

**ДИПЛОМНА РАБОТА**

**по професия код 481020 „Системен програмист“**

**специалност код 4810201**  **„Системно програмиране“**

Тема: Генератор на случайни числа

Дипломант: Дипломен ръководител:

*Стефан Иванов Георгиев инж. Николай Ненов*

СОФИЯ

2025

Съдържание

[1 Резюме 7](#_Toc188260845)

[2 Увод 7](#_Toc188260846)

[2.1 Необходимост от случайни числа 7](#_Toc188260847)

[2.1.1 Криптография 7](#_Toc188260848)

[2.1.2 Изкуствен интелект 8](#_Toc188260849)

[2.1.3 Тестове и симулации на системи 8](#_Toc188260850)

[2.2 Случайни Числа – видове, ентропия 8](#_Toc188260851)

[2.2.1 Видове 8](#_Toc188260852)

[2.2.2 Ентропия 9](#_Toc188260853)

[2.3 Методи за атака на криптографски системи базирани на ниска ентропия 10](#_Toc188260854)

[2.4 Примери 10](#_Toc188260855)

[2.4.1 Пример: Атака чрез изчерпване на семена в автомобилни системи (Seed-Key Exchange) 10](#_Toc188260856)

[2.4.2 Пример: Ниска ентропия при ферми от сървъри 11](#_Toc188260857)

[2.5 Ентропия в съвременните ОС 11](#_Toc188260858)

[2.5.1 Централизирано генериране и защита на случайни числа 11](#_Toc188260859)

[2.5.2 Източници на ентропия 12](#_Toc188260860)

[2.5.3 Добавяне на ентропия от User Mode и Kernel Mode 12](#_Toc188260861)

[2.5.4 Генератори на ентропия 12](#_Toc188260862)

[2.5.5 Предизвикателства при събирането на ентропия 13](#_Toc188260863)

[2.6 Съществуващи решения 13](#_Toc188260864)

[2.6.1 OneRNG 13](#_Toc188260865)

[2.6.2 ubld.it TrueRNG V3 USB Hardware Random Number Generator 14](#_Toc188260866)

[2.6.3 Quantis QRNG USB 15](#_Toc188260867)

[2.6.4 Заключение на проучването за съществуващи генератори на случайни числа: 15](#_Toc188260868)

[3 Генерация на случайни числа 16](#_Toc188260869)

[3.1 Принципи на генериране на електронен шум 16](#_Toc188260870)

[3.1.1 Лавинен пробив 16](#_Toc188260871)

[3.1.2 Термичен шум 16](#_Toc188260872)

[3.1.3 Шум от токов удар (Shot Noise) 17](#_Toc188260873)

[3.1.4 Ценеров ефект 17](#_Toc188260874)

[3.2 Изселдвана схема за лавинен шум !!!! 17](#_Toc188260875)

[3.2.1 Основни компоненти и тяхната роля: 17](#_Toc188260876)

[3.2.2 Резултати 18](#_Toc188260877)

[3.3 Хардуер 19](#_Toc188260878)

[3.3.1 Захранване 19](#_Toc188260879)

[3.3.2 Помпа на заряд 19](#_Toc188260880)

[3.3.3 Шумогенератор 21](#_Toc188260881)

[3.3.4 Усилване не сигнал 22](#_Toc188260882)

[3.3.5 АЦП 22](#_Toc188260883)

[3.3.6 Използвани компоненти и консумацията им 23](#_Toc188260884)

[3.3.7 Схема 25](#_Toc188260885)

[3.3.8 Изработка на платка 26](#_Toc188260886)

[3.3.9 Блокова схема 28](#_Toc188260887)

[3.4 Софтуер 28](#_Toc188260888)

[3.4.1 Управление на помпа на заряд 28](#_Toc188260889)

[3.4.2 АЦП на второ ядро 30](#_Toc188260890)

[4 USB комуникация за Entropy Device 31](#_Toc188260891)

[4.1 USB 31](#_Toc188260892)

[4.1.1 USB архитектура 31](#_Toc188260893)

[4.1.2 USB скорости 33](#_Toc188260894)

[4.1.3 USB крайни точки (Endpoints) 34](#_Toc188260895)

[4.1.4 Комуникационен протокол 36](#_Toc188260896)

[4.1.5 USB дескриптори 38](#_Toc188260897)

[4.1.6 USB устройства от клас 41](#_Toc188260898)

[4.1.7 USB изброяване и Конфигурация при зареждане 42](#_Toc188260899)

[4.2 Raspberry Pico 42](#_Toc188260900)

[4.2.1 USB възможностите на Pico 42](#_Toc188260901)

[4.2.2 интерфейс с USB контролер 43](#_Toc188260902)

[4.2.3 Vendor-specific идентификация 45](#_Toc188260903)

[4.2.4 Цикличен буфер на случайни числа 46](#_Toc188260904)

[4.2.5 Индикации 49](#_Toc188260905)

[4.3 Linux kernel driver 49](#_Toc188260906)

[4.3.1 Живот на кърнълски модул 49](#_Toc188260907)

[4.3.2 Rule файл за зареждане на драйвер на USB 50](#_Toc188260908)

[4.3.3 Драйвери, отнасящи се до USB 50](#_Toc188260909)

[4.3.4 Регистриране на USB драйвер 51](#_Toc188260910)

[4.3.5 Живот на USB device 52](#_Toc188260911)

[4.3.6 Регистриране и премахване на USB устройства: 52](#_Toc188260912)

[4.3.7 Прехвърляне на данни от USB endpoint в bulk режим 53](#_Toc188260913)

[4.3.8 Захранване на ОС с ентропия 53](#_Toc188260914)

[4.3.9 Файлов интерфейс за четене на суровите данни 54](#_Toc188260915)

[5 тестове на случайни числа 55](#_Toc188260916)

[5.1 Визуален анализ на тестове 55](#_Toc188260917)

[5.1.1 BITMAP 55](#_Toc188260918)

[5.1.2 Битова хистограма 57](#_Toc188260919)

[5.1.3 Хистограма на байтове 57](#_Toc188260920)

[5.2 NIST 58](#_Toc188260921)

[5.2.1 Frequency (Честота) 58](#_Toc188260922)

[5.2.2 Block Frequency (Честота в блокове) 58](#_Toc188260923)

[5.2.3 Cumulative Sums (Натрупани суми) 58](#_Toc188260924)

[5.2.4 Runs (Поредици) 58](#_Toc188260925)

[5.2.5 Longest Run (Най-дълга поредица) 59](#_Toc188260926)

[5.2.6 Rank (Ранг) 59](#_Toc188260927)

[5.2.7 FFT (Бързо преобразуване на Фурие) 59](#_Toc188260928)

[5.2.8 Non-Overlapping Template (Не-препокриващи се шаблони) 59](#_Toc188260929)

[5.2.9 Overlapping Template (Препокриващи се шаблони) 59](#_Toc188260930)

[5.2.10 Universal (Универсален тест) 59](#_Toc188260931)

[5.2.11 Approximate Entropy (Приблизителна ентропия) 59](#_Toc188260932)

[5.2.12 Random Excursions (Случайни отклонения) 59](#_Toc188260933)

[5.2.13 Random Excursions Variant (Вариант на случайни отклонения) 60](#_Toc188260934)

[5.2.14 Linear Complexity (Линейна сложност) 60](#_Toc188260935)

[6 Потребителски интерфейс 60](#_Toc188260936)

[6.1 Комуникация на приложение с Entropy устройството 60](#_Toc188260937)

[6.2 Визуализация на шум 61](#_Toc188260938)

[6.3 Използвани библиотеки 61](#_Toc188260939)

[7 Бъдещи подобрения 61](#_Toc188260940)

[7.1 ВЧ усилване на шума за подобряване на разпределението на числата 61](#_Toc188260941)

[7.2 Режим с ниска консумация 61](#_Toc188260942)

[7.3 Криптирана комуникация по USB с Diffie-Hellman 62](#_Toc188260943)

[8 Заключение 62](#_Toc188260944)

[9 Цитирани източници 62](#_Toc188260945)

[10 Приложениe 65](#_Toc188260946)

[Приложение 1 65](#_Toc188260947)

[Приложение 2 65](#_Toc188260948)

[Приложение 3 65](#_Toc188260949)

[Приложение 4 68](#_Toc188260950)

[Приложение 5 68](#_Toc188260951)

[Приложение 6 68](#_Toc188260952)

[Приложение 7: 68](#_Toc188260953)

Фигури

[Фигура 2‑1 Съобщение за грешка в линукс 7](#_Toc188282669)

[Фигура 2‑2 OneRNG 14](#_Toc188282670)

[Фигура 2‑3 TrueRNG V3 14](#_Toc188282671)

[Фигура 2‑4 Quantis QRNG 15](#_Toc188282672)

[Фигура 3‑1 Лавинен генератор на бял шум 18](#_Toc188282673)

[Фигура 3‑2 Генериран шум 18](#_Toc188282674)

[Фигура 3‑3 Изглаждащи кондензатори 19](#_Toc188282675)

[Фигура 3‑4 Едностъпална помпа на заряд 20](#_Toc188282676)

[Фигура 3‑5 Помпа на заряд с шотки диоди 21](#_Toc188282677)

[Фигура 3‑6 Шумогенератор 21](#_Toc188282678)

[Фигура 3‑7 Усилвател и делител на напрежение 22](#_Toc188282679)

[Фигура 3‑8 SAR ADC Блокова диаграма 23](#_Toc188282680)

[Фигура 3‑9 Финална схема 25](#_Toc188282681)

[Фигура 3‑10 Блокова схема на генератора на случайни числа 26](#_Toc188282682)

[Фигура 4‑1 Много периферни могат да се свържат в един хост 29](#_Toc188282683)

[Фигура 4‑2 Нива на свързаност 30](#_Toc188282684)

[Фигура 4‑3 Абтракция на интерфейса 31](#_Toc188282685)

[Фигура 5‑1 random.org TRNG 53](#_Toc188282686)

[Фигура 5‑2 PHP rand() PRNG 53](#_Toc188282687)

[Фигура 5‑3 Примерна битова хистограмаХистограма на байтове 54](#_Toc188282688)

[Фигура 5‑4 Хистограма на разпределение на стойности 55](#_Toc188282689)

Таблици

[Таблица 10‑1 Електронни компоненти 62](#_Toc188282690)

[Таблица 10‑2 Дескриптор на устройства 67](#_Toc188282691)

[Таблица 10‑3 Конфигурационен дескриптор 68](#_Toc188282692)

[Таблица 10‑4 Дескриптор на интерфейс 68](#_Toc188282693)

[Таблица 10‑5 Дескриптор на endpoint 69](#_Toc188282694)

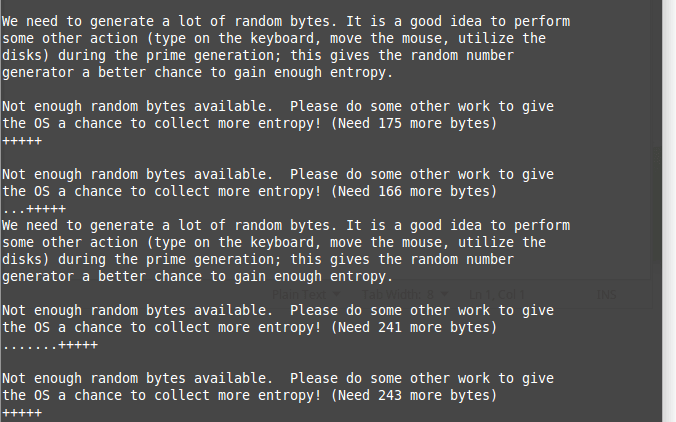
# Резюме

Случайните числа са ключови за много области като криптография, моделиране и др. Генерирането на редици от псевдослучайни числа е предвидимо при захранване на генератора с малко ентропия. Настоящата работа адресира този проблем чрез USB устройство, което захранва Линукс машина с ентропия, генерирана от лавинен шум в електронна схема.

# Увод

## Необходимост от случайни числа

Случайните числа играят ключова роля в редица области на съвременните технологии и науки, като криптография, моделиране и изкуствен интелект. Тяхното приложение е не само широко, но и критично за осигуряване на надеждността и сигурността на различни системи и алгоритми[1]. Във Фигура 2‑1 се вижда как генерацията на случайни числа изисква достатъчна ентропия.



Фигура ‑ Съобщение за грешка в линукс

### Криптография

Случайните числа са основен компонент в криптографските системи, където те се използват за генериране на ключове, вектори и други параметри, необходими за сигурността на данните. Качеството на криптографската защита зависи пряко от способността за генериране на наистина случайни или криптографски устойчиви псевдослучайни числа, които да не могат да бъдат предсказани. Дори минимално отклонение от случайността може да компрометира цялата система и да направи чувствителна информация уязвима.

### Изкуствен интелект

В изкуствения интелект (ИИ) случайните числа се използват за разнообразие от задачи – от инициализацията на теглата в невронните мрежи до създаването на случайни тренировъчни набори от данни и реализирането на алгоритми за обучение, като например стохастичен градиентен спуск. Тези приложения са особено важни за предотвратяване на проблеми като пренасищане с данни (overfitting) или достигане на локални оптимуми. Освен това, методи като Монте Карло симулациите, широко използвани в машинното обучение и оптимизацията, разчитат изцяло на случайни числа.

### Тестове и симулации на системи

Случайните числа се използват за тестване на софтуерни системи и хардуер чрез генериране на различни входни данни. Това помага за идентифициране на грешки и слабости в системата. Например, в тестването на мрежи или алгоритми за сигурност случайните данни могат да разкрият уязвимости или несъответствия в поведението.

## Случайни Числа – видове, ентропия

### Видове

#### Псевдослучайни числа

Псевдослучайните числа са редица, генерирана от псевдослучаен генератор (PRNG). Тази редица числа има свойства, които приблизително наподобяват свойствата на поредици от случайни числа. Поредицата, генерирана с PRNG, не е наистина случайна, защото е напълно определена от начална стойност, наречена семе (seed) на PRNG (което може да включва наистина случайни стойности). Въпреки че поредици, които са по-близки до наистина случайните, могат да бъдат генерирани с помощта на хардуерни генератори на случайни числа, псевдослучайните генератори са важни на практика заради тяхната бързина при генерирането на числа [2].

#### Истински случайни числа

Генераторът на истински случайни числа (TRNG) е устройство или система, която генерира поредица от числа, основаваща се на физически феномени, които са наистина случайни. За разлика от псевдослучайните генератори, TRNG не използва математически алгоритми, а черпи стойности от процеси като електронен шум, радиоактивен разпад или други хаотични явления. Тези физически източници гарантират, че генерираната поредица е непредсказуема и не може да бъде възпроизведена. TRNG е изключително важен за приложения, при които абсолютната случайност и сигурност са критични, например в криптографията. Въпреки че TRNG обикновено е по-бавен в сравнение с PRNG, той осигурява ненадмината надеждност за генериране на наистина случайни числа [3].

### Ентропия

Ентропията е мярка за безпорядък в дадена система или набор от данни. Колкото по-висока е ентропията, толкова по-малко предсказуем е изходът и толкова по-сложна е структурата на данните. Ентропията в контекста на случайните числа се отнася до степента на непредсказуемост или случайност на генерираните стойности. Ако имаме набор от случайни числа, ентропията измерва колко "непредсказуеми" или "несигурни" са тези числа. Колкото по-равномерно разпределени са числата в набора и колкото по-малко се повтарят модели или закономерности, толкова по-висока е ентропията и, съответно, случаят е по-близо до идеален случай [4].

В идеалния случай за генератор на случайни числа ентропията трябва да бъде максимална, което означава, че всяка стойност има еднаква вероятност да бъде генерирана и няма лесно откриваеми модели или редове. Ако генераторът има висока ентропия, това обикновено означава, че той генерира "истински" случайни числа. Пример за такъв генератор е истински случайно генерираните числа от физически процеси (като шум в електронни вериги), които могат да имат максимална ентропия.

От друга страна, ако генераторът има ниска ентропия, това може да означава, че съществуват повтарящи се модели или предсказуеми закономерности в генерираните числа, което е често срещано при псевдослучайни генератори. Псевдослучайните числа се генерират чрез алгоритми, които използват начален "seed" и дори ако тези числа изглеждат случайни на повърхността, техният ред може да бъде предсказан или възпроизведен, ако се знае началното число. Това води до липса ентропия.

