SSD Doujinshi

SSDの同人誌

TAKE FREE ¥0

データセンターやノートPCなどで
活躍する「はたらくSSD」をご紹介！
製作記事や技術解説がてんこ盛り！

Contents

宍戸（ししど）ちゃん

SSD に詳しい謎の小学生。
キオクシアの SSD が搭載された
ノート PC は匂いで分かるらしい。
好きな生き物は珪藻と放散虫。

Maker Faire Tokyo 2021 出展にあたり

ragnag

こんにちは。キオクシアです。このたび、7回目のMaker Faire Tokyo出展となりました。ん？　キオクシアなんて会社これまでに出展していたっけ？と思われた方、いらっしゃるかもしれません。私たちキオクシアは、2019年10月に「東芝メモリ株式会社」から、社名を変更し、「キオクシア株式会社」となりました。2014年から2019年までの6年間は東芝および東芝メモリとして、そして今年はキオクシアとして初出展します。

今年のテーマは、「はたらくSSD ～ SSDの役割ってなんだろう～」としました。

キオクシアは、1987年にNAND型フラッシュメモリを発明したメモリのメーカーです。このフラッシュメモリを使って、ソリッド・ステート・ドライブ（SSD）を開発・製造しています。SSDと言えば、外付けや自作パソコンで使ったりする、あれ？　はい、そうです。今回のMaker Faire Tokyoでは、SSDの中でも特に「はたらくSSD」、ビジネス向け（B2B向け）SSDにフォーカスしました。

「ビジネス向け？」なんだか難しそう…って思われたかもしれません。いえいえ、日ごろビジネス向けSSDを目にすることはあまりないかもしれませんが、最近では、ノートパソコンの記憶装置にハードディスクの置き換えとして搭載されたり、ネットワーク経由でメールや写真データの保管に使われるクラウドサーバーにたくさん使われており、実は世界中の皆さんが日常的に利用されているんですよ。見えないところで皆さんの「データ」を保存し、皆さんの「記憶」を支えているのはキオクシアのSSDかも…！

今回のMaker Faire Tokyoでは、普段皆さんが目にすることのないSSD製品たち、その技術、その活用事例を、皆さんとお会いして製品実物・デモなどでわかりやすくご説明できるよう準備を進めてきました。しかしながらコロナの影響で、東京ビックサイトでのオンサイト開催が中止となり、それがかなわなくなってしまいました。皆さんとお会いできないのはとても残念ですが、YouTube動画とこのSSD同人誌で、はたらくSSDについてじっくりご紹介します。

ようこそ、「はたらくSSD」の世界へ。

ragnag（@ragnag1109）

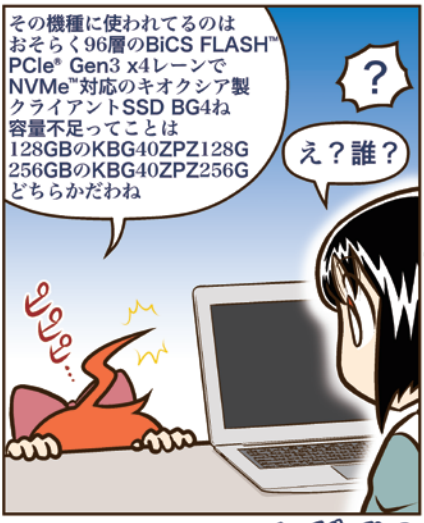
まんがを読むのと描くのをこよなく愛する広報担当。キオクシアとキオクシアSSDのプレゼンスアップを模索する中で、データセンターTENCOへ足を踏み入れた！・・・w。

いろいろなフォームファクターが愛らしいキオクシアSSDたちをよろしくお願いします～。


創意工夫で
お客さまのニーズに
何でも応えるお店
ホームセンター
なんでもそろう店!
TENCO
ホームセンター
TENCO
…なんだけど


ううう…
黒鉄先生に
頼まれたこれは…
まっぴー
ホームセンターTENCO店員
井本典子(てんこ)


データセンター
TENCO
てんこ
ピ
ノーパソの容量
増やすなんて
無理〜〜〜〜!!
機械オンチ

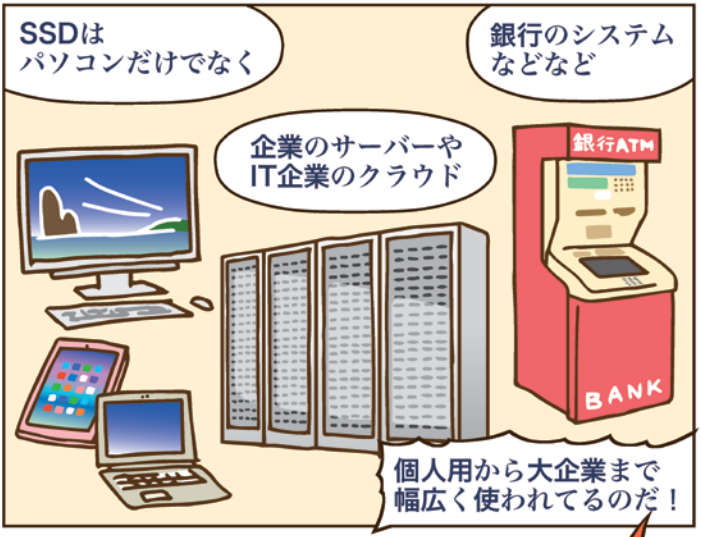

その機種に使われてるのは
おそらく96層のBiCS FLASH™
PCIe® Gen3 x4レーンで
NVMe™対応のキオクシア製
クライアントSSD BG4ね
容量不足ってことは
128GBのKBG40ZPZ128G
256GBのKBG40ZPZ256G
どちらかだわね
ピピピ…
?
え？誰？


