), который подключает функции, инициализирующие данные в момент запуска программы (до запуска функции main). dtors устанавливает список деструкторов, которые могут быть вызваны при завершении программы.
> FLASH обозначает то, что секция text расположена в области FLASH памяти.

Пролистав и эту унылую теоретическую справку, вы можете догадаться, что и здесь со стандартным скриптом компоновщика все в порядке, ничего трогать не нужно.

3.4. startup файл

Если коротко, то это такой исходник, который не виден обычным пользователям, но именно он вызывает функцию main. Вообще возможен и Си вариант, но как правило пишется на ассемблере. Также есть стандартный в библотеке по пути `CMSIS_5/Device/ARM/ARMCM3/Source/GCC/startup_ARMCM3.S`.
Все что нам нужно в нем поменять, так это добавить местных векторов прерываний. То есть нужно, что бы компилятор знал, с какого адреса запускать то или иное прерывание, если оно произойдет. Позже пользователями в коде данные прерывания могут быть переопределены. Важно заметить, что по умолчанию для каждого из прерываний в качестве обработчика указан бесконечный цикл. Так сделано для того, чтобы контроллер там можно было поймать во время отладки. Но важно помнить помнить об этом факте во время написания прошивки, т.к. при вызванном и не определенном прерывании программа останется в бесконечном цикле. Итак, ориентируясь на Reference guide переписываем таблицу прерываний в наш пример. И это все что нужно добавить в этот файл для поддержки МК. Т.о. вместо:

```
.long    Interrupt0_Handler                 /*   0 Interrupt 0 */
.long    Interrupt1_Handler                 /*   1 Interrupt 1 */
.long    Interrupt2_Handler                 /*   2 Interrupt 2 */
.long    Interrupt3_Handler                 /*   3 Interrupt 3 */
```

поучится:

```
.long   WWDG_Handler
.long   PVD_Handler
.long   TAMPER_Handler
.long   RTC_Handler
```

Также и в списке определений обработчиков:

```
Set_Default_Handler   WWDG_Handler
Set_Default_Handler   PVD_Handler
Set_Default_Handler   TAMPER_Handler
Set_Default_Handler   RTC_Handler
```

3.5. Makefile

Стандартный мейкфайл будет выглядеть так.

3.6. Структура проекта.

Несмотря на кажущуюся простоту, проект все же состоит из 2к строчек кода. Потому он очень аккуратно структурирован.

3.7. Опрос кнопок.

По стандарту геймпады и клавиатуры должны отправлять информацию о состоянии кнопок в хост. Либо с запрошенной периодичностью в диапазоне от 4 до 1020 миллисекунд. Либо, если запрошен период равный нулю, информация отправляется только тогда, когда состояние кнопок поменялось.
Итак, для того, что бы получить полную поддержку стандарта, кнопки нужно опрашивать постоянно, при этом нужно не забыть про дребезг контактов. В моей реализации для опроса состояния кнопок выделен отдельный таймер счетчик МК. Счетчик запускает прерывание с периодом в одну миллисекунду. В этом прерывании происходит опрос кнопок, если кнопка зафиксирована нажатой больше семи раз подряд, программа принимает решение зафиксировать это нажатие. Нажатия длительностью менее 7 миллисекунд считаются дребезгом контактов и игнорируются.
Это же прерывание задает периодичность отправки данных в хост. То есть если cnt больше запрошенного периода (в стандарте обозначен как report duration) данные о нажатых кнопках (в стандарте report) отправляются в хост. Где cnt счетчик прерываний, срабатывающих каждую миллисекунду. Исключение с с отправкой по изменению так же учтено. Вот листинг прерывания:

```
static int cnt=0;
portPoll();
reportUpdate();
++cnt;
sendReport(gamepadPar.report, &cnt);
```

Сперва опрос портов (portPoll()), потом проверка времени нажатия и запись состояния кнопок в переменную report (reportUpdate()), функция отправки report. В функцию отправки так же передается внутренний счетчик количества вызовов прерываний cnt.