## Методи за атака на криптографски системи базирани на ниска ентропия

Криптографските системи, разполагащи с ниска ентропия, често са уязвими от различни методи за атака, които се възползват от ограниченото количество случайност в техния дизайн или работа. Ниската ентропия означава, че броят на възможните ключове е недостатъчен за осигуряване на надеждна защита.

Съществуват няколко методa за атака

Brute-force атака: При тази атака се изпробват всички възможни стойности на ключовете, докато не бъде намерен правилният. Ако обхватът на стойностите е малък, това може да бъде осъществено за кратко време използвайки модерни технологии.

lookup tables : Атаки lookup tables включва предварително изчисляване и съхранение на всички възможни комбинации от семена и ключове. При реално приложение атакуващият може просто да използва таблицата, за да намери ключа за дадено семе почти мигновено.

Линейна зависимост в алгоритмите: Ако съществуват предвидими зависимости между seed-ове и ключове, атакуващите могат да използват математически модели за намиране на ключовете [5].

## Примери

### Пример: Атака чрез изчерпване на семена в автомобилни системи (Seed-Key Exchange)

Seed-Key Exchange е процес в автомобилната индустрия, използван за удостоверяване на диагностично устройство, което иска да комуникира с електронен управляващ блок (ECU) за да модифицира функции, свързани с безопасността или емисиите на превозното средство. ECU изпраща случайно число, което диагностичното устройство използва, за да изчисли ключ. След това диагностичното устройство изпраща ключа обратно на ECU за проверка. ECU сравнява получения ключ със своето вътрешно изчисление. Ако съвпадат, диагностичната сесия продължава.

Уязвимости

Малкият диапазон от стойности за семена (например само 65,535 стойности) прави системата уязвима на brute-force атаки, които могат да се осъществят за няколко дни или по-бързо [6].

### Пример: Ниска ентропия при ферми от сървъри

При стартиране на голям брой сървъри (например във ферма от сървъри), те често използват едни и същи стойности на началните параметри (seed), ако липсва ентропия в генератора на случайни числа. Това може да доведе до дублиране на ключове: Сървърите генерират идентични криптографски ключове, което компрометира сигурността на системата.

Предвидимост: Атакуващ може да предвиди стойностите, използвани от системите, ако има информация за началното състояние на генератора [7].

## Ентропия в съвременните ОС

Ентропията е от съществено значение за генерирането на защитени случайни числа, които играят критична роля в съвременните компютърни системи. Генераторите на случайни числа (Random Number Generators, RNGs) разчитат на различни източници на ентропия, за да осигурят сигурността на криптографските операции като криптиране, удостоверяване, създаване на ключове и други.

### Централизирано генериране и защита на случайни числа

Съвременните операционни системи централизирано управляват процеса на събиране и използване на ентропия в рамките на ядрото (kernel) чрез специализирани механизми и подсистеми. Примери за такива механизми включват [4]:

/dev/random и /dev/urandom в Unix-базирани системи, които предоставят интерфейси за достъп до криптографски защитени случайни числа.

CryptGenRandom в Windows за генериране на криптографски сигурни числа.

Тези механизми използват набор от източници на ентропия, които осигуряват достатъчно ниво на случайност и защита срещу предсказуемост.

### Източници на ентропия

Операционните системи събират ентропия от различни псевдослучайни събития и източници. Примери за такива източници включват:

Движение на мишката: Проследяване на случайни и неравномерни движения на мишката от потребителя.

Клавишни натискания: Анализ на времевите интервали между натисканията на клавиши, които са трудни за предсказване.

Прекъсвания на външни устройства: Събития от хардуерни устройства, като мрежови карти, твърди дискове или USB устройства, които включват непредвидими временни вариации.

Времена на изпълнение на заявки: Латентност при изпълнение на външни мрежови заявки или операции с файловата система.

Сензори: В мобилните устройства се използват допълнителни източници като акселерометри и жироскопи, които осигуряват допълнителна ентропия [4].

### Добавяне на ентропия от User Mode и Kernel Mode

Операционните системи позволяват добавянето на ентропия както от потребителския режим (User Mode), така и от режима на ядрото (Kernel Mode).

User Mode: Приложенията могат да добавят ентропия към пуловете на случайни числа чрез специализирани API-та. Например, програмист може да подаде допълнителна случайна информация от приложение, базирана на специфични източници на ентропия чрез /dev/random [8].

Kernel Mode: Ядрото събира и обработва ентропия автоматично от хардуерни събития и системни процеси. Това е по-сигурен подход, тъй като ядрото има достъп до източници, които са недостъпни за потребителския режим, като прекъсвания, хардуерни регистри и времеви параметри.

### Генератори на ентропия

Съвременните операционни системи използват криптографски защитени генератори на случайни числа (CSPRNGs - Cryptographically Secure Pseudo-Random Number Generators), които осигуряват високо ниво на сигурност. Основните им характеристики включват:

Използване на пулове с ентропия: Генераторите натрупват случайни данни в пулове, които периодично се използват за създаване на нови случайни числа.

Ресийдинг: Периодичното презареждане на генератора с нова ентропия предотвратява предсказването на бъдещи стойности, дори ако част от вътрешното състояние е компрометирана.

Forward Secrecy (Напредъчна сигурност): След всяка употреба ключовете и данните се презаписват, за да се гарантира, че компрометиране на бъдеща стойност не застрашава минали стойности [9].

### Предизвикателства при събирането на ентропия

Въпреки напредъка в технологията, събирането на достатъчно ентропия може да бъде предизвикателство, особено в следните сценарии:

Стартиране на системата: При стартиране на операционната система (например в сървърни ферми) пуловете с ентропия са все още празни, което прави криптографските операции по-уязвими.

Виртуални машини: Те често споделят един и същ пул с ентропия с хоста, което ограничава наличната случайност.

Вградени устройства: Ограничените ресурси и липсата на хардуерни генератори на случайни числа могат да доведат до ниска ентропия [1].

## Съществуващи решения

### OneRNG

Устройството показано на Фигура 2‑2 е хардуерен генератор на случайни числа, свързващ се чрез USB, който осигурява висококачествени случайни данни за подобряване на сигурността на компютърните системи. Устройството използва отворен хардуер и софтуер, което позволява на потребителите да проверяват и удостоверяват неговата работа. OneRNG е защитено от софтуерни атаки и не позволява фърмуерна промяна през USB, като същевременно може да се интегрира с операционната система за подобряване на съществуващите механизми за генериране на случайни числа.(9 years ago last commit and support) [10]



Фигура ‑ OneRNG

### ubld.it TrueRNG V3 USB Hardware Random Number Generator

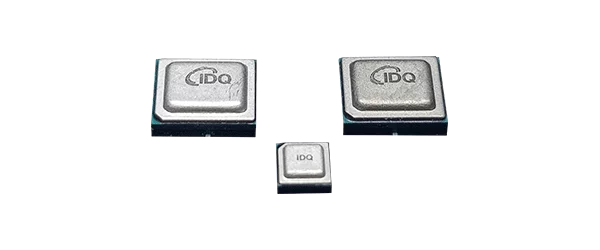
TrueRNG V3 от ubld.it е хардуерен генератор на случайни числа, проектиран за висока скорост и качество на случайността. Той използва лавинен ефект в полупроводников преход за генериране на случайни числа, осигурявайки постоянен поток от над 400 килобита в секунда през USB CDC сериен порт. Устройството включва усъвършенствани техники за „whitening“, които гарантират максимална ентропия и минимални изкривявания, което го прави подходящо за приложения в сигурността, игрите и статистическите задачи. Генераторът е показан в Фигура 2‑3 [11].



Фигура ‑ TrueRNG V3

### Quantis QRNG USB

Quantis QRNG (показано в Фигура 2‑4) от ID Quantique е хардуерен генератор на случайни числа, който използва квантовата физика за създаване на истинска случайност. Той използва елементарен процес на квантова оптика, което го прави неуязвим за външни влияния и позволява проверка на състоянието в реално време. Устройството предоставя ентропия със скорост до 4 Mbps и е съвместимо с повечето платформи, като се интегрира лесно в съществуващи приложения. Quantis QRNG е сертифициран от множество реномирани институции и е преминал строги статистически тестове, които гарантират високо качество на случайността [12].



Фигура ‑ Quantis QRNG

### Заключение на проучването за съществуващи генератори на случайни числа:

След проведено проучване беше установено, че на пазара съществуват три различни вида хардуерни генератори на случайни числа (HWRNG), които са създадени за различни приложения, свързани със сигурност и криптография. Обаче всеки от тях има съществени ограничения.

Първият от тях, OneRNG, е устройство с отворен код, което е популярно в средите на ентусиастите и експертите, използващи Linux. OneRNG позволява на потребителите да инспектират физически и да проверят надеждността на устройството, което е една от неговите силни страни. Въпреки това, устройството не е получавало актуализации от около 10 години и поддържа единствено Linux платформи. Това прави OneRNG неподходящ за потребители, които работят с Windows или macOS, и поставя под въпрос неговата съвместимост с по-нови системи и технологии.

Второто устройство, TrueRNG V3, предоставя висококачествени случайни числа и е предпочитано от мнозина заради своята способност да генерира поток от над 400 килобита, което е 50кB в секунда. Въпреки това, съществува съществена липса на документация и спецификации за продукта, описано е че генераторът преминава Dieharder, ENT, Rngtesт, обаче не са споменати NIST тестовете.

Накрая, Quantis QRNG USB от ID Quantique [12] е едно от най-надеждните устройства на пазара, тъй като използва квантова физика за генериране на истинска случайност, което го прави практически неуязвимо за преднамерени атаки или влияния от околната среда. Въпреки това, това устройство не може да бъде закупено лесно поради изключително високата си цена, което затруднява достъпа до него за широк кръг потребители и организации.

Тези ограничения създават предизвикателства за потребителите, които търсят сигурни и надеждни хардуерни генератори на случайни числа, особено при избора между ограничената функционалност, липсата на документация или трудния достъп до най-качественото решение.

# Генерация на случайни числа

## Принципи на генериране на електронен шум

### Лавинен пробив

При напрежения на пробив над 7 волта се активира лавинен процес, при който електроните се сблъскват с други атоми, освобождавайки допълнителни носители на заряд. Това води до по-висока мощност на шума, но също така генерира сложни шумови компоненти, включително шум в система с много състояния [13].

### Термичен шум

Термичният шум е резултат от движението на заряди в резистор при определена температура и е един от основните видове шум, използван като стандарт. Шумовото напрежение, генерирано от резистор, зависи от температурата, стойността на съпротивлението и честотната лента. Например, 1 kΩ резистор при стайна температура и честотна лента от 10 kHz генерира RMS шумово напрежение от приблизително 400 nV. Генераторите на термичен шум често използват два резистора с различни температури, които се превключват, за да създадат изходен шум [14].

### Шум от токов удар (Shot Noise)

Шумът от токов удар възниква, когато електрони преминават през бариера и пристигат на дискретни интервали. Тези случайни пристигания създават шум, чиято интензивност се контролира чрез промяна на постоянния ток през бариерата. Често като бариера се използва диод, който е лесен за интеграция в шумови генератори. Интензивността на шума се настройва чрез регулиране на тока, преминаващ през диода [14].

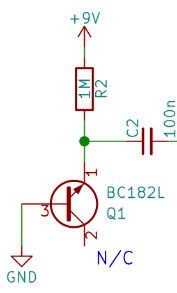
### Ценеров ефект

Преобладава при напрежения на пробив под 7 волта и се основава на вътрешна полева емисия. Този тип шум е прост и се класифицира като шум от токов удар [15].

## Изселдвана схема за лавинен шум !!!!

### Основни компоненти и тяхната роля:

Схемата работи със захранване по-голямо от 9V DC захранване. Това напрежение е критично, тъй като определя режима на пробив на транзистора.



Фигура ‑ Лавинен генератор на бял шум

### Резултати

#### Фигура 3‑2 показва генерираният случаен сигнал.



Фигура ‑ Генериран шум

Равномерно разпределеният шум се отличава с висока амплитуда –амплитудата от край до край (peak-to-peak) достига до 80 миливолта. Този сигнал е стабилен и непредсказуем, което го прави подходящ за генериране на висококачествена ентропия в приложения, изискващи случаен шум.

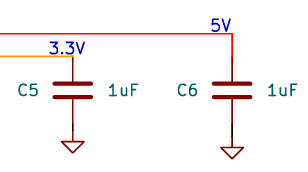
В приложение !!!! са сложени допълнителни изображения.

## Хардуер

Хардуерната схема се базира на схемата от Фигура 3‑1, в която шумът се генерира от вътрешните шумови характеристики на емитер-базовия преход на NPN транзистор, работещ в режим на пробив.

### Захранване

Схемата се захранва през захранващата линия на микроконтролера, който получава 5 волта от USB връзка. За да се изглади сигнала от захранването са поставени успоредно изглаждащи кондензатори, както е показано на Фигура 3‑3. По USB стандарт максималният ток, който може да осигури захранването, е 500 mA, като номиналната консумация на микроконтролера е около 100 mA. Електронната система се захранва чрез пиновете Vbus, 3V3 out, за които са свързани в паралел изглаждащи кондензатори [16].



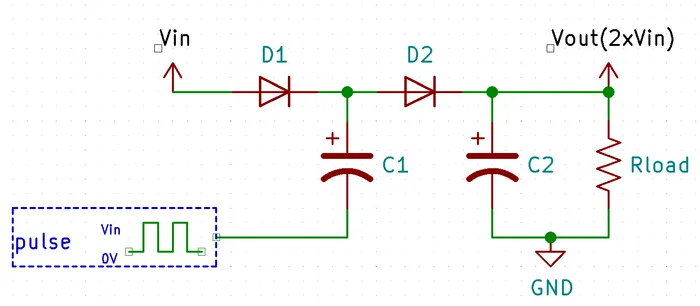
Фигура ‑ Изглаждащи кондензатори

### Помпа на заряд

Зарядната помпа е необходима, за да се постигне достатъчно високо напрежение, което да активира ефекта на транзистора, описан по-рано. Теория

Зарядната помпа, известна още като преобразувател с кондензатори, е предназначена за преобразуване на напрежението чрез използване на кондензатори, диоди и осцилаторен превключвател (Фигура 3‑3). Основният принцип на работа се състои в това да се зареждат кондензатори в паралел от входното напрежение и след това да се свържат последователно, за да осигурят по-високо изходно напрежение. В основната си форма, схемата използва два основни компонента: летящ кондензатор, който временно се изключва от веригата, за да пренесе заряд, и резервен кондензатор, който съхранява стабилизираното изходно напрежение.

Когато схемата се активира, входното напрежение зарежда кондензаторите чрез диоди, като напрежението се удвоява или увеличава в зависимост от топологията. Например, в схема за удвояване на напрежението, осцилаторен сигнал управлява превключването на кондензатора между зареждане и прехвърляне на заряда към изходния кондензатор, стабилизирайки напрежението след няколко цикъла. Шотки диодите са предпочитани заради ниския спад на напрежение и бързодействието им. Този тип схеми са ефективни при ниска мощност, работят с нисък шум и изискват минимален брой компоненти, което ги прави подходящи за маломощни приложения [17].



Фигура ‑ Едностъпална помпа на заряд

#### Реализация

Зарядната помпа е проектирана като двустепенна схема(показано в Фигура 3‑4), която позволява увеличаване на напрежението над 11 волта. За целта са използвани шотки диоди, които са предпочетени заради ниското си напрежение на пробив и по-малките загуби при превключване. Управлението се осъществява чрез широчинно-импулсна модулация (PWM) на 3.3 волта, което, заедно със захранването от на USB-то 5 волта , осигурява достигането на желаното изходно напрежение. Допълнително, бобината в схемата играе ключова роля за изглаждане на импулсите от зареждането, което стабилизира напрежението и намалява пулсациите в изхода.

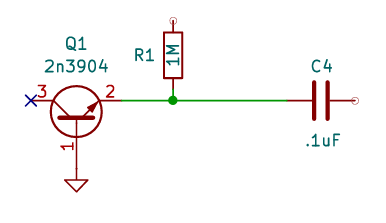
Формулата за изчисляване на крайното напрежение е в Приложение 1

#### 

Фигура ‑ Помпа на заряд с шотки диоди

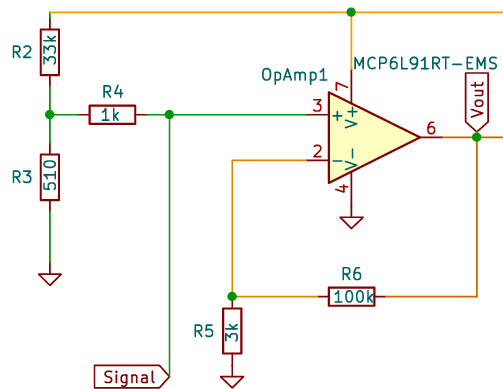
### Шумогенератор

Шумогенераторът показан на Фигура 3‑5 работи на принципа „Лавинен пробив“!!!