キオクシア
激推し♥
SSDの申し子
宍戸(ししど)ちゃん
で——す!!
SSD(ししど)ちゃん

作 とだ勝之

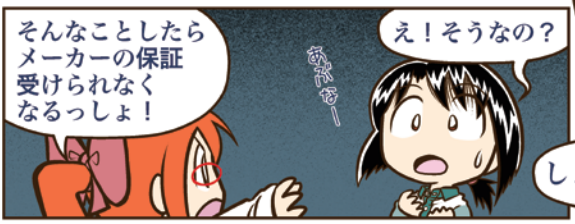

SSD？
記憶しぃや？
？
？
おねぇちゃん
何も知らんのな


SSDは
Sソリッド
Sステート
Dドライブ
フラッシュメモリを
使った記憶媒体！
ど
ーん
キオクシアは
日本に拠点を置く
SSDメーカーだよ！


SSDは
パソコンだけでなく
銀行のシステム
などなど
企業のサーバーや
IT企業のクラウド
銀行ATM
BANK
個人用から大企業まで
幅広く使われてるのだ！


へ――っ！
すごい！
個人で買える
のもあるのね？
もちろん！


よーし！
んじゃさっそく
分解して
取り出してみよ
ダメ――!!


そんなことしたら
メーカーの保証
受けられなく
なるっしょ！
え！そうなの？
あぶな～


宍戸ちゃんが
SSDの基礎から
教えてあげるよ！
しょーがない！
つづきはWEBで？

キオクシアの SSD たち

ragnag

世界で生み出される情報量は爆発的に増えていて、その情報を記録するストレージ装置が必要になっている。パソコンやタブレットにたくさんの写真、音楽、ビデオなどのデータが保存できるようになったり、クラウドを利用してデータを保存して、どこからでも使えるスタイルが定着してきた。

ビジネスユース

エンタープライズ SSD

エンタープライズ SSD は、企業の業務のためのデータ処理や従来型のサーバーやストレージ向けに設計され、ミッションクリティカル[1]なシステムに適した高いパフォーマンスと高信頼性、セキュリティ機能が特徴だ。フルスピードで継続的に読み書きを必要とする、24 時間年中無休で動作するワークステーション、サーバー、ストレージデバイス向けに設計されている。

データセンター SSD

データセンターSSDは、サーバーやネットワーク機器などの IT 機器を収容する施設で、システムやデータを安心、安全なロケーションで安定した運用をしたいというニーズに応えている。低消費電力と高性能を両立していることが特徴で、大規模なクラウドデータセンターに適している。

エンタープライズストレージ

データセンターサーバー

1　止まってしまうと業務の遂行に致命的な影響が出てしまうシステム。

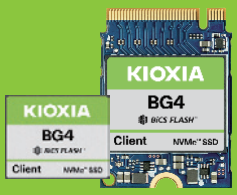
SSD は、Solid State Drive（ソリッド・ステート・ドライブ）の略で NAND フラッシュメモリ、コントローラ、ファームウェアなどで構成されるストレージ製品のひとつだ。
SSD は幅広い分野で使われ、人々の生活になくてはならないものになってきた。

キオクシアの SSD には、ビジネス用途のエンタープライズ SSD、データセンター SSD、クライアント SSD、パーソナル用途のコンシューマ SSD がある。

パーソナルユース

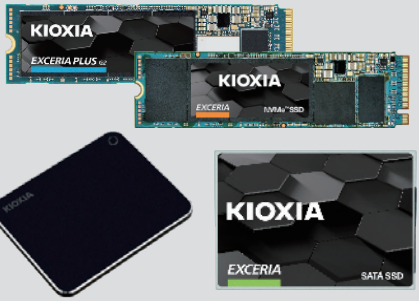
クライアント SSD

クライアント SSD は、モバイルノート PC やタブレットなどのコンピュータ向けで、小型・軽量・低消費電力が特徴だ。
PC からエントリーレベルのサーバーを中心に、幅広い分野で使用可能な設計になっている。



コンシューマ SSD

コンシューマ SSD は、高性能ゲーミングデスクトップ PC の自作からノート PC の手軽なアップグレードまで、幅広いニーズに対応している。
「EXCERIA(エクセリア)」というブランド名で展開していて、インターネットサイトなどで購入できる。



タブレット

パソコン

ゲーミング

ポータブル
ストレージ

2 年ぶりの Maker Faire Tokyo と SSD 同人誌の製作

Pochio

久々に同人誌製作の夏がやってきましたが、オライリーさんから今年の Maker Faire Tokyo のオンサイト開催中止が発表されました。2年ぶりの出展でとても楽しみにしていましたが、この状況ではやむをえませんし、オライリーさんにとってかなり苦渋の決断だったのではないかと察するところです。来年こそは、Maker の皆さんのたくさんの作品を拝見できることを願っています。早く落ち着いてほしいですね。