4. USB (универсальная последовательная шина)

Интересно заметить, что данный стандарт разработан не IEEE и не ISO, и даже не какой нибудь отдельно взятой компанией. Его разработал форум компаний, таким образом крупные фирмы решили не городить каждый по своему, как это обычно бывает. А просто вместе создали стандарт, который бы всех устраивал. К тому же USB Forum организация еще и некоммерческая

Еще один интересный факт, многие из вас помнят шутку про то сколько раз нужно перевернуть USB разъем, чтобы его подключить. Тем не менее, стандарт учел и эту особенность. Он советует располагать значок USB сверху разъема, тогда пользователь может на ощупь определить где верх, и подключить разъем не глядя с первого раза.

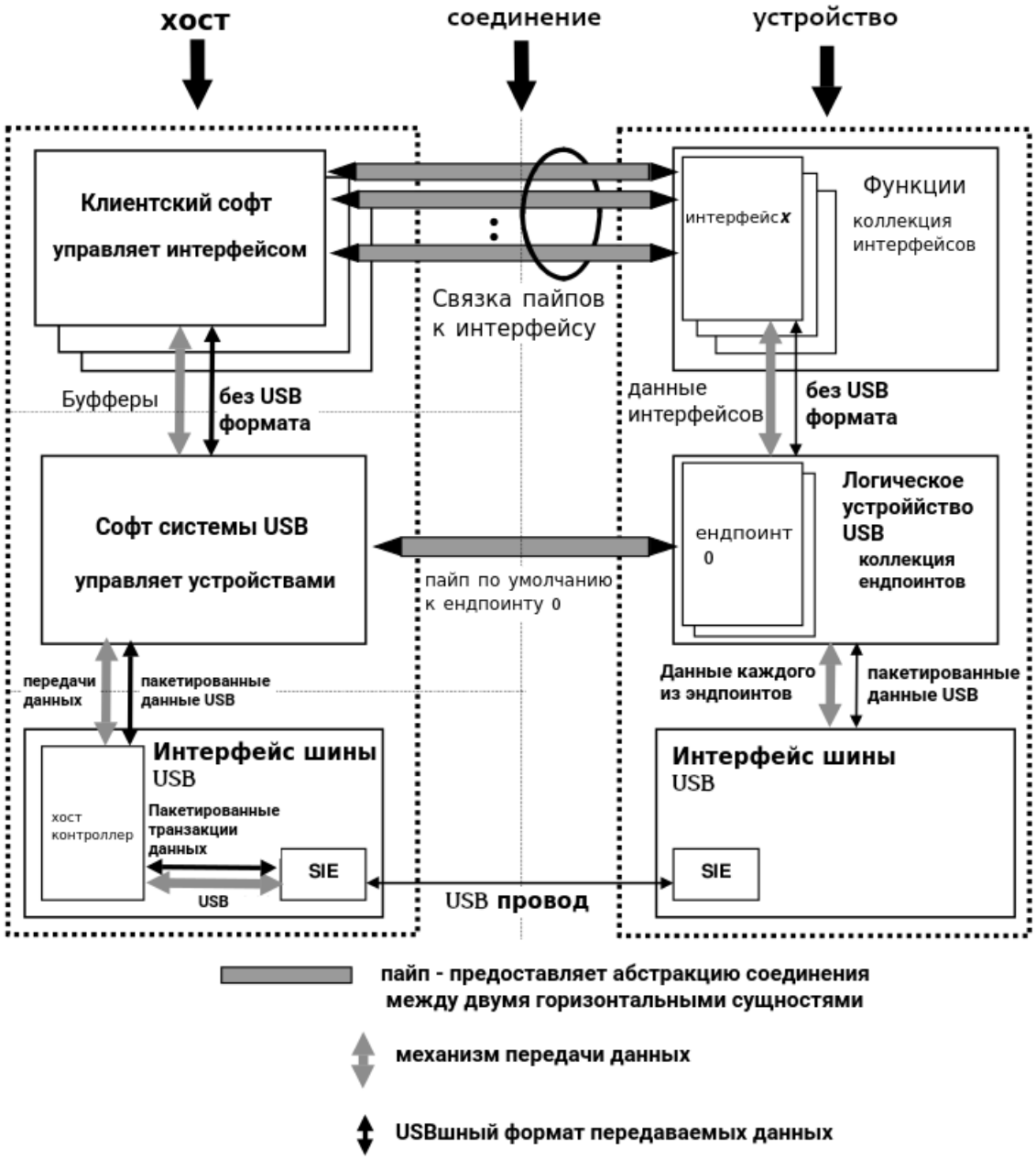
И как же рассказывать про интерфейс, не упоминая о его скорости. В самом лучшем случае, используя USB 2.0 full speed от технологии можно добиться 12 Мбит/с. Чего хватит для большинства приложений, например поток несжатого аудио занимает всего лишь 1 мегабит. Шина дифференциальная, потому вполне работает по длинным проводам.