Фигура ‑ Шумогенератор

### Усилване не сигнал



Фигура ‑ Усилвател и делител на напрежение

За усилване на сигнала е използван операционен усилвател, свързан в неинвертиращ режим [18]. На неинвертиращия вход е зададено отместване, което е прецизно изчислен, така че финалният шум да бъде центриран спрямо средата на референтна стойност на АЦП-то. Теоретичното усилване на усилвателя е зададено на 33 пъти, постигнато чрез използването на резистори със стойности 3 kΩ и 100 kΩ в обратната връзка. Схемата е прикачена на Фигура 3‑6. Описания и сметки могат да се намерят в Приложение 2

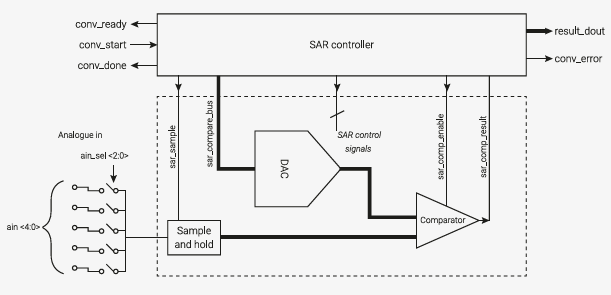
### АЦП

#### Възможности

RP2040 разполага с вграден 12-битов SAR ADC (Successive Approximation Register), който комбинира цифров контролер и аналогова схема за висока ефективност. Той поддържа максимална скорост на пробовземане от 500ksps, захранван от независим 48MHz тактов сигнал, и осигурява 8.7 ефективни бита (ENOB) за прецизно измерване на аналогови сигнали. ADC модулът включва пет входни мултиплексора, като четири от тях са достъпни през GPIO пинове 26–29, предоставяйки гъвкавост за свързване на външни сензори и устройства [19].

#### Как работи SAR ADC на теория

SAR ADC работи на принципа на последователното приближение. Това означава, че сравнява входния сигнал с вътрешен референтен сигнал и постепенно уточнява стойността на преобразувания сигнал, докато достигне точната цифрова стойност (12-бита). На Фигура 3‑7 е показана блоковата диаграма. ADC използва 48MHz тактова честота и всяка проба се извършва за 96 тактови цикъла, което отнема 2μs на проба, осигурявайки максималната скорост от 500ksps [19].



Фигура ‑ SAR ADC Блокова диаграма

### Използвани компоненти и консумацията им

#### Мотиви за избор

##### Pico vs. Arduino Nano

Raspberry Pi Pico и Arduino Nano имат различни характеристики, които ги правят подходящи за различни приложения. Raspberry Pi Pico разполага с два ядра Cortex-M0+ и тактова честота до 133MHz, докато Arduino Nano използва 8-битов AVR микроконтролер (ATmega328P) с тактова честота от 16MHz, което прави Pico значително по-бърз и способен на паралелна обработка. Вграденото ADC на Pico поддържа скорост до 500ksps, докато това на Nano е ограничено до 10-битово резолюционно преобразуване с 15kSPS, което поставя Pico напред за прецизни и бързи аналогови сигнали. Pico има и вградена USB функционалност, позволяваща реализация на устройства като клавиатура или мишка, докато Nano изисква допълнителен хардуер за подобни функции. Освен това, програмируемите входно-изходни модули (PIO) на Pico осигуряват значително по-голяма гъвкавост в работата с протоколи и специализирани комуникации, функция, която липсва при Nano. Nano обаче е по-просто устройство с по-ниска консумация на енергия и е по-лесно за работа в проекти, които не изискват високопроизводителни изчисления или разширени функционалности [19] [20].

##### SMD – through-hole

Използването на SMD компоненти в проекта е основно продиктувано от необходимостта да се постигне компактност и ефективност в дизайна. Тъй като разработвам shield за Raspberry Pi Pico, ключов приоритет е размерът да бъде малък и дизайнът – максимално опростен и интегриран. SMD компонентите са идеалният избор за тази цел, тъй като те заемат значително по-малко пространство в сравнение с through-hole компонентите и позволяват по-висока плътност на монтаж върху печатната платка. Това не само прави целия модул по-компактен, но също така улеснява монтажа върху Pico, което е от решаващо значение при работа с ограничено пространство. Освен това, SMD компонентите позволяват по-добра автоматизация на процеса на производство и спояване, което води до по-прецизен и надежден краен продукт. Through-hole компонентите, макар и подходящи за определени приложения, не могат да осигурят необходимата компактност и лекота на интеграция, която е необходима за shield, предназначен за директно монтиране върху микорконтролера.

#### Спецификация (консумация)

Компонентите използвани за изграждане на устройството са 2n3904 Транзистор, Raspberry Pi Pico, MCP6291-E/MS операционен усилвател, SS14 Шотки диод, OSTB8BS4C2B светодиод, SMD резистори и кондензатори с корпус 1206. Описание и спецификации в Приложение 3

### Схема

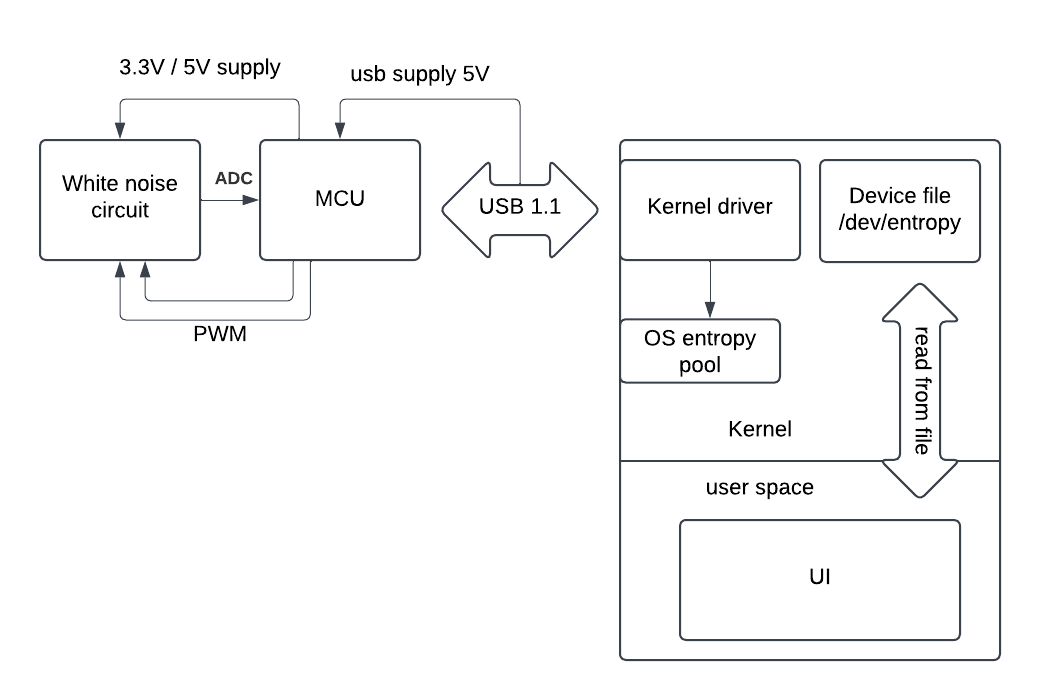
### 

Фигура ‑ Финална схема

### Изработка на платка

Използва KiCAD 8, 3D printer ELEGOO Saturn 3 ultra, UVtools software и химични преработки, където изработката на платката включва експонира на фоторезиста върху медната платка за да се открият зоните, от които трябва да се махне медта. След експонацията с UV светлина експонираните области се отмиват с подходящ разтворител. Следващата стъпка е ецването на медната повърхност под падналият фоторезист с киселина, за да се премахнат нежеланите зони. Накрая останалият фоторезист, който пази медта се отстранява от пътечките, разкривайки готова платка. Целият процес е описан в Приложение 4

### Блокова схема



Фигура ‑ Блокова схема на генератора на случайни числа

## Софтуер

### Управление на помпа на заряд

void setup\_PWM(uint gpio) {

    gpio\_set\_function(gpio, GPIO\_FUNC\_PWM);

    uint slice\_num = pwm\_gpio\_to\_slice\_num(gpio);

    pwm\_config config = pwm\_get\_default\_config();

    pwm\_config\_set\_clkdiv(&config, 32.0f);

    pwm\_config\_set\_wrap(&config, 49);

    pwm\_init(slice\_num, &config, true);

    pwm\_set\_gpio\_level(gpio, 40); // Set duty cycle ( 40/50 --> 80% )

    // 80 kHz with 32 prescale Divider

}

setup\_PWM(uint gpio):

Тази функция конфигурира определен GPIO пин за работа с PWM (Pulse Width Modulation) на микроконтролер

gpio\_set\_function(gpio, GPIO\_FUNC\_PWM):

Тази команда задава режима на работа на GPIO пина като PWM изход. След това пинът може да генерира PWM сигнали.

uint slice\_num = pwm\_gpio\_to\_slice\_num(gpio):

Тази команда връща съответния PWM slice номер (група от хардуерни регистри за управление на PWM). Всеки GPIO пин с PWM функция е свързан към конкретен slice.

pwm\_config config = pwm\_get\_default\_config():

Това създава основна конфигурация за PWM. Използва се стойността по подразбиране, която след това ще бъде променяна.

pwm\_config\_set\_clkdiv(&config, 32.0f):

Тази команда задава константа на делител за тактовия сигнал, който управлява PWM. В случая стойността е 32, което означава, че тактовият сигнал за PWM ще бъде разделен с 32.

pwm\_config\_set\_wrap(&config, 49):

Тази команда задава горната граница на броя на тактовете, след които се връща на нула (период на PWM сигнала). В този случай максималният брой тактове е 49, което означава, че PWM периодът е 50 такта.

pwm\_init(slice\_num, &config, true):

Инициализира PWM модула с конфигурираната стойност. Третият параметър (true) означава, че PWM ще започне да излъчва веднага след инициализацията.

pwm\_set\_gpio\_level(gpio, 40):

Тази команда задава дължината на импулса на PWM сигнала (или така наречения "duty cycle"). В този случай стойността 39 показва, че импулсът ще бъде с 40 единици на всеки 50 (съответства на около 80% duty cycle) [21].

### АЦП на второ ядро

adc\_init();

adc\_gpio\_init(ANALOG);

adc\_select\_input(0); // Assuming ADC0 is used

multicore\_launch\_core1(readValues);

void readValues()

{

    for (;;)

        enqueue(adc\_read());

}

adc\_init(); - Инициализира ADC.

adc\_gpio\_init(ANALOG); - Настройва пина ANALOG за аналогов вход.

adc\_select\_input(0); - Избира канал 0 на ADC за четене.

multicore\_launch\_core1(readValues); - Стартира второто ядро на процесора, което ще изпълнява функцията readValues().

readValues() - Непрекъснато чете стойности от ADC и ги поставя в опашка за последваща обработка [21].

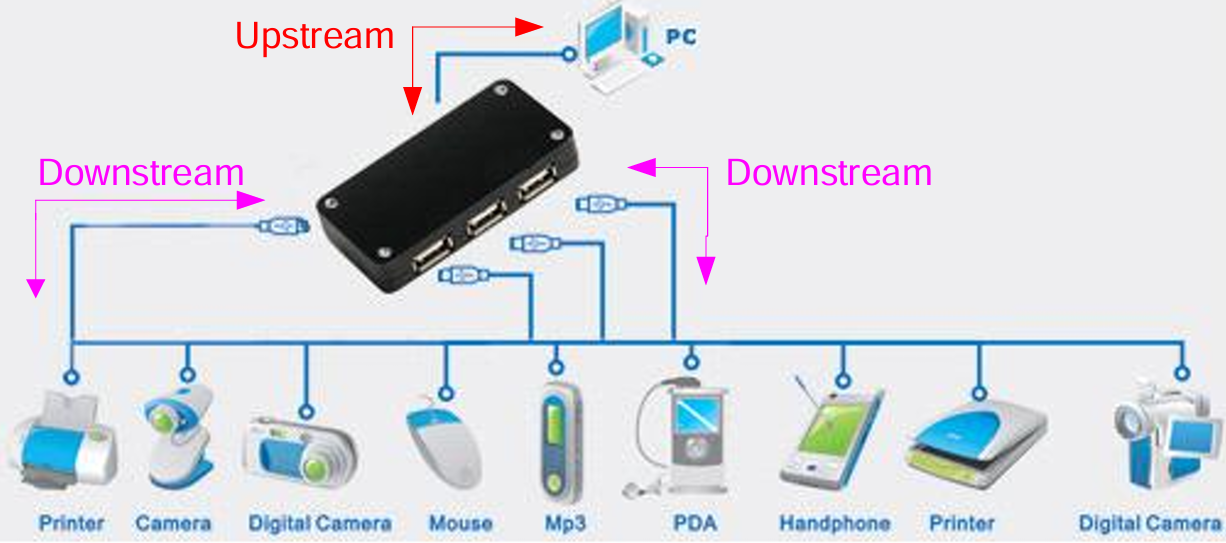
# USB комуникация за Entropy Device

## USB

### USB архитектура

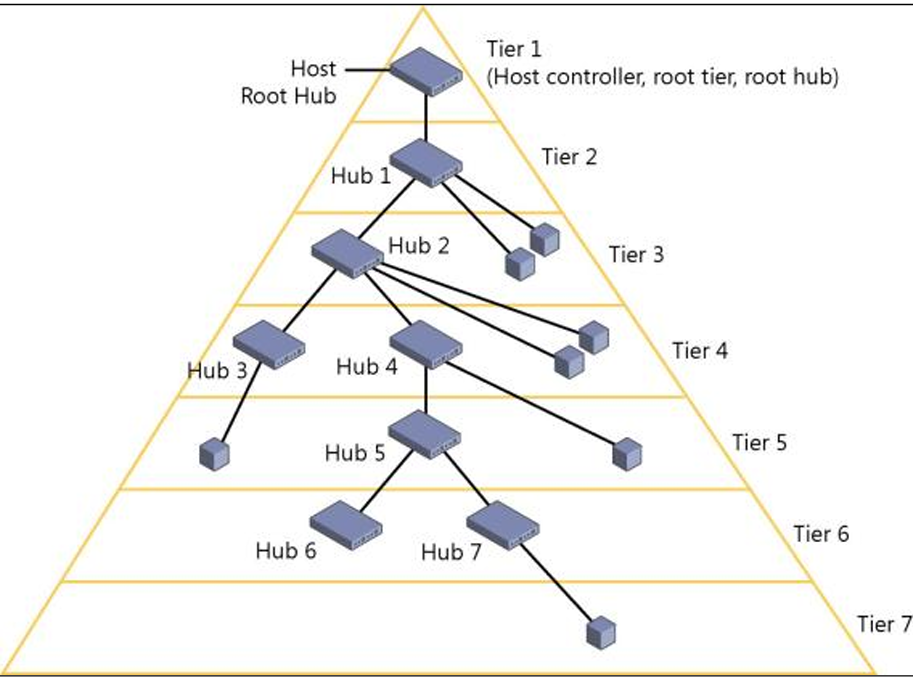
USB архитектурата включва един хост, който управлява комуникацията с устройства, които са свързани към него като "downstream" компоненти [16].

Показана е логическа схема на фигура Фигура 2‑1.