さて、Maker Faire Tokyo の会場といえば東京ビッグサイトですが、この夏はオリンピックのプレスセンターとして使われていました。たまたま近くに寄る機会があったのですが、夏には人だらけのビッグサイト付近が、いつになく閑散としていつもと違う夏を演出していたかのようでした。

最後だと思っていた Maker Faire Tokyo 2019 出展

さて、Maker Faire Tokyo には 2014 年から出展させていただきましたが、6回目の参加となる 2019 年をもって、諸般の事情で一区切りを迎えました。2018 年までは主に無線 LAN 機能を搭載した SD カード、「FlashAir™」をメインに展示をしてきました。2019 年は趣向を変えまして、本物のシリコンウェハの展示（図 1）やまんが「フラッシュメモリのひみつ」の紹介など、フラッシュメモリの製造や動作の仕組みに焦点をあてた展示を展開しました。直径 30cm のシリコンウェハを模したうちわ（図 2）を製作したところ、Twitter でぷちバズってしまい、本物のシリコンウェハでできていると誤解された方がいらっしゃいました。ちなみに本物なら扇いだ途端に割れてしまい、うちわの機能を果たさないでしょうね（汗）。

図 1: 実物のシリコンウェハの展示

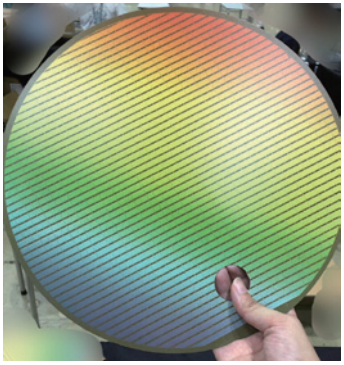


図 2: シリコンウェハうちわ

この 2019 年の出展が事実上最後になるとわかっていたのですが、同人誌を毎年楽しみにされている方からは、「やめないでほしい」とか、「有料でもいいので続けてほしい」といった大変ありがたい言葉をいただきました。こちらが想定している以上に楽しみにされている方の存在を知って、とても励みになったのでした。その後、会社が独立したり社名が変わったりと様々な環境の変化がありました。それまで出展に関わってきた関係者の所属や、仕事も役割も徐々に変わりつつあったので、再び集まって Maker Faire Tokyo に出展する機会はもう来ないだろう、と思っていたのでした。

突然やってきた復帰のチャンス

ところが様々な偶然が重なって、出展のチャンスが突然巡ってきました。今年の展示のテーマは SSD（Solid State Drive）です。いまやハードディスクドライブとの入れ替わりが進みつつある、フラッシュメモリを用いた補助記憶装置です。SSD には家電量販店や自作 PC パーツショップで買えるような一般消費者向け（いわゆる B2C 向け）の SSD と、企業や法人向け（いわゆる B2B 向け）の SSD があり、今回は後者、つまりサーバーやクラウド向けの SSD が展示のメインとなりました。個人で入手する機会がほとんどない SSD ですから、ちょっとマニアックですね。

そんな B2B 向け SSD は、なんだか遠い存在のように思えます。「さすがに見たことがない」という方がほとんどでしょう。ところが実は、身近な存在だったりするのです。例えばみなさんが SNS でつぶやいてアップロードした写真は、どこにあるのでしょうか。闇に葬られたりしなければ、きっと B2B 向けの SSD やハードディスクに格納されて、スマートフォンなどの端末に表示されたりするのです。となれば現代のコミュニケーションツールとして、B2B 向けの SSD というのは重要な役割を担い、かつ必要不可欠な存在といえます。

さらに B2B 向けの SSD はそういったサーバー向けのものだけではありません。メーカーが販売するパソコンには、当社製のクライアント SSD とよばれるものを搭載しているものがあります。実は先日、某社製のゲーミングノート PC を購入したのですが、キオクシアの 512GB のクライアント SSD が搭載されていました。もちろん狙って買ったわけではありませんから、ちょっと驚きました。この SSD は B2B 製品なので、量販店やインターネットで個人が買えるものではありません。

そういえば、最近某大手ネット通販サイトにキオクシア製の B2C 向け 2.5 インチ SSD が販売されていて、2 台購入して古い PC やゲーム機のハードディスクと交換してみました。すると OS の起動がとても速くなったり、それまで動作が重たかったブラウザも軽くなったりと、SSD の恩恵を十分に感じることができました。SSD のおかげで、個人の IT 環境もだいぶ快適になっている感がありますね。

SSD 同人誌の製作

図 3: 人が少なめの秋葉原

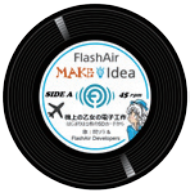
というわけで、Maker Faire Tokyo に 2 年ぶりに出展できることになりましたので、久しぶりに関係者が集まって議論をし、同人誌製作を開始しました。もちろん同人誌のテーマは B2B 向け SSD ですが、SSD について幅広く知っていただくために、その機能や仕組みの紹介、SSD の活用事例などを載せることにしました。ところがこれでは Maker Faire っぽさが足りません。