Данный раздел по сути является переводом официального стандарта, по части того, что понадобилось в проекте. На всякий случай, приведу таблицу со стандартными типами USB устройств:

00h	N/A	Не задано
01h	Audio	Звуковая карта, MIDI
02h	Communication Device (CDC)	Модем, сетевая карта, COM-порт
03h	Human Interface Device (HID)	Клавиатура, мышь, джойстик
06h	Image	Веб-камера, сканер
07h	Printer	Принтер
08h	Mass Storage Device (MSD)	USB-накопитель, карта памяти, кардридер
05h	Physical Interface Device (PID)	Джойстик с поддержкой Force feedback
09h	USB hub	USB-хаб
0Ah	CDC Data	Сетевая карта, USB-COM
0Bh	Smart Card Reader (CCID)	Считыватель смарт-карт
0Dh	Content security	Биометрический сканер
0Eh	Video Device Class	Веб-камера
0Fh	Personal Healthcare	Индикатор пульса, медицинское оборудование
DCh	Diagnostic Device	USB устройства в отладке
E0h	Wireless Controller	Bluetooth-адаптер
EFh	Miscellaneous	все что не влезло в пр. классы
FEh	Application-specific	режим обновления прошивки (DFU)

4.1. Краткий обзор технологии USB

Несмотря на внешнюю простоту (штыревой разъем на четыре провода, куда уж проще), у этой технологии есть протокол и четыре уровня абстракции. В документациях часто бывают картинки, которые просто нужно вдумчиво разглядывать, как будто это ковер. Неторопливо рассматривать каждую завитушку причудливого узора. Такая картинка, вернее схема, представлена ниже и описывает модель передачи данных USB.



Получив фотокопию внутри себя, пора прочесть пояснения. Все что находится на нижних уровнях реализовано аппаратно, то есть давным давно отлажено компанией Intel в их IP ядре UTMI, которое и применено в выбранном нами чипе. И мы, реализовывая устройство, даже и не увидим, что там внутри. Тем не менее, то что обозначено на схеме, как SIE (движок последовательного интерфейса) скрывает под собой:

- электрическую схему интерфейса (драйвер шины)

- кодировщик NRZI
- обработчик пакетов

В то время, как физически соединение на нижнем уровне представлено четырьмя проводами. Положительным и отрицательным проводами дифференциальной шины данных (DM и DP), и проводами питания (+5В, GND).
Обмен данными происходит не непрерывно, стандартом предусмотрен протокол, в котором осуществляется пакетный обмен данными. Протокол и пакеты это тема отдельного раздела стандарта, здесь её рассматривать так же не нужно. Все что нам нужно знать, это то, что во всем потоке данных, среди прочих, существуют пакеты с данными. И прошивка устройства на нижнем уровне работает с их приемом и передачей по событию CTR.
Поднимаемся на уровень выше. В середине картинки можно увидеть так называемые ендпоинты, дословный перевод конечные точки. Эндпоинт это такая фиксированная порция USB устройства, у каждого эндпоинта есть свой уникальный буфер и адрес, друг с другом они никак не связаны. А вот поток данных между ендпоинтом на компьютере и ендпоинтом в устройстве называется пайплайном. Получается, что средний уровень определен обязательным, нулевым эндпоинтом, он же еще называется контрольным ендпоинтом. По которому осуществляется настройка, определение и начальная конфигурация устройства.
На верхнем уровне находится интерфейсы устройства, то есть уже само приложение или приложения (технология USB поддерживает и комбинированные устройства). Работают они через все остальные ендпоинты. В моем случае интерфейс один, и это интерфейс USB HID устройства с двумя кнопками и двумя осями. Работает он поверх одного единственного ендпоинта типа Interrupt под номером 1. Ендпоинт отправляет пакет с информацией о состоянии кнопок с периодичностью в 32 миллисекунды.