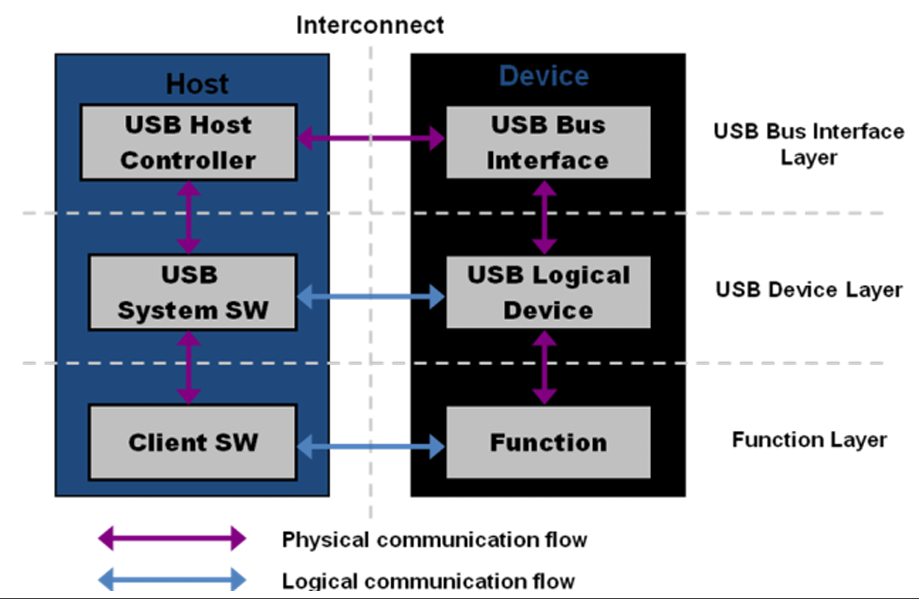
Фигура ‑ Много периферни могат да се свържат в един хост

Хостът контролира всички трансфери на данни, като OUT трансферите изпращат данни към периферни устройства, а IN трансферите получават данни от тях. Има три основни типа хост контролери: UHCI, OHCI и EHCI, които поддържат различни скорости на трансфер за различни версии на USB. Хъбовете играят важна роля, като позволяват свързването на до 127 устройства с хоста, като повторят трафика чрез един "upstream" порт [16]. Илюстрирана е примерна свързаност на различни нива на Фигура 4‑2.



Фигура ‑ Нива на свързаност

USB системата е организирана в три основни слоя: Bus Interface Layer, който отговаря за физическата свързаност и сигнализация, Device Layer за обработка на данни от хардуера, и Function Layer за изпълнение на специфични функции от софтуера на устройството [16]. На Фигура 4‑3 е визуализирана абстракцията на интерфейса



Фигура ‑ Абтракция на интерфейса

### USB скорости

USB спецификацията дефинира четири скорости за USB системи: Low-Speed, Full-Speed, Hi-Speed и SuperSpeed.

Разликите между различните скорости включват:

Low-Speed устройства работят на 1.5 Mbps и обикновено се използват за периферни устройства като клавиатури, мишки и игрови джойстици. Максималната ефективна скорост на данни за тези устройства е 800 B/s.

Full-Speed устройства оперират на 12 Mbps и включват устройства като телефони, аудио устройства и компресирани видеокамери, като максималната ефективна скорост на данни е 1.2 MB/s.

High-Speed устройства работят на 480 Mbps и обикновено включват видеокамери, имиджинг и съхранение на данни, като максималната ефективна скорост на данни е 53 MB/s.

SuperSpeed устройства работят на 10 Gbps и са част от по-новите USB 3.1 устройства.

Съществува важна разлика в начина, по който устройствата откриват своята скорост при свързване към хост. Това става чрез използване на pull-up резистори на линиите D+ или D-. Например, ако има 1.5-kΩ pull-up резистор на D+ линията, това показва, че свързаното устройство е Full-Speed. Ако на D- линията има 1.5-kΩ pull-up резистор, това показва, че устройството е Low-Speed. High-Speed устройствата започват като Full-Speed, като поставят pull-up резистор на D+ линията и изпращат серия от J- и K-състояния по време на фазата на рестартиране на нумерацията. Ако хъбът поддържа High-Speed, pull-up резисторът се премахва.

Разпространено недоразумение е, че USB 2.0 устройство винаги е High-Speed. Въпреки че всички High-Speed устройства са съвместими с USB 2.0, USB 2.0 спецификацията включва и Full-Speed и Low-Speed устройства. Освен това, различните скорости влияят на времето за битовете при USB сигнализацията. Например, за Full-Speed устройства се използва 48 MHz тактова честота, което води до 4 такта на бит, докато за Low-Speed устройства същата честота води до 32 такта на бит [16].

### USB крайни точки (Endpoints)

В USB спецификацията, endpoint е уникално адресируем компонент на USB устройство, който е източник или резервоар на информация в комуникационния поток между хоста и устройството. В процеса на USB нумерация и конфигурация, устройството отговаря на стандартния адрес, преди хостът да започне да чете други дескриптори, включително дескрипторите на endpoint-ите. По време на този процес, се използват специални endpoint-и, известни като Control Endpoint или Endpoint 0, които включват Endpoint 0 IN и Endpoint 0 OUT. Въпреки че тези два endpoint-а са технически отделни, те действат като един и същ от гледна точка на разработчика. Всеки USB устройство трябва да поддържа Endpoint 0 и за него не е необходим отделен дескриптор.

Освен Endpoint 0, броят на поддържаните endpoint-и в устройството зависи от проектните изисквания. Например, простото устройство като мишка може да се нуждае само от един IN endpoint, докато по-сложни устройства може да изискват множество data endpoint-и. USB спецификацията определя лимити за броя на endpoint-ите, като за High-Speed и Full-Speed устройства е позволено до 16 IN и 16 OUT endpoint-а (общо 32), докато Low-Speed устройствата са ограничени до два endpoint-а. Важно е да се отбележи, че data endpoint-ите са bidirectional, но когато са конфигурирани, те стават unidirectional – например, Endpoint 1 може да бъде IN или OUT, в зависимост от конфигурацията на дескриптора на устройството.

USB системата използва циклични контролни суми (CRC) за откриване на грешки в транзакциите. CRC е изчислена стойност, използвана за проверка на грешки, и ако данните и CRC не съвпадат, транзакцията се повтаря.

Съществуват четири основни типа endpoint-и, всеки със специфични характеристики:

Control Endpoints – Поддържат контролни трансфери, които всички устройства трябва да поддържат. Тези трансфери изпращат и получават информация за устройството и осигуряват висока точност и корекция на грешки.

Interrupt Endpoints – Поддържат трансфери с прекъсване, които се използват за устройства, които трябва да комуникират малки количества данни с висока надеждност. Въпреки името си, тези трансфери не са действителни прекъсвания, а използват периодично осъществяване на проверка.

Bulk Endpoints – Поддържат bulk трансфери, които са полезни за прехвърляне на големи количества данни при променлива честота. Те осигуряват точност, но времето за доставка не е гарантирано, тъй като зависи от наличната пропускателна способност на шината.

Isochronous Endpoints – Поддържат isochronous трансфери, които са непрекъснати и времеви критични с предоговарена пропускателна способност. Тези трансфери осигуряват гарантирано доставяне на данни, но без възможност за корекция на грешки, което ги прави подходящи за приложения като стрийминг аудио и видео.

Всеки тип endpoint има различни характеристики на максимален размер на пакета и скорост на трансфер в зависимост от устройството и неговата скорост. Например, Full-Speed устройства поддържат максимум 64 байта за Interrupt и Bulk трансфери, докато за Isochronous трансфери размерът на пакета е 1023 байта (за Full-Speed) или 1024 байта (за High-Speed) [16].

### Комуникационен протокол

USB комуникацията се състои от поредица от рамки, всяка от които включва Start of Frame (SOF) и една или повече транзакции. Тези транзакции се състоят от пакети, които са предшествани от синхронизационен (sync) шаблон и завършват с End of Packet (EOP) шаблон. В най-малкия случай транзакцията съдържа токен пакет, а в зависимост от типа на транзакцията, могат да бъдат включени допълнителни пакети за данни и ръчно потвърждение.

Структура на транзакцията:

Компоненти на транзакцията: Транзакцията се състои от три типа пакети:

Токен пакет: Започва транзакцията и предоставя информация за адреса и крайна точка.

Пакет с данни (по избор): Съдържа полезни данни, които се изпращат между хоста и устройството.

Пакет за потвърждение (по избор): Потвърджава успешното получаване на данни или сигнализира за грешка.

Транзакциите се групират в рамки и не се разделят по между им (с изключение на високоскоростните изокронни трансфери). Множество транзакции могат да се случат в една рамка, но те са отделни и не се прекъсват една друга.

Типове пакети:

Токен пакети: Насочват се от хоста, за да започнат и управляват потока на трафика по шината. Видовете включват:

IN токен: Изисква данни от устройството.

OUT токен: Изпраща данни от хоста към устройството.

SETUP токен: Предхожда команда от хоста.

SOF токен: Отбелязва началото на рамката за синхронизация.

Пакети с данни: Съдържат полезни данни и се изпращат след съответния токен пакет. Размерът на полезните данни може да варира в зависимост от типа на трансфера, като използвате механизъм за превключване (toggle), за да откривате грешки чрез сравняване на идентификаторите на пакета (DATA0 и DATA1).

Пакети за потвърждение: Завършват транзакциите и служат за потвърждение от страна на получателя. Видовете включват:

ACK: Потвърждение за успешно завършена транзакция (за всички скорости на USB).

NAK: Негативно потвърждение, което показва, че устройството не е готово да продължи.

STALL: Отговор за грешка, изпратен от устройството.

NYET: Указва, че устройството не е готово за друг пакет с данни (само за висока скорост).

Специални пакети: Тези пакети помагат при специфични сценарии, като например работа с устройства с ниска скорост или различия в скорости. Видовете включват:

PRE: Указва, че следващият пакет ще бъде с ниска скорост.

SPLIT: Започва транзакция с разделяне (само за висока скорост).

ERR: Докладва грешки по време на разделена транзакция (само за висока скорост).

PING: Проверява състоянието след получаване на NYET потвърждение (само за висока скорост).

Видове транзакции:

IN/Четене/Нагоре Транзакции:

Започват от хоста с IN токен, устройството отговаря с пакети с данни, а хостът потвърджава получаването чрез пакет за потвърждение.

OUT/Запис/Надолу Транзакции:

Започват от хоста с OUT токен, хостът изпраща пакети с данни и устройството завършва транзакцията с пакет за потвърждение.

Контролни транзакции:

Тези транзакции се използват за идентифициране, конфигуриране и контролиране на устройства. Контролите имат три етапа:

Етап на настройка: Хостът изпраща USB заявки към устройството.

Етап на данни: (по избор) Прехвърляне на данни между хоста и устройството.

Етап на потвърждение: Потвърдителна ръчна обработка за успех или неуспех на предишните етапи.

Варианти на Контролни Транзакции:

Контролно записване: Хостът изпраща данни към устройството.

Контролно четене: Хостът получава данни от устройството.

Контрол без данни: Хостът изпълнява контролни действия без прехвърляне на данни [16] [20].

### USB дескриптори

По-конкретни описания на полетата са предоставени в Приложение 5.

#### Дескриптор на устройство (Device Descriptor)

Дескрипторите на устройство предоставят информация на хоста, като например версията на USB спецификацията, към която устройството отговаря, броя на конфигурациите и поддържаните протоколи.

#### Конфигурационен дескриптор (Configuration Descriptor)

Конфигурационният дескриптор предоставя информация за конкретна конфигурация на устройството, като броя на интерфейсите, дали устройството е захранвано от шина или от външен източник, и колко мощност изисква.

#### Дескриптор на интерфейс (Interface Descriptor)

Дескрипторът на интерфейс описва конкретен интерфейс в рамките на дадена конфигурация. Броят на endpoint-овете и USB класът на устройството се дефинират тук. По-конкретни описания на полетата в приложение !!!!!!

#### Дескриптор на endpoint (Endpoint Descriptor)

Дескрипторът на endpoint предоставя информация за всяка крайна точка на устройството, включително типа на трансфера и посоката. По-конкретни описания на полетата в приложение !!!!!!

### USB устройства от клас

USB устройствата се класифицират в различни класове, като всеки от тях е предназначен за специфична функционалност. Human Interface Device (HID) включва устройства като клавиатури, мишки и джойстици, които позволяват директно взаимодействие с потребителя и използват контролни и прекъсващи трансфери с максимална скорост ~64 KB/s. Mass Storage Device (MSD) обхваща устройства като USB флашки и външни твърди дискове, предоставяйки надеждно съхранение на данни чрез групови трансфери. Communication Device Class (CDC) обхваща устройства като модеми и мрежови адаптери, които осигуряват комуникация чрез сериен интерфейс, използвайки контролни и групови трансфери с максимална скорост ~80 KB/s. Vendor Specific устройствата се отличават със своята персонализираност и се използват за специализирани приложения, като изискват специфични драйвери (например WinUSB, CyUSB или LibUSB). Тези класове осигуряват широка съвместимост с операционни системи, което улеснява разработката и използването на устройства [16] [20].

### USB изброяване и Конфигурация при зареждане

USB устройствата преминават през три основни етапа: динамично откриване, итерация и конфигурация. При динамичното откриване, когато устройство се свърже към USB порт, хъбът го разпознава чрез промяна в напрежението на линиите D+ и D-, като се активира съответен pull-up резистор според скоростта на устройството. Следва преоцес на изброяване, при който хостът открива новото устройство чрез съобщения от хъба и го ресетва, като издърпва линиите D+ и D- към земя. По време на ресета се проверява дали устройството поддържа висока скорост (High-Speed) чрез обмен на специфични сигнали (K и J състояния). След това хостът задава уникален адрес на устройството, което позволява директна комуникация. В етапа на конфигурация хостът извлича пълната информация за устройството чрез **GET\_DESCRIPTOR** команди, включително конфигурации, интерфейси и ендпойнти, и зарежда съответния драйвер въз основа на идентификатори като Product ID и Vendor ID. След като устройството бъде конфигурирано, то преминава в състояние "конфигурирано" и е готово за работа с приложението [16] [20].

## Raspberry Pico

### USB възможностите на Pico

Микроконтролерът RP2040, използван в Raspberry Pi Pico, разполага с USB 2.0 контролер, който може да работи като Full Speed устройство (12 Mbps) или като хост, комуникиращ както с Low Speed (1.5 Mbps), така и с Full Speed устройства. Контролерът включва интегриран USB 1.1 PHY, който осигурява интерфейс между контролера и USB пиновете DP и DM на чипа. Хардуерът на контролера автоматично обработва основния USB протокол, освобождавайки програмиста от сложните ниско ниво задачи. Също така, той разполага с 4KB DPSRAM за конфигурация и буферни данни [19].

#### Режим хост (Host Mode)

В режим на хост, RP2040 може да комуникира както с Full Speed (12 Mbps), така и с Low Speed (1.5 Mbps) устройства. Освен това, контролерът поддържа връзка с множество устройства чрез USB хъбове, включително Low Speed устройства, свързани към Full Speed хъб. Хардуерът е способен да управлява до 15 interrupt endpoints, които се използват за събития като свързване/разкачване на устройства или съобщения от периферни устройства, като мишки или клавиатури [19].

#### Режим устройство (Device Mode)

Когато RP2040 функционира като USB устройство, той е съвместим с USB 2.0 Full Speed стандарта (12 Mbps) и поддържа до 32 крайни точки (Endpoints 0–15 както в посока вход, така и в изход). Освен това, контролерът позволява използването на Control, Isochronous, Bulk и Interrupt типове крайни точки, както и двойно буфериране за по-ефективна работа. От 4KB DPSRAM памет, 3840 байта са достъпни за буфериране, което позволява съхранението на до 60 буфера с по 64 байта [19].

### интерфейс с USB контролер

#### DPSRAM

DPSRAM (Dual Port SRAM) на USB контролера в RP2040 е 4KB (4096 байта) памет, използвана за съхранение на контролни регистри и буфери за данни. Тя е достъпна като 32-битова памет на адрес 0x50100000. DPSRAM, поддържа 8/16/32-битови достъпи (стандартните регистри обикновено поддържат само 32-битови достъпи) и не поддържа set/clear алиаси, за разлика от повечето регистри на RP2040.

Данните обикновено се съхраняват в буфери от 64 байта, тъй като това е максималният размер на нормален Full Speed пакет. За Isochronous ендпойнтове се поддържат буфери с максимален размер от 1023 байта [19].

#### Едновременен достъп

DPSRAM е асинхронна и неатомарна памет. Тя е двупортова, което позволява както процесорът, така и USB контролерът да четат/записват едновременно. Това може да доведе до несъответствия, ако контролерът чете, докато процесорът записва. За избягване на такива сценарии се използва AVAILABLE битът в контролния регистър на буфера:

Процесорът задава AVAILABLE бит на 1, за да предаде контрола на буфера на USB контролера.

Контролерът връща AVAILABLE бит на 0, след като е обработил буфера [19].

#### Структура на паметта

DPSRAM е разделена на два основни региона:

DPSRAM (Dual-Port SRAM) е структурирана в два основни региона, които изпълняват различни функции. Първият регион обхваща контролни регистри с адреси от 0x0 до 0xFF. Тези регистри се използват за конфигуриране на ендпойнтовете и управление на буферите, което осигурява правилното функциониране на USB комуникацията. Вторият регион съдържа буфери за данни с адреси от 0x100 до 0xFFF. Този сегмент предоставя 3840 байта памет за съхранение на данни, прехвърляни чрез USB трансфери.