そこで、「公開情報だけで作る自作 SSD」というネタが浮上しました。マイコンの力を借りますが、基板にフラッシュメモリのチップを載せて、SSD を制御するプログラミングをするのです。フラッシュメモリチップへのアクセス方法から、実際にパソコンからストレージとして認識しているかを確認するところまで、この同人誌に解説記事が掲載されています。今後、こう言った自作 SSD の流れが徐々に盛り上がっていったりしませんかね・・・(願望)。

ところで先日ですが、所用のついでに秋葉原に行ってきました。このご時世なので、路地裏を歩く人がだいぶ少なかったです(図 3)。いくつか自作 PC パーツを扱う量販店に立ち寄ったのですが、なんとキオクシア製の SSD はどこも取り扱いがありませんでした・・・。

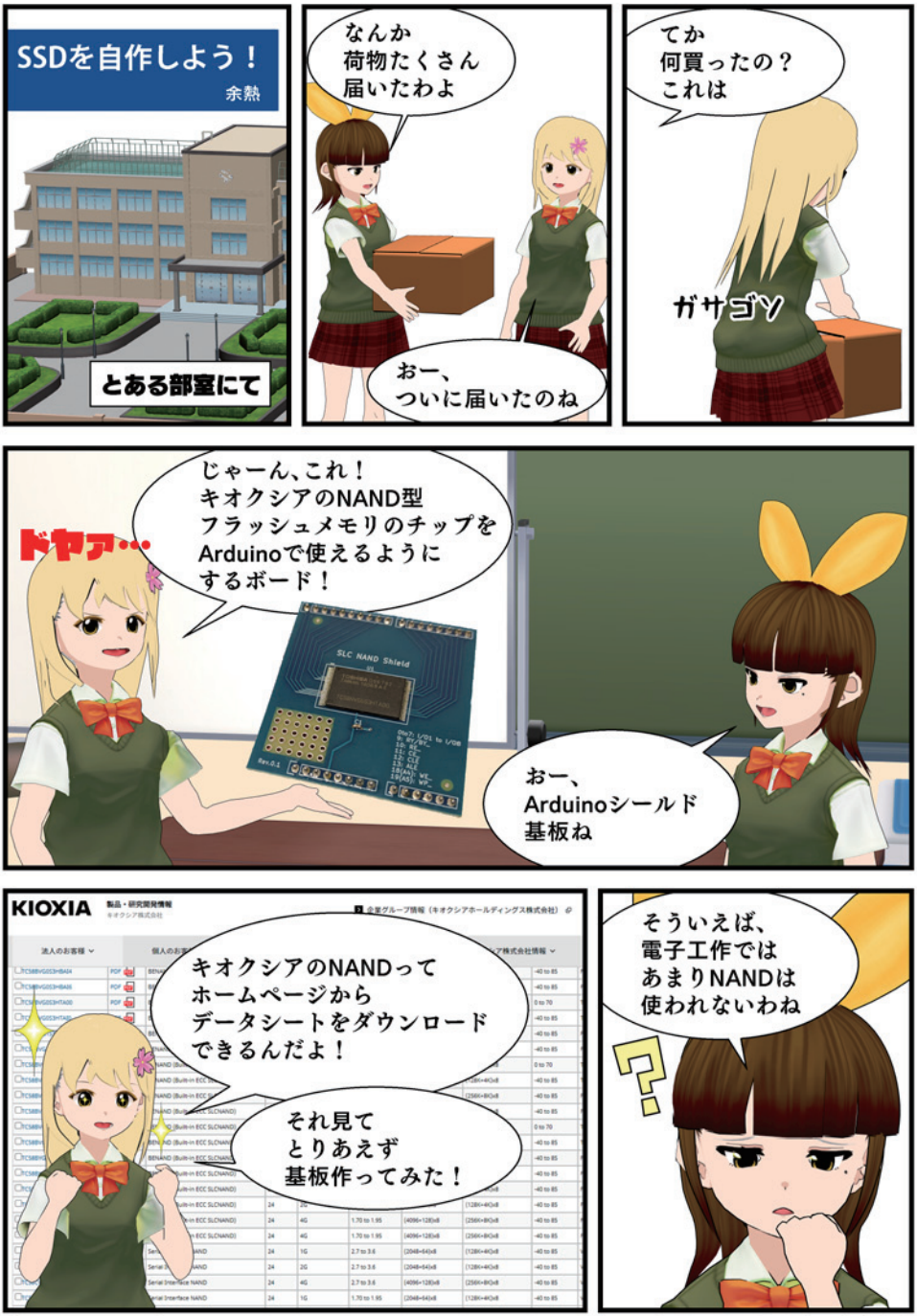
あとでわかったのですけど、実はキオクシアの B2C 向け SSD 製品は、現時点ではインターネット通販サイトでないと購入できないのだそうです。2.5 インチ SSD をネット通販で購入したことを先に述べましたが、正しい購入方法だったのですね。というわけで、キオクシアの SSD に興味のある方は、ぜひネット通販サイトでお探しください。