4.2. Периферия USB.

4.2.1. Включение

Как известно, выбранный МК имеет аппаратный USB. Данная реализация аппаратно обеспечивает работу двух нижних уровней. Теперь давайте посмотрим, что нужно сделать, что бы его хотя бы включить. То есть что бы по шине пошли какие нибудь данные.

4.2.2. Буффер USB STM32

Для USB в даннном микроконтроллере выделено 512 байт SRAM. Этот раздел памяти доступен непосредственно для USB периферии, и для программы пользователя. То есть, для отправки пакета программе нужно сложить данные именно в этот раздел памяти, а позже периферия их отправит. Так же и с приемом, успешно приняв пакет с данными периферия оставит их в этой области, где программе их нужно забрать. В документации много говориться о том, что доступ к памяти осуществляется на разных частотах (частота интерфейса и процессора разные). Но они решили этот вопрос, и там все синхронизировано, можно считывать и записывать в любой момент.
Теперь о том, как до нее добраться. Память разбита на слова (16бит), при этом указатели у нее 32 битные, потому считывание и запись происходит очень весело. Пример записи пакета для отправки:

```
uint16_t *bufferPtr = (uint16_t*)(USB_ADDR0_TX*2 + USB_CAN_SRAM_BASE_M
for(int i=0 ; i<(size/2) ; ++i) {
    *bufferPtr = *input;
    input++;
    bufferPtr += 2;
}
```

То есть здесь видно, что данные в буфер складываются по 16 бит, при этом на каждые 16 бит данных, предназначенных для отправки, указатель буффера инкрементируется дважды. То есть, в каждых 32х битах хранится по 16 бит, причем эти 16 бит младшие.
Но и это еще не все, таблица разметки этой памяти хранится в этом же злобном разделе памяти. Но и тут вопрос решается вот такими веселыми макросами, заготовленными заранее для каждого эндпоинта:

```
// addresses of endpoint 1
#define USB_ADDR1_TX     MMIO32(USB_CAN_SRAM_BASE_MY + ((uint16_t)USB_B
#define USB_COUNT1_TX    MMIO32(USB_CAN_SRAM_BASE_MY + ((uint16_t)USB_B
#define USB_ADDR1_RX     MMIO32(USB_CAN_SRAM_BASE_MY + ((uint16_t)USB_B
#define USB_COUNT1_RX    MMIO32(USB_CAN_SRAM_BASE_MY + ((uint16_t)USB_B
// addresses of endpoint 2
#define USB_ADDR2_TX     MMIO32(USB_CAN_SRAM_BASE_MY + ((uint16_t)USB_B
...
```

Где USB BTABLE это регистр, который хранит смещение, на случай если вы захотите разместить таблицу в конце раздела, а не просто в его начале. И вот это уже совсем непонятно зачем.

А теперь уже можно рассказать и о самой разметке. А вот и пример таблицы разметки из официальной документации:

4.2.3. Ендпоинты

4.3. Дескрипоры USB.