Разпределението на паметта варира в зависимост от режима на работа. В режим устройство (Device Mode) всеки ендпойнт разполага със собствени контролни и буферни регистри, защото хостът може да комуникира с множество ендпойнтове едновременно. В режим хост (Host Mode), обаче, съществуват само едни общи контролни и буферни регистри. В този случай хостът избира кой ендпойнт да използва, което опростява управлението на ресурсите, но изисква прецизен контрол от страна на хоста.

Регистрите в DPSRAM имат специфични функции. Контролният регистър на ендпойнтовете позволява конфигурирането на типа ендпойнт (Control, Isochronous, Bulk или Interrupt), задаването на базовия адрес на буферите, както и интервалите за полинг в interrupt режим. Освен това този регистър управлява активирането на прекъсвания за различни събития. Контролният регистър на буферите, от своя страна, следи текущото състояние на буферите, като указва дали са пълни или празни, както и дължината на трансфера. Този регистър също така съдържа бит за достъпност (AVAILABLE), който показва дали буферите са готови за използване от USB контролера. При работа в режим на двойно буфериране, системата автоматично превключва между буферите, освен ако процесорът не зададе "reset buffer select" бит за ръчно управление [19].

### Vendor-specific идентификация

Това е процес на разпознаване и настройка на комуникацията между хост и USB устройство чрез специфични за производителя параметри [23].

#### Закачане на USB устройството

Когато USB устройство се включи в хост система, USB контролерът на хоста открива новото устройство чрез промяна на състоянието на захранващия пин (VBUS). След това, хостът инициира процес на ресетване на шината, за да се синхронизира с устройството и да установи комуникация.

Хостът започва да изпраща SOF (Start-of-Frame) пакети, за да синхронизира времевия домейн.

Устройството преминава в състояние Default State, като сигнализира, че е свързано.

#### Задаване на адрес на устройството

Хостът изпраща SET\_ADDRESS команда чрез контролен трансфер към устройството. Това включва уникален адрес, който устройството трябва да използва за идентификация на USB шината.

Устройството приема адреса и преминава в състояние Addressed State.

От този момент, устройството отговаря само на пакети, адресирани към него.

#### Конфигуриране на устройството

Хостът изпраща серия от контролни трансфери, за да прочете описателите на устройството (Descriptors). Това включва:

Device Descriptor – съдържа основна информация за устройството, като тип, клас и идентификационен номер.

Configuration Descriptor – описва функционалностите на устройството и използваните ендпоинтове.

String Descriptors – съдържат текстови данни, като името на производителя и продукта.

След анализ на тези описатели, хостът изпраща SET\_CONFIGURATION команда, за да активира определена конфигурация на устройството. Устройството преминава в състояние Configured State.

#### Трансфериране на данни през ендпоинтовете

След конфигурирането, устройството е готово за трансфери на данни през дефинираните ендпоинтове:

Контролни трансфери – използват се за допълнителна конфигурация и управление.

Булк трансфери – използват се за големи обеми от данни, като файлове.

Isochronous трансфери – подходящи за аудио и видео потоци, където латентността е критична.

Interrupt трансфери – предназначени за малки, но важни данни, като събития от клавиатура или мишка.

#### Vendor-specific идентификация

Хостът може да използва Vendor-specific команди, за да достъпи специфична функционалност на устройството. Тези команди са извън стандарта на USB и се дефинират от производителя.

Хостът изпраща команда чрез контролен или bulk пакет към предварително зададен endpoint.

Устройството обработва командата и връща отговор, който може да съдържа данни или потвърждение за успешна операция.

#### Завършване на комуникацията

След приключване на трансферите, хостът може да изключи устройството или да го ресетира за нова конфигурация. Ако устройството бъде изключено, контролерът на хоста спира комуникацията и освобождава ресурси.

### Цикличен буфер на случайни числа

Кодът имплементира lock-free кръгова опашка (circular queue), която позволява две нишки да работят паралелно върху опашката: една, която добавя данни (enqueue), и друга, която премахва данни (dequeue). Използва се атомарна операция \_\_atomic\_compare\_exchange\_n, за да се гарантира сигурност при едновременен достъп до споделените ресурси (head и tail указателите) [21].

#### Код

struct queue

{

    union

    {

        struct

        {

            uint8\_t \* head, \*tail;

        };

        uint64\_t atomic;

    };

};

enum {buffer\_size = 1 << 17};

static uint8\_t buf[ buffer\_size ];

static struct queue queue = {{{buf, buf}}};

bool enqueue(uint8\_t data)

{

    struct queue old = queue, new = old;

    do

    {

        // Check if the queue is full

        if ((old.head >= old.tail && (old.head - old.tail == buffer\_size - 1)) ||

            (old.head < old.tail && (old.tail - old.head == 1)))

        {

            return false;

        }

        \*new.head = data;

        // Move head to the next position, wrapping around if necessary

        if (old.head + 1 >= buf + buffer\_size)

            new.head = buf;

        else

            new.head = old.head + 1;

    } while (!\_\_atomic\_compare\_exchange\_n(&queue.atomic, &old.atomic, new.atomic, true, \_\_ATOMIC\_RELEASE, \_\_ATOMIC\_RELAXED));

    return true;

}

bool dequeue(void \*ptr, int size)

{

    struct queue old = queue, new = old;

    do

    {

        // Check if the queue is empty or there is not enough data to dequeue

        int available\_data = (old.head >= old.tail) ? (old.head - old.tail) : (buffer\_size - (old.tail - old.head));

        if (available\_data < size)

        {

            return false;

        }

        // Calculate the new tail position

        if (old.tail + size >= buf + buffer\_size)

        {

            // Handle wrap-around

            int sz1 = buf + buffer\_size - old.tail; // Space available till end

            int sz2 = size - sz1;                  // Remaining size after wrapping

            memcpy(ptr, old.tail, sz1);             // Copy the end part

            memcpy((uint8\_t \*)ptr + sz1, buf, sz2); // Copy second part

            new.tail = buf + sz2;                   // Update tail after wrap

        }

        else

        {

            memcpy(ptr, old.tail, size);

            new.tail = old.tail + size;

        }

    } while (!\_\_atomic\_compare\_exchange\_n(&queue.atomic, &old.atomic, new.atomic, true, \_\_ATOMIC\_RELEASE, \_\_ATOMIC\_RELAXED));

    return true;

}

#### Описание на кода

enqueue(uint8\_t data)

Функцията enqueue добавя елемент в опашката, като първо проверява дали опашката е пълна чрез изчисляване на разликата между указателите head и tail. Ако има място, данните се записват в буфера на позицията, посочена от head, след което указателят head се премества напред. Ако head достигне края на буфера, той се връща в началото (wrap-around). Накрая, за да се гарантира коректността при конкурентен достъп от множество нишки, се извършва атомарна проверка и актуализация на стойностите на опашката с помощта на функцията \_\_atomic\_compare\_exchange\_n.

Функцията dequeue извлича данни от опашката, като първо проверява дали има достатъчно налични данни за изтегляне. Това се осъществява чрез изчисляване на наличното количество данни в буфера, като ако заявеният размер size надвишава това количество, функцията връща false. В случай че буферът се "прекъсва" (wrap-around), данните се копират на две части: първо от текущата позиция на указателя tail до края на буфера, а след това от началото на буфера до оставащата дължина. След извличането указателят tail се актуализира, като се премества напред и отчита wrap-around, ако достигне края на буфера. Накрая, за да се осигури коректна синхронизация между нишките, се извършва атомарна проверка и обновяване на стойностите с помощта на \_\_atomic\_compare\_exchange\_n.

Функцията \_\_atomic\_compare\_exchange\_n е атомарна операция, която се използва за безопасно сравняване и замяна на стойности в многонишкови среди. Тя сравнява текущата стойност на дадена променлива с очаквана стойност и, ако съвпадат, записва нова стойност на същото място. В противен случай обновява очакваната стойност с текущата, без да извършва промяната. Тази функция гарантира синхронизация между нишките и е основен инструмент за изграждане на lock-free структури от данни. В кода тя осигурява безопасно актуализиране на общата опашка, позволявайки добавянето и премахването на елементи без използване на заключване, като същевременно предотвратява конфликтите между нишките [21].

### Индикации

!!!!

### USB Protocol implemetation

Имплементацията на USB устройство е базирана на вече създадените примери за raspberry pi pico. Като точният пример е dev\_lowlevel.c в pico-examples/usb/device/dev\_lowlevel/dev\_lowlevel.c [25].

#### Описание

Функционалностите са разделени на няколко основни компонента, които работят заедно, за да осигурят правилното функциониране на USB връзката. Кодът е приложен в Приложение 7:

На първо място, кодът дефинира структурата на USB устройството чрез неговите дескриптори. Те включват "device descriptor", "interface descriptor", "configuration descriptor", както и низове за идентификация на производителя и продукта. Тези дескриптори се използват от хоста, за да разбере какви са възможностите на устройството. Кодовата структура организира тези дескриптори в глобална конфигурация (usb\_device\_configuration), която съдържа информация за ендпойнтите (endpoints), техните дескриптори и буфери.

Основната функционалност на кода е обработката на USB заявки чрез функции като usb\_handle\_device\_descriptor, usb\_handle\_config\_descriptor и usb\_handle\_setup\_packet. Тези функции разпознават входящите заявки от хоста (като искане за дескриптор или конфигурация) и подготвят отговори, които се изпращат обратно. Освен това, функциите управляват обработката на специфични команди, като например задаване на адрес за устройството (usb\_set\_device\_address) и конфигуриране на устройството (usb\_set\_device\_configuration).

Друг ключов компонент е управлението на ендпойнтите. Кодът дефинира структура за конфигурация на ендпойнти, която съдържа информация за техните буфери и контролни регистри. Чрез функциите usb\_setup\_endpoint и usb\_setup\_endpoints, ендпойнтите се настройват спрямо спецификацията на устройството. Това позволява правилно предаване и приемане на данни.

Интеграцията с USB хардуера е постигната чрез usb\_device\_init, която инициализира контролера и регистрите, изчиства предишно състояние и включва USB PHY. Тази функция симулира "включване" на устройството към хоста, активирайки връзката. Освен това, interrupt handler-ът isr\_usbctrl обработва прекъсванията, които са критични за синхронната комуникация между хост и устройство.

Накрая, има множество помощни функции като usb\_prepare\_string\_descriptor и usb\_buffer\_offset, които се грижат за специфични задачи като преобразуване на низове в Unicode или изчисление на адреси в паметта. Тези функции оптимизират работата с ограничените ресурси на устройството.

Като цяло, кодът предоставя модулна и структурирана имплементация на USB протокол за устройство, включваща настройка на хардуер, обработка на заявки и управление на буфери и ендпойнти.

## Linux kernel driver

### Живот на кърнълски модул

Кърнълските модули са части от програмния код, които могат да бъдат зареждани и премахвани динамично от ядрото на операционната система Linux, без необходимост от рестартиране на системата. Те позволяват разширяване на функционалността на ядрото, като например обслужване на нов хардуер или допълнителни функции, по удобен и ефективен начин [22].

Динамичното зареждане на модули се осъществява чрез използването на системни инструменти като insmod за зареждане на модули и rmmod за премахване на такива. Тези инструменти взаимодействат директно с ядрото, за да добавят или премахнат код, който е специално създаден да бъде модул. При зареждане модулът преминава през два ключови етапа: инициализация и освобождаване [22].

module\_init е функцията, която се изпълнява при зареждането на модула в ядрото. Тя съдържа логиката за инициализация на модула, като например регистриране на драйвери, резервиране на ресурси или настройка на необходимите структури от данни.

module\_exit е функцията, която се извиква при премахването на модула. Нейната задача е да освободи всички заети ресурси, да отпише регистрираните драйвери и да върне системата в първоначалното ѝ състояние. [22]

### Rule файл за зареждане на драйвер на USB

За автоматично зареждане на драйвери за USB устройства, Linux използва udev, което позволява създаването на "rule" файлове. Тези файлове дефинират правила, които определят как да се обработват устройствата, когато те бъдат свързани към системата.

Този подход позволява на системата автоматично да зарежда драйвери при откриване на съвместими устройства, което подобрява потребителското изживяване и улеснява управлението на хардуера [23].

### Драйвери, отнасящи се до USB

Драйверите, свързани с USB, са софтуерни компоненти, които позволяват на операционната система да комуникира и управлява USB устройства. Те играят ключова роля в осигуряването на съвместимост и функционалност на различните видове USB хардуер. USB драйверите са разделени на няколко основни категории в зависимост от тяхната роля и нивото на взаимодействие с устройствата.

#### USB Host Controller Drivers

Драйверите за USB хост контролери осигуряват основната комуникация между операционната система и USB хост контролера – хардуерният компонент, отговорен за управлението на USB шината. Тези драйвери обработват задачи като откриване на свързани устройства, управление на захранването и осигуряване на пренос на данни между устройствата и компютъра. Примери за такива драйвери са тези за xHCI, EHCI и OHCI, които поддържат различни версии и типове USB стандарти [16] [20].

#### USB Device Drivers

Драйверите за USB устройства са отговорни за управлението на специфични USB устройства, свързани към системата. Те предоставят интерфейс между приложния софтуер и устройството, като гарантират, че данните се предават правилно и функционалността на устройството е достъпна за потребителя. Например, драйверите за USB мишки, клавиатури, камери или външни дискове попадат в тази категория. Те комуникират с хост контролера чрез съответните протоколи, за да осигурят плавна и надеждна работа на устройствата.USB Type-Specific Drivers [16] [20].

### Регистриране на USB драйвер

#### struct usb\_driver

static struct usb\_driver entropy\_driver = {

    .name   = "usb\_entropyeton",

    .id\_table = entropy\_table,

    .probe  = entropy\_probe,

    .disconnect = entropy\_disconnect,

};

Структурата usb\_driver в ядрото на Linux представлява описанието на USB драйвер, което включва информация за името му, таблицата с идентификатори на поддържаните устройства, както и указатели към функциите за обработка на събития като свързване и изключване на устройства. В дадения пример структурата entropy\_driver дефинира USB драйвер с име "usb\_entropyeton". Полето id\_table указва таблицата entropy\_table, която съдържа списък с идентификатори (vendor ID и product ID) на устройствата, които този драйвер може да разпознае и управлява. Полето probe посочва функцията entropy\_probe, която се извиква автоматично при свързване на устройство, съвпадащо с някой от идентификаторите в таблицата. Полето disconnect указва функцията entropy\_disconnect, която се извиква при изключване на устройството, за да се освободи заделената памет и да се извърши почистване. Тази структура представлява основата за регистриране на USB драйвер в ядрото и свързва функционалността на драйвера с конкретни събития и устройства [24].

#### Регистриране на USB драйвер: usb\_register

Функцията usb\_register се използва за регистриране на USB драйвер в ядрото на операционната система. Тя е от съществено значение за всяка USB функционалност, тъй като чрез нея драйверът информира ядрото, че е готов да управлява определени USB устройства. При регистрирането драйверът предоставя структура от тип usb\_driver, която съдържа информация като идентификаторите на устройствата, които поддържа, и функциите за обработка на събития като свързване или премахване на устройство. След успешното извикване на usb\_register, ядрото започва да следи за свързани устройства, които съответстват на описаните от драйвера идентификатори, и автоматично свързва тези устройства с регистрирания драйвер [24].

#### Премахване на USB драйвер: usb\_deregister

Функцията usb\_deregister се използва за премахване на USB драйвер от ядрото. Тази операция е необходима, когато драйверът вече не е нужен, или когато трябва да бъде актуализиран или заменен. При извикването на usb\_deregister, ядрото прекратява асоциацията на драйвера с всички свързани устройства и освобождава ресурсите, които драйверът е използвал. Тази функция гарантира безопасното деинсталиране на драйвера и предотвратява потенциални конфликти или изтичане на памет. Използването на usb\_deregister е важна част от поддръжката на модулите в ядрото, особено за динамично зареждащи се модули [24].

### Живот на USB device

#### probe

Тази функция се извиква, когато ново USB устройство бъде свързано към хост системата. Тя е част от процеса на откриване на устройства и е предназначена да провери дали свързаното устройство съвпада с типа устройства, който драйверът може да управлява. Ако устройството отговаря на изискванията, probe осигурява необходимите настройки и инициализация на устройството. Например, може да се създадат съответните файлове за устройства в /dev, да се започне комуникация с устройството и да се извършат други необходими стъпки за правилното му функциониране [24].