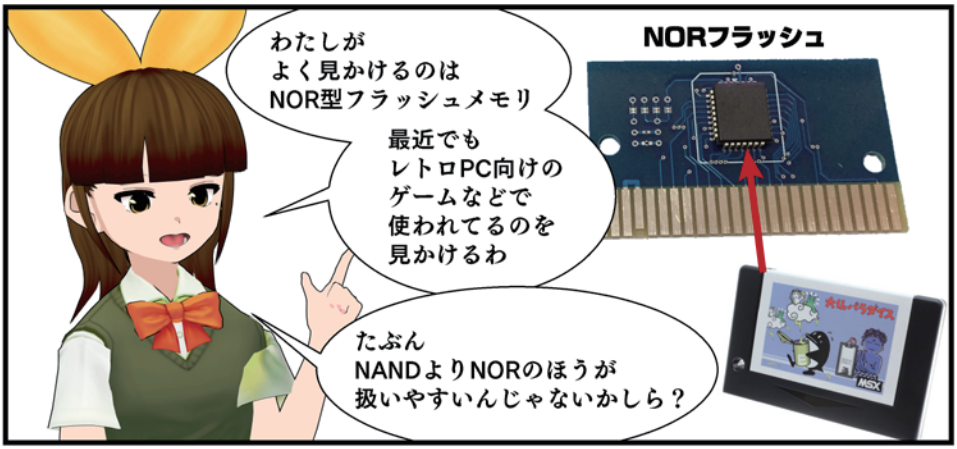
そんなわけで、この SSD 同人誌のご感想をお聞かせいただけると幸いです。ハッシュタグ「#SSD 同人誌」でツイートしていただければ、執筆陣に感想がフィードバックされるでしょう。そしてもしかしますと、SSD 同人誌 2 号の製作につながるかもしれません。その時にはぜひ自作 SSD 製作のツワモノが表れて記事を書いてくださると、とてもありがたいです。



Pochio (@I_love_nintendo)

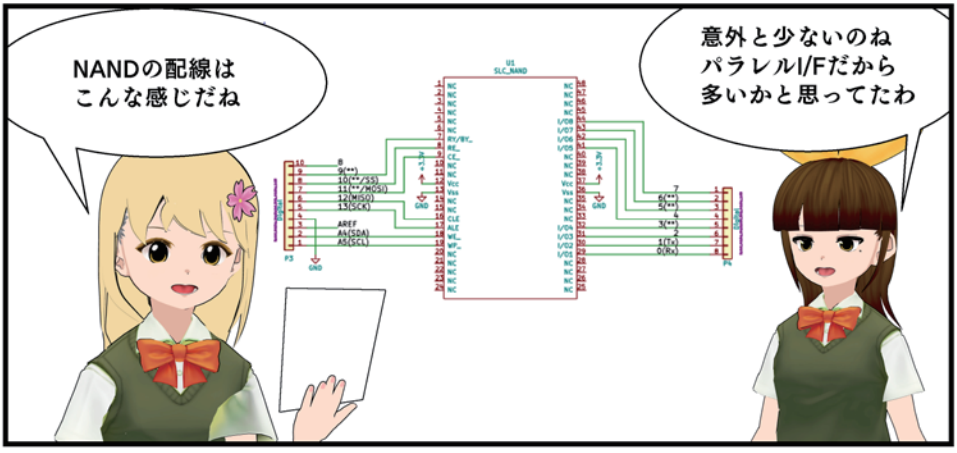
元自称 FlashAir™ 芸人です。この同人誌のシリーズは、みなさんの製作記事を Maker Faire Tokyo の参加者にお披露目する紙上展示的な役割があるのかも、と思いました。自宅に持ち帰ってゆっくり読めますからね。

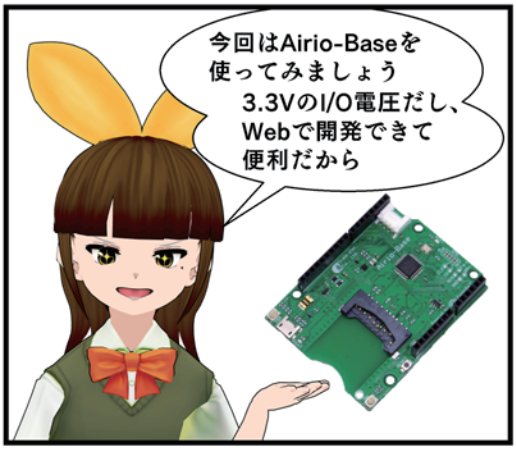

SSDを自作しよう！
余熱
とある部室にて
なんか
荷物たくさん
届いたわよ
おー、
ついに届いたのね
てか
何買ったの？
これは
ガサゴソ
じゃーん、これ！
キオクシアのNAND型
フラッシュメモリのチップを
Arduinoで使えるように
するボード！
ドヤア…
SLC NAND Shield
おー、
Arduinoシールド
基板ね
KIOXIA
キオクシアのNANDって
ホームページから
データシートをダウンロード
できるんだよ！
それ見て
とりあえず
基板作ってみた！
そういえば、
電子工作では
あまりNANDは
使われないわね
?