Во первых их несколько. Во вторых дескрипторы это база данных устройства. Как известно, USB поддерживает горячее подключение (hot plug). Так вот, после подключения устройства к компьютеру, до этого момента не знавшего ничего о нем. Хост компьютер может определить его стандартным драйвером за считанные миллисекунды (если это linux), или секунды (если это windows), и оно будет готово к использованию.
Такие хорошие возможности обеспечивают во многом дескрипторы. Т.к. именно в них хранится вся информация об устройстве: его тип, режимы работы, скорость, энергопотребление, и то как с ним общаться. И именно дескрипторы и запрашиваются во время начальной конфигурации устройства. Так же из-за того, что дескрипторы жестко типизированы, занимают они считанные байты. Вот существующие типы дескрипторов:

- Device - содержит основную информацию об устройстве
- Configuration - здесь особенности данного устройства
- Interface - у каждого устройства может быть несколько интерфейсов, в простейшем случае он один
- Endpoint - тут конфигурируется ендпоинт, определяются размеры пакетов и частота их передачи, подробнее про ендпоинты написано во введении 4.1
- String - название устройства, которое можно увидеть по команде lsusb, хранится в дескрипторах данного типа.
- Report - формат репорта

4.3.1. Device дескриптор.

4.3.2. Configuration дескриптор.

4.3.3. Interface дескриптор.

4.3.4. Endpoint дескриптор.

4.3.5. String дескриптор.

4.3.6. Report дескриптор.

4.4. Стандартный протокол USB.

В USB имеется так называемый нулевой конфигурационный ендпоит, он же ендпоинт по умолчанию. Он существует для того, чтобы обрабатывать setup транзакции (транзакции настройки). В транзакциях данного типа упакованы запросы. Среди них есть обязательные, те что определены в основном стандарте. И дополнительные, те что определены в стандарте на какой то из конкретных типов устройств, например HID реквесты 4.5.1.
Нумерация устройства, запрос дескрипторов, отключение ненужных ендпоинтов и включение их обратно. Да и вообще вся служебная работа осуществляется через стандартные запросы. Вот их список:

- getStatus
- clearFeature
- setFeature
- setAddress
- getDescriptor
- getConfiguration
- setConfiguration
- getInterface
- setInterface

4.4.1. getStatus дескриптор.

4.4.2. clearFeature дескриптор.

4.4.3. setFeature дескриптор.

4.4.4. setAddress дескриптор.

4.4.5. getDescriptor дескриптор.

4.4.6. set Interface

4.5. USB HID

HID устройства человеко машинного интерфейса. В это подмножество USB устройств укладывается практически все, что взаимодействует с человеком. Это и клавиатуры и мышки, и джойстики и рули, и даже костюм виртуальной реальности. Также подсветка, управляемая по USB это тоже HID устройство.
Как и говорилось ранее, единственный поток данных который идет из/в USB HID это так называемые репорты. Примеры репортов следующие: например, клавиатура докладывает о нажатых клавишах, либо руль говорит о том, что его повернули на несколько градусов, либо компьютер просит поменять цвет подсветки. Репорты чувствительны к времени отправки, но поток данных совсем небольшой. Поэтому они передаются через interrupt endpoint.
Поток маленький попросту потому, что типы данных и величины определены заранее в дескрипторе. А в репорте отправляются сами данные в битовых полях. Таким образом, определенная заранее кнопка занимает всего лишь один бит в репорте.
Далее рассмотрим формат репорта на примере моего устройства. Его дескриптор обсуждался в разделе 4.3.

4.5.1. HID реквесты

Для данного типа устройств определены реквесты, дополнительные к основным. Их на самом деле много, обязательных всего три, и те, как показала практика драйвер так ни разу и не запросил. Вот они:

- get report - дополнительный способ запросить репорт.
- get idle - считать текущий период отправки репорта.
- set idle - установить новый период отправки репорта.

Если основной поток для репорта это interrupt endpoint, то по контрольному пайпу тоже можно запрашивать репорт, но такой способ считается дополнительным. Также хост может менять периодичность отправки репортов. Форматы передаваемых запросов и возвращаемых ими данных можете посмотреть в коде, файлы:
`https://github.com/dltech/usb_device3/blob/main/lib/usb_hid.c`
`https://github.com/dltech/usb_device3/blob/main/lib/usb_hid.h`

5. Отладка.

Я люблю писать рабочий код с первого раза, потому процесс отладки программы я производил очень примитивным способом. Отладчиком gdb через терминал, складывая в глобальные переменные все что вызывало сомнения.