#### disconnect

Когато USB устройство бъде премахнато или изключено от хост системата, се извиква функцията disconnect. Тя осигурява безопасно прекратяване на всички операции, свързани с устройството, като освобождаване на ресурсите и премахване на всякакви връзки или файлови дескриптори, които може да са свързани с него. Освен това, функцията извършва почистване на драйвера и системата, за да предотврати изтичане на памет или други ресурси след изключването на устройството [24].

### Регистриране и премахване на USB устройства:

usb\_register\_dev: Тази функция е отговорна за регистрирането на USB устройства, които са открити и поддържани от съответния драйвер. След като устройството е открито чрез функцията probe, usb\_register\_dev създава съответната устройствена точка (например файл в /dev), чрез която потребителските приложения могат да взаимодействат с устройството. Тази стъпка гарантира, че драйверът за устройството е правилно свързан със системата и че устройството е готово за работа [25].

usb\_deregister\_dev: Когато устройството бъде премахнато чрез функцията disconnect, usb\_deregister\_dev е отговорна за премахването му от системата. Това включва изтриването на съответния запис в /dev и освобождаването на всякакви ресурси, свързани с устройството. Тази операция осигурява правилното и безопасно премахване на устройството от операционната система, предотвратявайки изтичания на памет или други потенциални проблеми след физическото премахване на USB устройството [25].

### Прехвърляне на данни от USB endpoint в bulk режим

Функцията usb\_bulk\_msg в ядрото на Linux се използва за синхронен трансфер на данни по USB чрез bulk (пакетни) канали, предназначени за големи обеми от данни с гаранция за коректност, но без гаранция за време на доставка. Тя приема няколко аргумента: указател към структурата на устройството (usb\_device), указател към буфер за данни, размер на данните, указател към променлива за броя на трансферираните байтове и таймаут. Функцията инициира трансфер на данни към или от устройството чрез предварително конфигуриран bulk endpoint. Работата ѝ е синхронна, като блокира изпълнението, докато трансферът приключи или изтече зададеният таймаут. При успех тя връща 0, а при грешка връща отрицателен код, описващ проблема. usb\_bulk\_msg е полезна за прехвърляне на големи количества данни към устройства като USB памети, принтери или скенери. Въпреки че е проста за използване, блокиращата ѝ природа може да доведе до забавяния, което я прави по-подходяща за случаи, в които не е необходима асинхронна обработка [24] [25] [26].

### Захранване на ОС с ентропия

probe и създаване на нишка за четене на данни от USB: В контекста на захранване на операционната система с ентропия, функцията probe играе важна роля. При откриването на USB устройство, което генерира случайни числа или данни, probe инициализира нишка, която започва да чете тези данни от USB устройството. Тази нишка е основна за получаването на случайни числа от външен източник (например USB устройство, което генерира ентропия), които ще бъдат използвани за криптографски цели или за други нужди, изискващи висока степен на случайност в операционната система.

Предаване на данни на кърнъла чрез add\_hwgenerator\_randomness: След като нишката е започнала да чете данни от USB устройството, тя използва функцията add\_hwgenerator\_randomness, за да предаде събраната случайност на ядрото на операционната система. Това се осъществява чрез добавяне на новопостъпилата ентропия в пул с данни за случайни числа, който се използва от ядрото за различни цели. Тази случайност може да бъде използвана за криптографски алгоритми, генериране на случайни числа за приложения или други дейности, които изискват надеждни и непредсказуеми източници на случайност. Този процес осигурява на операционната система стабилен и сигурен поток от случайни данни за нейни вътрешни нужди [27] [26].

### Файлов интерфейс за четене на суровите данни

#### Броене на отваряния на entropy в /dev

Процесът на броене на отваряния на entropy устройството се свързва с използването на механизма за управление на референтни броячи на устройства в ядрото, което е важно за правилното управление на ресурсите и за предотвратяване на изтичане на памет. Това се постига чрез използването на kref\_get и kref\_put за управление на броя на референтните връзки към устройството [28].

kref\_get(&dev->kref): Тази функция увеличава референтния брояч на структурата dev, която представлява устройството. При всяко отваряне на entropy устройството се извиква kref\_get, което означава, че броячът на референтните връзки се увеличава, за да се гарантира, че устройството не се освобождава, докато има активни връзки към него. Това е необходимо, за да се уверим, че устройството не бъде изтрито, докато е в употреба [28].

kref\_put(&dev->kref, skel\_delete): Когато устройството вече не е нужно, kref\_put се извиква, за да намали референтния брояч. Ако референтният брояч достигне нула, което означава, че няма активни връзки към устройството, функцията skel\_delete се използва за окончателно изтриване на устройството и освобождаване на свързаните с него ресурси. Това предотвратява изтичането на памет, като гарантира, че ресурсите на устройството се освобождават само когато не се използват повече [28].

#### Реализация на функцията за четене от user mode

Структурата file\_operations (в примера entropy\_fops) определя функциите, които се използват за взаимодействие с устройството от страна на потребителския режим. Полето owner указва, че модулът, който съдържа тези операции, е текущият модул (THIS\_MODULE), което предотвратява премахването му, докато устройството се използва. Полето read е свързано с функцията entropy\_read, която се извиква, когато потребителското приложение чете данни от устройството. Полето open посочва функцията entropy\_open, която се изпълнява при отваряне на устройството, а release се свързва с функцията entropy\_release, която се извиква при затварянето на устройството. [29] [26]

Структурата usb\_class\_driver (в примера entropy\_class) служи за регистриране на драйвера като част от USB класа. Полето name указва името на устройството в /dev, като устройствата ще бъдат достъпни с имена, базирани на посоченото име (например /dev/entropy). Полето fops сочи към entropy\_fops, което указва, че операциите за това устройство ще се обработват от функциите, дефинирани в тази структура. Полето minor\_base определя основния минорен номер за устройствата, регистрирани от този драйвер [29] [26].

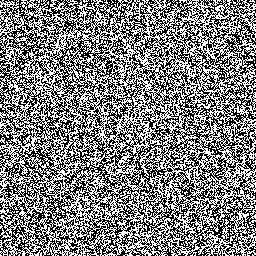
Четенето от потребителския режим се реализира чрез извикване на read системното повикване в потребителското приложение, което води до изпълнението на функцията entropy\_read в драйвера. В тази функция обикновено се осигуряват данни чрез копиране от ядрото към потребителската памет с помощта на функции като copy\_to\_user. Това позволява данни, генерирани или събрани от драйвера (например ентропия от хардуерния генератор), да бъдат достъпни за приложенията в потребителския режим [29].

# тестове на случайни числа

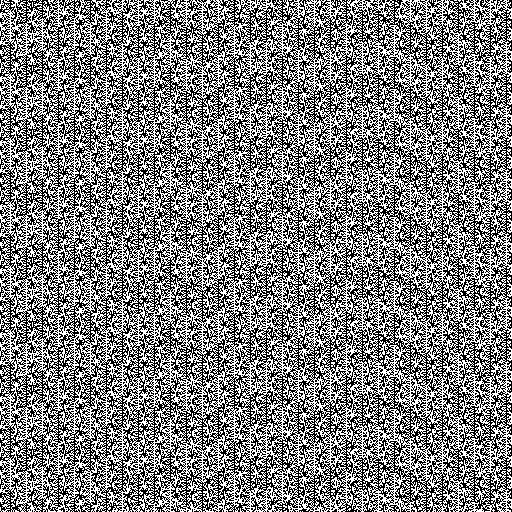
## Визуален анализ на тестове

### BITMAP

Един от най-простите методи за оценка на генератор на случайни числа е чрез визуален анализ на числата, които генерира. Хората са изключително добри в разпознаването на модели, а визуализацията предоставя интуитивен начин за оценка на качеството на генератора. Въпреки че този подход не е изчерпателен или формален, той предлага бърз и практичен начин за първоначално впечатление относно качеството на произволността.



Фигура ‑ random.org TRNG



Фигура ‑ PHP rand() PRNG

По-долу са показани два примера на битмапи, създадени през април 2008 г. от Бо Алън, за да сравни работата на два генератора на случайни числа. Битмапът вляво е генериран с помощта на Bitmap Generator на RANDOM.ORG, базиран на Истински Генератор на Случайни Числа (TRNG). Битмапът вдясно е създаден с функцията rand() на PHP на Microsoft Windows, която е Псевдо-Генератор на Случайни Числа (PRNG) [30].

### Битова хистограма

Битовата хистограма е инструмент за анализ на бинарни файлове, който се фокусира върху честотата на единиците (битове със стойност 1) спрямо целия брой битове в рамките на файла. Този метод разглежда файла като поток от бинарни данни, разделен на блокове от по 8 бита (1 байт), и изчислява съотношението на 1 спрямо общия брой битове (1 и 0) във всеки блок. Идеалното разпределение на битовете в случайни данни е 50% за 1 и 50% за 0. Отклонения от това съотношение могат да бъдат индикатор за липса на случайност или други несъответствия в данните.



Фигура ‑ Примерна битова хистограмаХистограма на байтове

Байтовата хистограма е метод за анализ на бинарни файлове, при който файлът се обработва байт по байт, а честотата на срещане на всяка възможна стойност (от 0 до 255) се записва и визуализира. Резултатът е хистограма с 256 категории (по една за всяка възможна стойност на байта). На графиката хоризонталната ос представя байтовите стойности (от 0 до 255), а вертикалната ос показва колко често всяка стойност се среща във файла.



Фигура ‑ Хистограма на разпределение на стойности

## NIST

### Frequency (Честота)

Тестът за честота проверява дали нулите и единиците в дадена бинарна последователност са разпределени равномерно. Той изчислява колко пъти се срещат 0 и 1 и сравнява резултатите със статистическия идеал. Ако броят на нулите и единиците е значително различен, това може да означава липса на произволност.

### Block Frequency (Честота в блокове)

Този тест разделя последователността на блокове с фиксиран размер и анализира честотата на нулите и единиците във всеки блок. Това позволява да се открият локални аномалии, които може да останат незабелязани при глобалния тест за честота.

### Cumulative Sums (Натрупани суми)

Натрупаните суми анализират сумата от битовете на последователността, където 1 се приема като +1, а 0 – като -1. Тестът измерва дали натрупаната стойност се отклонява значително от нулата. Това може да покаже тенденции или модели в последователността.

### Runs (Поредици)

Тестът за поредици проверява дали има прекалено дълги или прекалено къси последователности от последователни 0 и 1. Нормалната произволност предполага, че поредиците имат определена вероятностна дължина. Прекалено много или твърде малко поредици с определена дължина може да показват липса на произволност.

### Longest Run (Най-дълга поредица)

Този тест измерва дължината на най-дългата поредица от еднакви битове (само нули или само единици) в даден интервал. Ако дължината на най-дългата поредица се различава значително от очакваната, това е знак за липса на произволност.

### Rank (Ранг)

Тестът за ранг анализира матрици, формирани от битовете на последователността. Той проверява линейната зависимост на редовете в матриците. Отклоненията от очакваната честота на ранговете подсказват, че битовете не са случайно подредени.

### FFT (Бързо преобразуване на Фурие)

Тестът с FFT изследва честотния спектър на последователността, за да открие периодичност или повтарящи се модели. Ако спектърът показва неочаквани върхове, това може да означава наличие на систематични модели в данните.

### Non-Overlapping Template (Не-препокриващи се шаблони)

Този тест анализира колко често се среща конкретен шаблон от битове в последователността, без припокриване. Ако честотата на шаблона е извън очакваните граници, това е знак за липса на произволност.

### Overlapping Template (Препокриващи се шаблони)

За разлика от предходния тест, тук се отчита честотата на шаблоните, като се допуска припокриване. Този метод е по-чувствителен към повтарящи се модели.

### Universal (Универсален тест)

Универсалният тест измерва способността на последователността да компресира информация. Последователностите с висока степен на произволност са трудно компресируеми. Ако последователността може да се компресира значително, това е знак за ниска произволност.

### Approximate Entropy (Приблизителна ентропия)

Този тест изчислява ентропията на последователността, за да оцени непредсказуемостта на шаблоните. Високата ентропия е знак за добра произволност, докато ниската показва наличие на предвидими структури.

### Random Excursions (Случайни отклонения)

Тестът анализира траекторията на последователността като случайно движение. Той измерва как битовете се отклоняват от определена стойност (обикновено 0). Непредвидими отклонения са знак за произволност.

### Random Excursions Variant (Вариант на случайни отклонения)

Подобен на предходния тест, но измерва отклоненията по различни критични точки. Този тест е по-чувствителен към специфични модели в последователността.

### Linear Complexity (Линейна сложност)

Тестът измерва минималната дължина на линейния регистър със смяна на позиции (LFSR), който може да генерира последователността. Високата линейна сложност показва, че последователността е трудно предсказуема и има висока степен на произволност.

Множество статии са посветени на NIST тестовете, и техният списък ще бъде включен в Приложение 6.

# Потребителски интерфейс

## Комуникация на приложение с Entropy устройството

Комуникацията между потребителското приложение и Entropy устройството се осъществява чрез обикновено четене от специалния файл /dev/entropy, без да е необходимо използването на администраторски привилегии. Приложението отваря файла и чете произволни данни (ентропия), предоставяни от драйвера. Четенето се извършва с помощта на стандартни системни повиквания като open и read. Тази проста и интуитивна схема позволява приложенията лесно да получават достъп до данни, генерирани от хардуерния генератор на ентропия.

## Визуализация на шум

За визуализация на шума, генериран от Entropy устройството, данните се използват за създаване на bitmap изображение, което се изобразява на екрана. Визуализацията преобразува стойностите на ентропията в цветове или пикселни модели, като по този начин показва произволността и характеристиките на данните. С помощта на библиотеката SDL се създава прозорец, където генерираното изображение се изобразява в реално време. Визуализацията може също да се обновява динамично, като позволява на потребителя да наблюдава промените в генерирания шум.

## Използвани библиотеки

За изграждане на графичния интерфейс и визуализацията се използват библиотеките SDL и SDL\_ttf. SDL предоставя основните функции за управление на прозорци, рендеринг и обработка на събития, докато SDL\_ttf се използва за визуализиране на текст на екрана. Тези библиотеки предлагат удобен начин за създаване на кросплатформен потребителски интерфейс, който е лесен за разработка и интеграция. SDL също така осигурява лесен достъп до манипулация на пиксели, което улеснява създаването на bitmap изображения за визуализиране на данните от Entropy устройството [31] [32].

# Бъдещи подобрения

## ВЧ усилване на шума за подобряване на разпределението на числата

Едно от бъдещите подобрения е включването на високочестотен (ВЧ) усилвател за подобряване на качеството на генерираните случайни числа. Усилването на шума ще осигури по-добро разпределение на числата чрез увеличаване на амплитудата на високочестотните компоненти в сигнала, което ще повиши ентропията. Това е особено важно за приложения, които изискват висока степен на случайност и равномерност в разпределението, като криптографски системи и симулации. С добавянето на ВЧ усилвател ще се постигне още по-прецизно и надеждно генериране на случайни числа.

## Режим с ниска консумация

Друго потенциално подобрение е разработването на режим с ниска консумация на енергия за Entropy устройството, който ще се активира автоматично, когато операционната система няма нужда от допълнителна ентропия. В този случай устройството ще преминава в low-power режим и ще спира консумацията на енергия, докато отново не бъде изисквано. Това би било особено полезно за предотвратяване на ненужна употреба на ресурси и за удължаване на живота на системите, които разчитат на ограничено захранване, като лаптопи.

## Криптирана комуникация по USB с Diffie-Hellman

Друго важно бъдещо подобрение е добавянето на криптирана комуникация по USB, базирана на алгоритъма Diffie-Hellman за обмен на ключове. Това ще гарантира сигурността на данните, генерирани от Entropy устройството, особено когато се използват за криптографски цели. Чрез криптиране на комуникацията се предотвратява възможността за прихващане или манипулиране на данните от външни източници. Използването на Diffie-Hellman за установяване на сигурна връзка е ефективен начин за осигуряване на защитен канал между устройството и драйвера.