わたしが
よく見かけるのは
NOR型フラッシュメモリ
最近でも
レトロPC向けの
ゲームなどで
使われてるのを
見かけるわ
NORフラッシュ
たぶん
NANDよりNORのほうが
扱いやすいんじゃないかしら？


NANDは制御が難しいけど、
とりあえず動かすだけなら
割と簡単なはず！
（根拠なし）
まずはやってみよう！

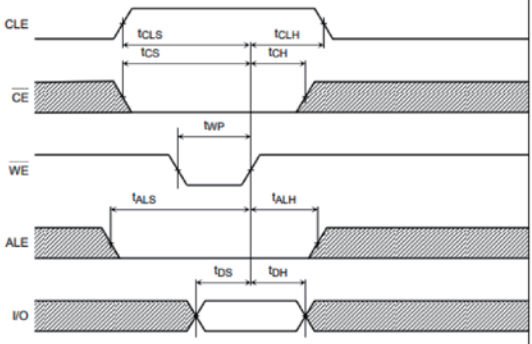

最初は
データシートの
確認だね


NANDの配線は
こんな感じだね
意外と少ないのね
パラレルI/Fだから
多いかと思ってたわ

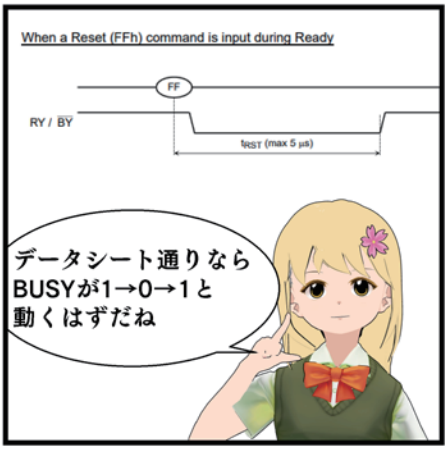


RESETの動作

Command Input Cycle Timing Diagram




```
// RESET
CommandLatchEnable = 1;
ChipEnable = 0;

WriteEnable = 0;

IO1 = 1;
IO2 = 1;
IO3 = 1;
IO4 = 1;
IO5 = 1;
IO6 = 1;
IO7 = 1;
IO8 = 1;

WriteEnable = 1;

CommandLatchEnable = 0;
ChipEnable = 1;

while(!rb) {
rb = Busy.read();
serial.printf("Busy:%x\n",rb);
myled = 1;
}
```



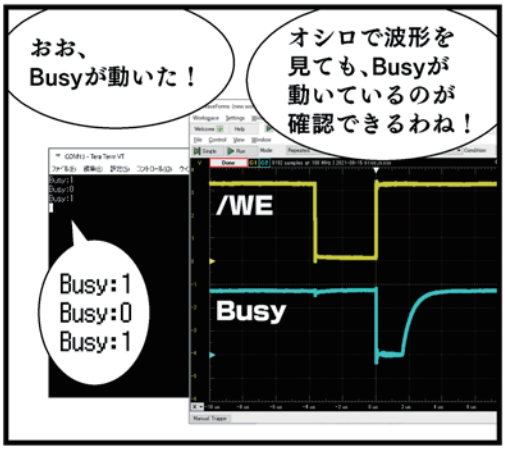
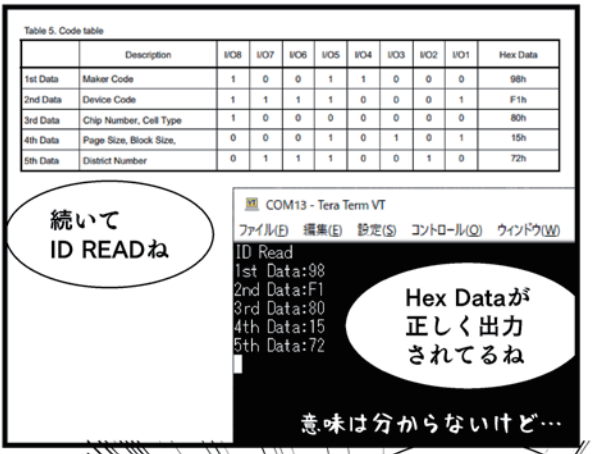
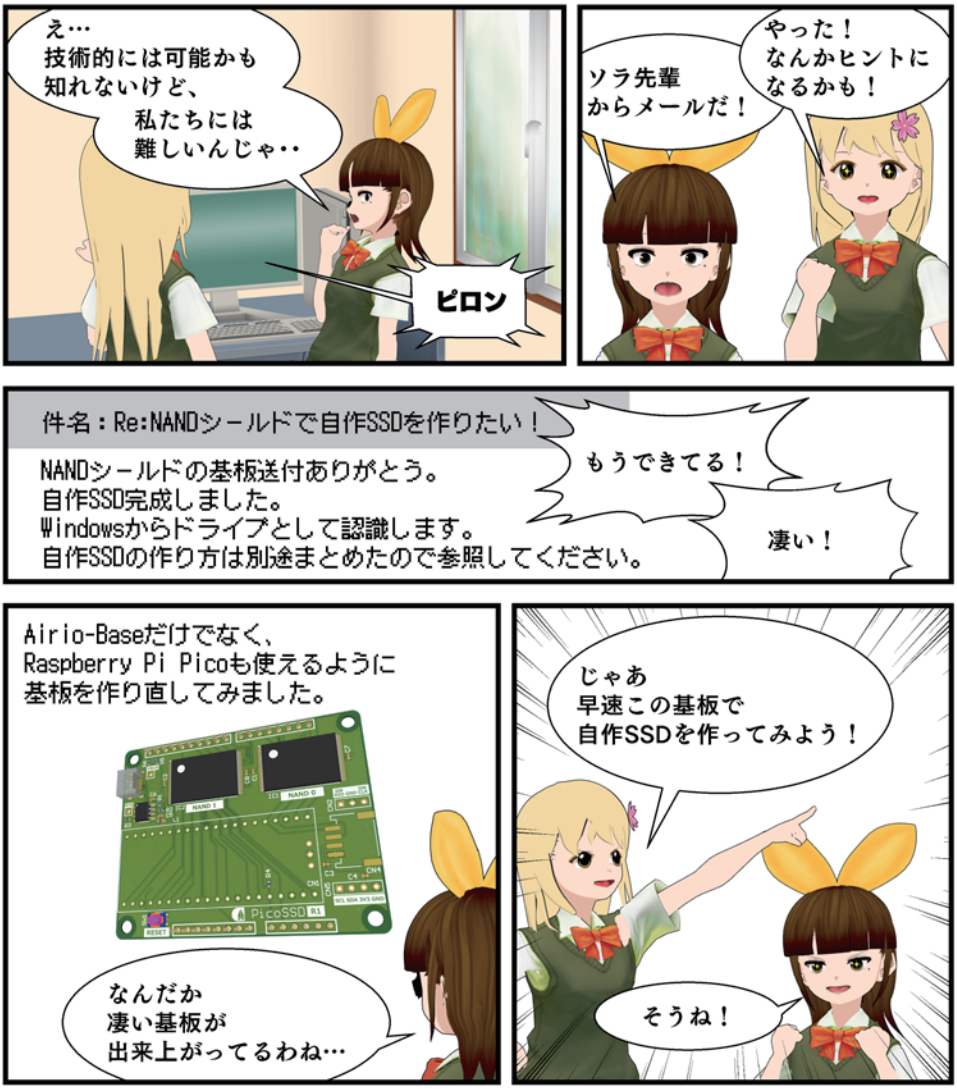