# Заключение

# Цитирани източници

|  |  |
| --- | --- |
| [1] | M. Boelen, „Linux audit,“ 6 1 2025. [Онлайн]. Available: https://linux-audit.com/gpg-key-generation-not-enough-random-bytes-available/. |
| [2] | „Wikipedia,“ 9 1 2025. [Онлайн]. Available: https://en.wikipedia.org/wiki/Pseudorandomness. |
| [3] | Wikipedia, „Random\_number\_generation,“ [Онлайн]. Available: https://en.wikipedia.org/wiki/Random\_number\_generation#True\_vs.\_pseudo-random\_numbers. |
| [4] | Wikipedia, „Entropy (computing),“ [Онлайн]. Available: https://en.wikipedia.org/wiki/Entropy\_(computing). |
| [5] | „Wikipedia,“ [Онлайн]. Available: https://en.wikipedia.org/wiki/Random\_number\_generator\_attack. |
| [6] | J. M. C. R. K. HOWELL, „engr.colostate,“ [Онлайн]. Available: https://www.engr.colostate.edu/~jdaily/presentations/2017%20Seed%20Key%20Exchange.pdf. |
| [7] | „Wikipedia,“ [Онлайн]. Available: https://en.wikipedia.org/wiki/Random\_number\_generator\_attack#Prominent\_examples. |
| [8] | manned.org, „random, urandom - kernel random number source devices,“ [Онлайн]. Available: https://manned.org/random.4. |
| [9] | [Онлайн]. Available: https://en.wikipedia.org/wiki/Cryptographically\_secure\_pseudorandom\_number\_generator. |
| [10] | J. Cheetham, „onerng,“ [Онлайн]. Available: https://onerng.info/#:~:text=OneRNG%20is%20an%20entropy%20source%20/%20hardware. |
| [11] | „ubld.it,“ [Онлайн]. Available: https://ubld.it/truerng\_v3. [Отваряно на 2013]. |
| [12] | [Онлайн]. Available: https://www.idquantique.com/random-number-generation/request-a-quote/. |
| [13] | [Онлайн]. Available: https://en.wikipedia.org/wiki/Avalanche\_breakdown. |
| [14] | [Онлайн]. Available: https://en.wikipedia.org/wiki/Noise\_generator. |
| [15] | [Онлайн]. Available: https://en.wikipedia.org/wiki/Zener\_effect. |
| [16] | R. Murphy, „infineon,“ [Онлайн]. Available: https://www.infineon.com/dgdl/Infineon-AN57294\_USB\_101\_An\_Introduction\_to\_Universal\_Serial\_Bus\_2.0-ApplicationNotes-v09\_00-EN.pdf?fileId=8ac78c8c7cdc391c017d072d8e8e5256. |
| [17] | [Онлайн]. Available: https://blog.mbedded.ninja/electronics/components/power-regulators/charge-pumps/. |
| [18] | „electronics-tutorials,“ [Онлайн]. Available: https://www.electronics-tutorials.ws/opamp/opamp\_3.html. |
| [19] | R. P. Ltd, „datasheets.raspberrypi.com,“ [Онлайн]. Available: https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf. |
| [20] | arduino, 2025. [Онлайн]. Available: https://docs.arduino.cc/hardware/nano/#tech-specs. |
| [21] | raspberrypi, „Hardware APIs,“ 2025. [Онлайн]. Available: https://www.raspberrypi.com/documentation/pico-sdk/hardware.html. |
| [22] | C. Stevens, „usb.org,“ [Онлайн]. Available: https://www.usb.org/sites/default/files/usbprint11a021811.pdf. |
| [23] | „xmos,“ [Онлайн]. Available: https://www.xmos.com/documentation/XM-006260-AN/pdf/AN00136\_v3.0.0.pdf. [Отваряно на 2024]. |
| [24] | „gcc.gnu,“ [Онлайн]. Available: https://gcc.gnu.org/onlinedocs/gcc/\_005f\_005fatomic-Builtins.html. |
| [25] | raspberrypi, „github,“ 2024. [Онлайн]. Available: https://github.com/raspberrypi/pico-examples/blob/master/usb/device/dev\_lowlevel/dev\_lowlevel.c. |
| [26] | „linux-kernel-labs,“ [Онлайн]. Available: https://linux-kernel-labs.github.io/refs/heads/master/labs/kernel\_modules.html. |
| [27] | „linuxconfig,“ [Онлайн]. Available: https://linuxconfig.org/tutorial-on-how-to-write-basic-udev-rules-in-linux. |
| [28] | „kernel,“ [Онлайн]. Available: https://www.kernel.org/doc/html/latest/driver-api/usb/writing\_usb\_driver.html. |
| [29] | „kernel\_host\_api,“ [Онлайн]. Available: https://www.kernel.org/doc/html/latest/driver-api/usb/usb.html. |
| [30] | M. Teslia, „apriorit,“ [Онлайн]. Available: https://www.apriorit.com/dev-blog/195-simple-driver-for-linux-os. |
| [31] | torvalds, „random\_c,“ [Онлайн]. Available: https://github.com/torvalds/linux/blob/master/drivers/char/random.c. |
| [32] | C. Minyard, „kref,“ [Онлайн]. Available: https://www.kernel.org/doc/html/latest/core-api/kref.html. |
| [33] | D. Fliegl, „lmu.web.psi,“ [Онлайн]. Available: https://lmu.web.psi.ch/docu/manuals/software\_manuals/linux\_sl/usb\_linux\_programming\_guide.pdf. |
| [34] | „random.org,“ [Онлайн]. Available: https://www.random.org/analysis/. |
| [35] | ghwikipp, „wiki.libsdl,“ [Онлайн]. Available: https://wiki.libsdl.org/SDL2/Installation. |
| [36] | ghwikipp, „wiki.libsdl.SDL2\_ttf,“ [Онлайн]. Available: https://wiki.libsdl.org/SDL2\_ttf/FrontPage. |
| [37] | „Wikipedia,“ [Онлайн]. Available: https://en.wikipedia.org/wiki/Random\_number\_generator\_attack#Attacks. |
| [38] | L. Vogel, „circuitlab,“ [Онлайн]. Available: https://www.circuitlab.com/circuit/8wzr22/white-noise-generator/. |
| [39] | amicoleo, „electro-music,“ [Онлайн]. Available: https://electro-music.com/forum/post-377900.html. |
| [40] | S. Chip, „eeweb,“ [Онлайн]. Available: https://www.eeweb.com/simple-white-noise-generator/. |
| [41] | [Онлайн]. Available: https://en.wikipedia.org/wiki/Cryptographically\_secure\_pseudorandom\_number\_generator. |
| [42] | [Онлайн]. Available: https://en.wikipedia.org/wiki/Noise\_generator. |

# Приложениe

## Приложение 1

5V (входно напрежение) - 0.3V (спад върху Шотки диод) + 3.3V (осцилаторно напрежение) - 0.3V + 3.3V - 0.3V (двоен заряден цикъл с още един диод).

## Приложение 2

## Приложение 3

Таблица ‑ Електронни компоненти

|  |  |
| --- | --- |
| Компоненти | описание |
| [2n3904 Transistor](https://store.comet.bg/Catalogue/Product/21440/) | Maximum ratings: 40V (VCEO), 200mA (IC), 350mW power dissipation |
|  | High frequency: 300 MHz gain-bandwidth product |
|  | Amplification: hFE range of 100–300 |
|  | Low saturation voltages: VCE(sat) ≤ 0.3V at 50mA, VBE(sat) ≤ 0.95V |
|  | Fast switching times: 35ns rise and delay, 200ns storage, 50ns fall |
|  | Small form factor: SOT-23 package for compact designs |
|  | Applications: Signal processing, switching, and amplification |
|  |  |
| Raspberry Pi Pico | Dual Cortex-M33 up to 150MHz |
|  | 520KB of SRAM, 4MB flash memory |
|  | USB 1.1 with device and host support |
|  | Low-power sleep and dormant modes |
|  | Drag-and-drop programming using mass storage over USB |
|  | 26× GPIO pins, 3 ADC |
|  | 2× SPI, 2× I2C, 2× UART |
|  | 3× 12-bit 500ksps Analogue to Digital Converter (ADC) |
|  | 24× controllable PWM channels |
|  | 2× Timer with 4 alarms |
|  | 1× AON Timer |
|  | Temperature sensor |
|  |  |
| [MCP6291-E/MS](https://store.comet.bg/Catalogue/Product/16935/) | Gain Bandwidth Product: 10 MHz typical |
|  | Quiescent Current: 1.0 mA typical |
|  | Supply Voltage: 2.4V to 6.0V |
|  | Rail-to-Rail Input/Output: Maximized signal swing capability |
|  | Input Offset Voltage: ±3.0 mV typical |
|  | Common-Mode Input Range: VSS - 0.3V to VDD + 0.3V |
|  | Input Bias Current: ±1.0 pA typical |
|  | Output Voltage Swing: VSS + 15mV to VDD - 15mV |
|  | Slew Rate: 7 V/µs for fast response |
|  | Noise Performance: 8.7 nV/√Hz at 10 kHz |
|  | Operating Temperature: -40°C to +125°C (-40°C to +150°C for H-Temp version) |
|  | Applications: Automotive, portable devices, analog filters, photodiode amplifiers, battery-powered systems |
|  |  |
| [SS14 Schottkey Diods](https://octopart.com/datasheet/ss14-onsemi-29205033?msclkid=543241f99b15108cfff9267f81fafeea&utm_source=bing&utm_medium=cpc&utm_campaign=b_cpc_emea-bg_search_dsa_english_en_usd_all-categories&utm_term=semiconductors&utm_content=Discrete%20Semiconductors%20DSA) | Peak Repetitive Reverse Voltage (VRRM): 40V (SS14) |
|  | Maximum Average Forward Current (IF(AV)): 1.0A typical (at TA = 75°C, 0.375-inch lead length) |
|  | Non-Repetitive Peak Forward Surge Current (IFSM): 40A for 8.3ms single half-sine wave |
|  | Forward Voltage (VF): 850mV at IF = 1.0A |
|  | Reverse Current (IR): 0.2mA at TA = 25°C, 10mA at TA = 100°C |
|  | Power Dissipation (PD): 1.1W typical |
|  | Thermal Resistance, Junction-to-Ambient (RθJA): 88°C/W typical (mounted on FE-4 PCB 0.013mm) |
|  | Operating Junction Temperature (TJ): -65°C to +125°C |
|  | Storage Temperature Range (TSTG): -65°C to +150°C |
|  | Applications: Power supplies, circuit protection, polarity correction, freewheeling diodes, converters |
|  |  |
| [OSTB8BS4C2B LED](https://store.comet.bg/Catalogue/Product/5301166/) | DC Forward Current (IF): 50 mA (Red), 30 mA (Green/Blue) |
|  | Pulse Forward Current (IFP): 120 mA (Red), 100 mA (Green/Blue) (Pulse width: Max 10ms, Duty ratio: Max 1/10) |
|  | Reverse Voltage (VR): 5V |
|  | Power Dissipation (PD): 130 mW (Red), 108 mW (Green/Blue) |
|  | Forward Voltage (VF): |
|  | Red: 1.8V (Min), 2.1V (Typ), 2.6V (Max) at IF = 20mA |
|  | Green/Blue: 2.9V (Min), 3.1V (Typ), 3.6V (Max) at IF = 20mA |
|  | Reverse Current (IR): ≤10 µA at VR = 5V |
|  | Dominant Wavelength (λD): |
|  | Red: 620–630 nm |
|  | Green: 520–530 nm |
|  | Blue: 465–475 nm |
|  | Luminous Intensity (Iv): |
|  | Red: 500–750 mcd at IF = 20mA |
|  | Green: 1120–1560 mcd at IF = 20mA |
|  | Blue: 330–500 mcd at IF = 20mA |
|  | Viewing Angle (2θ1/2): 120° at IF = 20mA |
|  | Operating Temperature (Topr): -30°C to +85°C |
|  | Storage Temperature (Tstg): -40°C to +100°C |
|  | Lead Soldering Temperature (Tsol): 260°C for 5 seconds |
|  | Applications: Toys, games, audio devices, small area illumination, backlighting, general lighting |
|  |  |
| [SMD Resistors/Capacitors 1206](https://www.amazon.com/Component-Assortment-Capacitor-Transistor-Soldering/dp/B07GMRJC1Q) | **Resistors 1600 pcs, each value 20 pcs** |
|  | 1206: Ω: 0 1 3.3 10 22 47 68 100 220 330 470 680 K: 1 2.2 3.3 4.7 6.8 10 22 33 47 68 100 220 330 470 680 M: 1M 4.7 10 |
|  | 0805: Ω: 10 22 47 100 220 470 750 K: 1 2.2 4.7 6.8 10 22 47 75 100 220 470 750 M: 1M |
|  | 0603: Ω: 0 1 10 22 33 47 75 100 220 330 510 750 K: 1 2 2.2 3 3.3 5.1 7.5 10 33 47 51 75 100 220 330 510 750 M: 1M |
|  | **Capacitors 560 pcs, each value 10 pcs** |
|  | 0805: 1p 2.2p 3.3p 4.7p 6.8p 10p 22p 33p 47p 68p 100p 220p 330p 470p 680p 1n 2.2n 3.3n 4.7n 6.8n 10n 22n 33n 47n 68n 100n 220n 330n 470n 680n 1uF 10uF |
|  | 1206: 100p 1n 10n 0.1u 0.22u 0.47u 0.68u 1u 2.2uF 4.7uF 10uF 22uF |
|  | 0603: 100p 1n 10n 0.1u 0.22u 0.47u 0.68u 1u 2.2uF 4.7uF 10uF 22uF |
|  | **Diodes 260 pcs** |
|  | n4001 1n4002 1n4004 1n4007 RL-207 1n5408 FR107 UF4007 SF14 SF18 1n5819 SR1100 SR240 SR2100 1n5822 SR3100 SR540 1n4148 BAV99 BAV70 BAT54C BAT54S |
|  | **Zener Diodes 160pcs** |
|  | 2.7V 3.3V 3.6V 3.9V 4.7V 5.1V 5.6V 6.8V 7.5V 8.2V 9.1V 10V 12V 18V 22V 30V |
|  | **Transistors 180 pcs** |
|  | S9012 S9013 S9014 S9015 S9018 A1015 C1815 A42 A92 2N222 S8050 S8550 5551 5401 A733 C945 2n3904 2n3906 |
|  | **LEDs 75 pcs** |
|  | 1206 0805 0603 each color 5 pcs: red green blue yellow white |
|  | **Trim '3362' Potentiometers 13 pcs** |
|  | 100Ω 200Ω 500Ω 1K 2K 5K 10K 50K 20K 100K 200K 500K 1M |
|  | **OpAmps 9 pcs** |
|  | LM741, LM358, NE5532 (1pcs), LM324, LM386 |
|  | **IC Semiconductors 16 pcs** |
|  | NE555 C4017 DB107S ULN2003 EL817C EL357 78M05 CJ78L05 |
|  | **PCB Adapters** |
|  | SOP8>DIP8, SOT223,SOT-89>TO92, SOT23>TO92 |
|  | **incl. Headers** |
|  | 1 pcs Soldering Tip |
|  | 1 pcs 0.4mm Solder |
|  | 1 pcs Tweezers |
|  | 1 pcs Training PCB |

## Приложение 4

### Експортиране от KiCAD KiCad EDA - Schematic Capture & PCB Design Software

Слоевете на платката трябва да се експортират като гербер файлове (.grb), като освен това трябва да се експортира и файла с отворите (.drl) като това се получава като от file->Fabrication Outputs-> Gerbers, след което от менщто в ляво се избират слоевете, които искаме да направим, задължителни са F.Cu, B.Cu, F.Mask, B.Mask. След това се натиска Plot бутонът, който ще генерира тези файлове в папката, в която се намира платката. Освен това се натиска и Generate Drill Files, който ще генерира 2 файла единият е с NPTH(non platet through hole) и PTH(plated through hole).

### Създаване на .goo файлове с UVtools

Първо от tools->change print resolution трябва да се сложат точните размери и модел на използваният принтер.

След това от Tools-> PCB Exposure трябва да се заредят един copper layer заедно с PTH drill file. След това от поленцата отдолу трябва да се въведе на Layer height – 100, Exposure time – 60s след което се натиска PCB Exposure.

След това от tools->layer arithmetics в поленцето се пише 0-1, което ще извади единият слой от другият и това ще направи така че in the pads медта къедето трябва да има дупки да я няма. След това отменщто в дясната част на екрана бутаме кръгчето таа че да отидем на слоя, който е с отворите и после от actions->removes layer го изтриваме. От Tools->pixel arithmetics се обръщат пикселите, тъй като този принтер конкретно работи в такъв режим, че трябва да се осветят пикселите, които няма да са на пистите.