Table 5. Code table

	Description	I/O8	I/O7	I/O6	I/O5	I/O4	I/O3	I/O2	I/O1	Hex Data
1st Data	Maker Code	1	0	0	1	1	0	0	0	98h
2nd Data	Device Code	1	1	1	1	0	0	0	1	F1h
3rd Data	Chip Number, Cell Type	1	0	0	0	0	0	0	0	80h
4th Data	Page Size, Block Size,	0	0	0	1	0	1	0	1	15h
5th Data	District Number	0	1	1	1	0	0	1	0	72h





余熱（@yone2_net）

SSD を自作したい！というネタは少し前から温めていたのですが、今回、色々な方のご協力によって実現できました。また、本原稿の執筆にあたり GrabTF のグラブさんなどにもお世話になりました。

自作 SSD 基板「PicoSSD」

（株）クレイン電子 福屋 新吾

電子工作ではあまり使われることのない、（素の）NAND 型フラッシュメモリのチップ（Raw NAND）ですが、自分で Raw NAND を動かすための基板があると勉強になるし、SSD を自作できるというのは何だか夢があるのでは？そう思い自作 SSD 基板「PicoSSD」を設計してみました（図 1）。PicoSSD は市販のマイコンボードと組み合わせる拡張基板です。Airio-Base と Raspberry Pi Pico の両方と繋がるように設計しています。

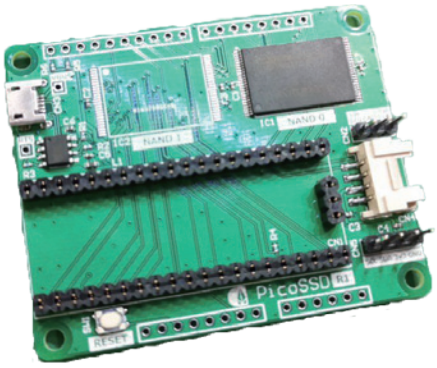


図 1: PicoSSD 外観

ホスト基板について

Airio-Base は LPC11U35、Raspberry Pi Pico は RP2040 マイコンを搭載しており、どちらも USB デバイス機能を持っているため PC と USB で接続してリムーバブルディスクとして認識させることが可能です。また、Airio-Base 以外の Arduino フォームファクターのマイコンボードを使用することも可能です。I/O 電圧は 3.3V なので注意してください。

Airio-Base（図 2）はクレイン電子にて設計・製造・販売しているマイコンボードで、Arm® Mbed™ Classic にて開発を行います。クラウドに開発環境があるため、ウェブブラウザのみでの開発が可能で、マイコンボードへの書き込みは USB のドラッグ & ドロップととてもお手軽です。詳細は商品紹介ページ[1]を参照してください。

Raspberry Pi Pico は Raspberry Pi 財団が開発したマイコンボードです。Cortex® M0+ マイコンが搭載されており、既存の Raspberry Pi とは異なり Linux™ OS は搭載できませんのでご注意ください。

ちなみに、両方のホスト基板を同時に挿した際には Raspberry Pi Pico の 3.3V レギュレータ出力（3V3OUT）が OFF になるため、I/O 電圧のコンフリクトは発生しないようになっています[2]。