След това чрез Tools -> flip, Horizontal Flip се неутрализира Mirror ефекта на принтера.

След като платката е позиционирана както трябва от File-> Convert to-> ELEGOO се конвертира файла в .goo, което е необходиият формат. След това тези файлове трябва да се запишат на флашка.

### Изработка

За да може да се направи платката, след като е облъчена с UV-то трябва веднага след това да се махне фоторезиста, след което и да се махне и ненужната мед (да се ецне), за тази цел трябва да се подготвят предварително 2 смеси.

3.1 Течност за махане на фоторезиста

За тази течност е необходимо да се сложи 300мл топла вода от чешмата (около 36 градуса), след което да се сложат 3 грама NaOH (или 1% от количеството вода). Тази течност се разбърква, докато не е мътна.

3.2 Течност за ецване

Тя се състои отново от 300мл вода, на температура от 50 градуса, към която се добавят 60грама от … (10 % от водата). Тази течност отново се разбърква, така че да е напълно прозрачна.

### UVоблъчване

Първо се приготвят течностите с 2-5 градуса в повече от казаното за да може докато се печата платката да не падне много температурата. Взима се платката и се махат сините повърхности които пазят фоторезиста. Включва се тестовият файл и се намества с тиксо и хартия мястото на платката. После от 3d принтера tools->clear\_display се пуска облъчването на цялата задна страна, обръща се и се включва и от другата страна на платката обаче реалният файл. След като се направи това, се поставя в течността и се държи и разбърква 20 секунди. След това се промива с топла вода и се поставя в другата течност. През това се изчиства всичко и от време на време се забърсва платката отгоре. Накрая след като е готова се поставя в топла вода за 3 мин и се облъчва за да падне фоторезиста върху пистите и пак се връща в течността за доизчистване.

## Приложение 5

### Дескриптор на устройства

|  |  |  |  |
| --- | --- | --- | --- |
| **Offset** | **Поле** | **Размер (байта)** | **Описание** |
| 0 | bLength | 1 | Дължина на този дескриптор = 18 байта |
| 1 | bDescriptorType | 1 | Тип на дескриптора = DEVICE (01h) |
| 2 | bcdUSB | 2 | Версия на USB спецификацията (BCD) |
| 4 | bDeviceClass | 1 | Клас на устройството |
| 5 | bDeviceSubClass | 1 | Подклас на устройството |
| 6 | bDeviceProtocol | 1 | Протокол на устройството |
| 7 | bMaxPacketSize0 | 1 | Максимален размер на пакет за Endpoint 0 |
| 8 | idVendor | 2 | Vendor ID (определен от USB-IF) |
| 10 | idProduct | 2 | Product ID (определен от производителя) |
| 12 | bcdDevice | 2 | Номер на версията на устройството (BCD) |
| 14 | iManufacturer | 1 | Индекс на низовия дескриптор за производителя |
| 15 | iProduct | 1 | Индекс на низовия дескриптор за продукта |
| 16 | iSerialNumber | 1 | Индекс на низовия дескриптор за сериен номер |
| 17 | bNumConfigurations | 1 | Брой поддържани конфигурации |

Таблица ‑ Дескриптор на устройства

### Конфигурационен дескриптор (Configuration Descriptor)

|  |  |  |  |
| --- | --- | --- | --- |
| **Offset** | **Поле** | **Размер (байта)** | **Описание** |
| 0 | bLength | 1 | Дължина на този дескриптор = 9 байта |
| 1 | bDescriptorType | 1 | Тип на дескриптора = CONFIGURATION (02h) |
| 2 | wTotalLength | 2 | Обща дължина на всички дескриптори в конфигурацията |
| 4 | bNumInterfaces | 1 | Брой интерфейси в тази конфигурация |
| 5 | bConfigurationValue | 1 | Стойност за избор на конфигурацията |
| 6 | iConfiguration | 1 | Индекс на низовия дескриптор за конфигурацията |
| 7 | bmAttributes | 1 | Атрибути на устройството (например дали е самозахранвано) |
| 8 | bMaxPower | 1 | Максимална мощност за тази конфигурация (в единици от 2 mA) |

Таблица ‑ Конфигурационен дескриптор

### Дескриптор на интерфейс (Interface Descriptor)

|  |  |  |  |
| --- | --- | --- | --- |
| **Offset** | **Поле** | **Размер (байта)** | **Описание** |
| 0 | bLength | 1 | Дължина на този дескриптор = 9 байта |
| 1 | bDescriptorType | 1 | Тип на дескриптора = INTERFACE (04h) |
| 2 | bInterfaceNumber | 1 | Номер на интерфейса |
| 3 | bAlternateSetting | 1 | Алтернативна настройка |
| 4 | bNumEndpoints | 1 | Брой на endpoint-овете за този интерфейс |
| 5 | bInterfaceClass | 1 | Клас на интерфейса |
| 6 | bInterfaceSubClass | 1 | Подклас на интерфейса |
| 7 | bInterfaceProtocol | 1 | Протокол на интерфейса |

Таблица ‑ Дескриптор на интерфейс

### Дескриптор на endpoint (Endpoint Descriptor)

|  |  |  |  |
| --- | --- | --- | --- |
| **Offset** | **Поле** | **Размер (байта)** | **Описание** |
| 0 | bLength | 1 | Дължина на този дескриптор = 7 байта |
| 1 | bDescriptorType | 1 | Тип на дескриптора = ENDPOINT (05h) |
| 2 | bEndpointAddress | 1 | Номер и посока на endpoint |
| 3 | bmAttributes | 1 | Тип на трансфера (контролен, изохронен, bulk, или прекъсване) |
| 4 | wMaxPacketSize | 2 | Максимален размер на пакета |
| 6 | bInterval | 1 | Интервал за polling (в милисекунди) |

Таблица ‑ Дескриптор на endpoint

## Приложение 6

Computer Security Resource Center by the National Institute of Standards and Technology- <https://csrc.nist.gov/projects/random-bit-generation/documentation-and-software/guide-to-the-statistical-tests>

A remark on the NIST 800-22 Binary Matrix Rank Test by Nicu Neculache- <https://eprint.iacr.org/2022/061.pdf>

Statistical Testing of Random Number Generators by Juan Soto- <https://csrc.nist.rip/nissc/1999/proceeding/papers/p24.pdf>

Random number generator testing by John D. Cook - <https://www.johndcook.com/blog/rng-testing/>

The actual tests by terrillmoore - <https://github.com/terrillmoore/NIST-Statistical-Test-Suite/blob/master/sts/src/utilities.c>

## Приложение 7:

### usb\_device\_configuration

// Struct defining the device configuration

static struct usb\_device\_configuration dev\_config = {

        .device\_descriptor = &device\_descriptor,

        .interface\_descriptor = &interface\_descriptor,

        .config\_descriptor = &config\_descriptor,

        .lang\_descriptor = lang\_descriptor,

        .descriptor\_strings = descriptor\_strings,

        .endpoints = {

                {

                        .descriptor = &ep0\_out,

                        .handler = &ep0\_out\_handler,

                        .endpoint\_control = NULL, // NA for EP0

                        .buffer\_control = &usb\_dpram->ep\_buf\_ctrl[0].out,

                        // EP0 in and out share a data buffer

                        .data\_buffer = &usb\_dpram->ep0\_buf\_a[0],

                },

                {

                        .descriptor = &ep0\_in,

                        .handler = &ep0\_in\_handler,

                        .endpoint\_control = NULL, // NA for EP0,

                        .buffer\_control = &usb\_dpram->ep\_buf\_ctrl[0].in,

                        // EP0 in and out share a data buffer

                        .data\_buffer = &usb\_dpram->ep0\_buf\_a[0],

                },

                {

                        .descriptor = &ep1\_out,

                        .handler = &ep1\_out\_handler,

                        // EP1 starts at offset 0 for endpoint control

                        .endpoint\_control = &usb\_dpram->ep\_ctrl[0].out,

                        .buffer\_control = &usb\_dpram->ep\_buf\_ctrl[1].out,

                        // First free EPX buffer

                        .data\_buffer = &usb\_dpram->epx\_data[0 \* 64],

                },

                {

                        .descriptor = &ep2\_in,

                        .handler = &ep2\_in\_handler,

                        .endpoint\_control = &usb\_dpram->ep\_ctrl[1].in,

                        .buffer\_control = &usb\_dpram->ep\_buf\_ctrl[2].in,

                        // Second free EPX buffer

                        .data\_buffer = &usb\_dpram->epx\_data[1 \* 64],

                }

        }

};

### usb\_handle\_device\_descriptor, usb\_handle\_config\_descriptor и usb\_handle\_setup\_packet

/\*\*

 \* @brief Respond to a setup packet from the host.

 \*

 \*/

void usb\_handle\_setup\_packet(void) {

    volatile struct usb\_setup\_packet \*pkt = (volatile struct usb\_setup\_packet \*) &usb\_dpram->setup\_packet;

    uint8\_t req\_direction = pkt->bmRequestType;

    uint8\_t req = pkt->bRequest;

    // Reset PID to 1 for EP0 IN

    usb\_get\_endpoint\_configuration(EP0\_IN\_ADDR)->next\_pid = 1u;

    if (req\_direction == USB\_DIR\_OUT) {

        if (req == USB\_REQUEST\_SET\_ADDRESS) {

            usb\_set\_device\_address(pkt);

        } else if (req == USB\_REQUEST\_SET\_CONFIGURATION) {

            usb\_set\_device\_configuration(pkt);

        } else {

            usb\_acknowledge\_out\_request();

            //f("Other OUT request (0x%x)\r\n", pkt->bRequest);

        }

    } else if (req\_direction == USB\_DIR\_IN) {

        if (req == USB\_REQUEST\_GET\_DESCRIPTOR) {

            uint16\_t descriptor\_type = pkt->wValue >> 8;

            switch (descriptor\_type) {

                case USB\_DT\_DEVICE:

                    usb\_handle\_device\_descriptor(pkt);

                    //f("GET DEVICE DESCRIPTOR\r\n");

                    break;

                case USB\_DT\_CONFIG:

                    usb\_handle\_config\_descriptor(pkt);

                    //f("GET CONFIG DESCRIPTOR\r\n");

                    break;

                case USB\_DT\_STRING:

                    usb\_handle\_string\_descriptor(pkt);

                    //f("GET STRING DESCRIPTOR\r\n");

                    break;

                default:

                ;

                    //f("Unhandled GET\_DESCRIPTOR type 0x%x\r\n", descriptor\_type);

            }

        } else {

            //f("Other IN request (0x%x)\r\n", pkt->bRequest);

        }

    }

}

/\*\*

 \* @brief Send the requested string descriptor to the host.

 \*

 \* @param pkt, the setup packet from the host.

 \*/

void usb\_handle\_string\_descriptor(volatile struct usb\_setup\_packet \*pkt) {

    uint8\_t i = pkt->wValue & 0xff;

    uint8\_t len = 0;

    if (i == 0) {

        len = 4;

        memcpy(&ep0\_buf[0], dev\_config.lang\_descriptor, len);

    } else {

        // Prepare fills in ep0\_buf

        len = usb\_prepare\_string\_descriptor(dev\_config.descriptor\_strings[i - 1]);

    }

    usb\_start\_transfer(usb\_get\_endpoint\_configuration(EP0\_IN\_ADDR), &ep0\_buf[0], MIN(len, pkt->wLength));

}

/\*\*

 \* @brief Send device descriptor to host

 \*

 \*/

void usb\_handle\_device\_descriptor(volatile struct usb\_setup\_packet \*pkt) {

    const struct usb\_device\_descriptor \*d = dev\_config.device\_descriptor;

    // EP0 in

    struct usb\_endpoint\_configuration \*ep = usb\_get\_endpoint\_configuration(EP0\_IN\_ADDR);

    // Always respond with pid 1

    ep->next\_pid = 1;

    usb\_start\_transfer(ep, (uint8\_t \*) d, MIN(sizeof(struct usb\_device\_descriptor), pkt->wLength));

}

/\*\*

 \* @brief Send the configuration descriptor (and potentially the configuration and endpoint descriptors) to the host.

 \*

 \* @param pkt, the setup packet received from the host.

 \*/

void usb\_handle\_config\_descriptor(volatile struct usb\_setup\_packet \*pkt) {

    uint8\_t \*buf = &ep0\_buf[0];

    // First request will want just the config descriptor

    const struct usb\_configuration\_descriptor \*d = dev\_config.config\_descriptor;

    memcpy((void \*) buf, d, sizeof(struct usb\_configuration\_descriptor));

    buf += sizeof(struct usb\_configuration\_descriptor);

    // If we more than just the config descriptor copy it all

    if (pkt->wLength >= d->wTotalLength) {

        memcpy((void \*) buf, dev\_config.interface\_descriptor, sizeof(struct usb\_interface\_descriptor));

        buf += sizeof(struct usb\_interface\_descriptor);

        const struct usb\_endpoint\_configuration \*ep = dev\_config.endpoints;

        // Copy all the endpoint descriptors starting from EP1

        for (uint i = 2; i < USB\_NUM\_ENDPOINTS; i++) {

            if (ep[i].descriptor) {

                memcpy((void \*) buf, ep[i].descriptor, sizeof(struct usb\_endpoint\_descriptor));

                buf += sizeof(struct usb\_endpoint\_descriptor);

            }

        }

    }

    // Send data

    // Get len by working out end of buffer subtract start of buffer

    uint32\_t len = (uint32\_t) buf - (uint32\_t) &ep0\_buf[0];

    usb\_start\_transfer(usb\_get\_endpoint\_configuration(EP0\_IN\_ADDR), &ep0\_buf[0], MIN(len, pkt->wLength));

}

### usb\_setup\_endpoints

/\*\*

 \* @brief Set up the endpoint control register for an endpoint (if applicable. Not valid for EP0).

 \*

 \* @param ep

 \*/

void usb\_setup\_endpoint(const struct usb\_endpoint\_configuration \*ep) {

    //f("Set up endpoint 0x%x with buffer address 0x%p\n", ep->descriptor->bEndpointAddress, ep->data\_buffer);

    // EP0 doesn't have one so return if that is the case

    if (!ep->endpoint\_control) {

        return;

    }

    // Get the data buffer as an offset of the USB controller's DPRAM

    uint32\_t dpram\_offset = usb\_buffer\_offset(ep->data\_buffer);

    uint32\_t reg = EP\_CTRL\_ENABLE\_BITS

                   | EP\_CTRL\_INTERRUPT\_PER\_BUFFER

                   | (ep->descriptor->bmAttributes << EP\_CTRL\_BUFFER\_TYPE\_LSB)

                   | dpram\_offset;

    \*ep->endpoint\_control = reg;

}

### usb\_device\_init

/\*\*

 \* @brief Set up the USB controller in device mode, clearing any previous state.

 \*

 \*/

void usb\_device\_init() {

    // Reset usb controller

    reset\_unreset\_block\_num\_wait\_blocking(RESET\_USBCTRL);

    // Clear any previous state in dpram just in case

    memset(usb\_dpram, 0, sizeof(\*usb\_dpram)); // <1>

    // Enable USB interrupt at processor

    irq\_set\_enabled(USBCTRL\_IRQ, true);

    // Mux the controller to the onboard usb phy

    usb\_hw->muxing = USB\_USB\_MUXING\_TO\_PHY\_BITS | USB\_USB\_MUXING\_SOFTCON\_BITS;

    // Force VBUS detect so the device thinks it is plugged into a host

    usb\_hw->pwr = USB\_USB\_PWR\_VBUS\_DETECT\_BITS | USB\_USB\_PWR\_VBUS\_DETECT\_OVERRIDE\_EN\_BITS;

    // Enable the USB controller in device mode.

    usb\_hw->main\_ctrl = USB\_MAIN\_CTRL\_CONTROLLER\_EN\_BITS;

    // Enable an interrupt per EP0 transaction

    usb\_hw->sie\_ctrl = USB\_SIE\_CTRL\_EP0\_INT\_1BUF\_BITS; // <2>

    // Enable interrupts for when a buffer is done, when the bus is reset,

    // and when a setup packet is received

    usb\_hw->inte = USB\_INTS\_BUFF\_STATUS\_BITS |

                   USB\_INTS\_BUS\_RESET\_BITS |

                   USB\_INTS\_SETUP\_REQ\_BITS;

    // Set up endpoints (endpoint control registers)

    // described by device configuration

    usb\_setup\_endpoints();

    // Present full speed device by enabling pull up on DP

    usb\_hw\_set->sie\_ctrl = USB\_SIE\_CTRL\_PULLUP\_EN\_BITS;

}