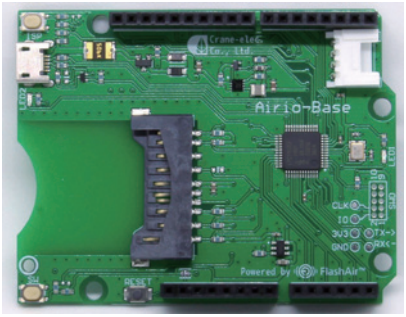


図 2: Airio-Base

1 https://crane-elec.co.jp/products/vol-14_airio-base/

2 ただし、故意に両方のボードを接続することはおやめください。

搭載部品などの情報

Raw NAND

PicoSSD に搭載している Raw NAND はキオクシアの SLC NAND「TC58NVG0S3HTA00」で、容量は 1Gbit、パッケージは TSOP48pin です(図 3)。データシートはキオクシアの web ページ[3]からダウンロードすることができます。

48 ピンパッケージですが NC ピンが多く、マイコンとのインターフェースは 15 本です。PicoSSD ではこの Raw NAND を 2 個搭載[4]できますが、Chip Enable 以外の信号は共用できるため、16 本接続すれば OK です。なお、WPB(/WP) をプルダウンしています。

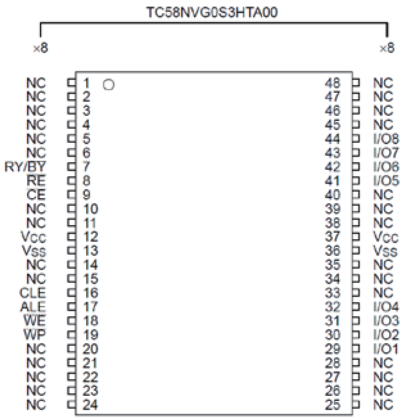


図 3: SLC NAND のピンアサイン

周辺回路

PicoSSD には USB 端子があります。これは、マイコンボードの USB 端子は PC との接続で使用するため、UART デバッグ用として使用してください。この USB 端子からは電源供給できません。また、I2C Grove 互換ポートがあり、温度センサや OLED を繋げることが可能です。

そのほか、リセット SW、Raspberry Pi Pico のデバッグ端子などが搭載されています。

入手方法など

PicoSSD の入手方法などの情報はクレイン電子 web ページを参照ください。面白い使い方があったら是非 Twitter などの SNS で投稿してみてください。

PicoSSD 製品紹介ページ：**https://crane-elec.co.jp/products/vol-22/**

3　https://business.kioxia.com/ja-jp/memory/slc-nand.html

4　デフォルトの PicoSSD では Raw NAND は 1 個だけ実装されており、2 個実装する場合はユーザにて部品手配・実装が必要です。

株式会社クレイン電子 (@crane_elec)　福屋 新吾

基板の半田付けをする会社をしています。PicoSSD 基板を使って皆様の開発ライフの一助になれば幸いです。

自作 SSD を動かそう！

村口

PicoSSD 基板を使い SSD をコンセプト実装してみました。まず、ストレージの役割について説明し、自作 SSD の構成概要、USB Mass Storage の仕組み、Raw NAND の特性、ベンチマーク結果等を説明します。

ストレージの役割

ストレージへのアクセス要求を階層ごとに分類すると、図 1 のようにアプリケーション階層、ファイルシステム階層、ブロックデバイス階層に分類することができます。SSD や HDD といったストレージは、ブロックデバイス階層に相当します。

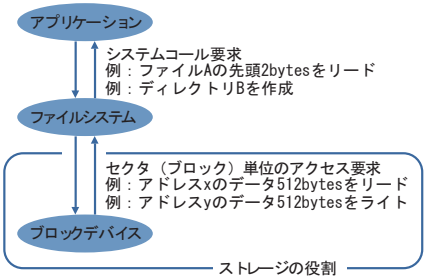


図 1: ストレージの役割

ファイルシステム階層では、ディレクトリ構造、ファイル名、ファイルのデータ格納場所が管理されており、上位階層の複雑なアクセス要求をブロックデバイス階層の単純なアクセス要求に変換します。ブロックデバイス階層の最小アクセス単位はセクタと呼ばれます。セクタサイズは、一般的に、512 ～ 4096[Bytes] で、ストレージ全域でセクタサイズは同一です。ファイルシステム階層が、アプリケーション階層からのシステムコール要求を下位のブロックデバイス階層のセクタ単位のアクセス要求に変換します。

つまり、ストレージの役割としては、固定サイズ（セクタサイズ）でランダムにリードライトするだけの機能を提供すればよいことになります。

自作 SSD の構成概要

自作 SSD の構成は図 2 のようになります。USB デバイス機能を持つマイコンボードの GPIO に不揮発メモリの Raw NAND を接続します。USB ホスト側からみると、USB Mass Storage として見えるようにマイコンファームウェアを構成します。この記事の執筆にあたり、Airio-Base でコンセプト実装を行いましたが、SRAM の大きい Raspberry Pi Pico も選択可能です。

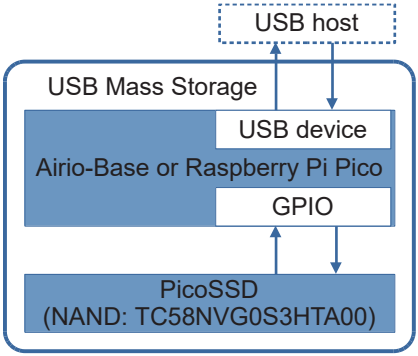


図 2: 自作 SSD の構成図

USB Mass Storage の仕組み

USB Mass Storage 機能については、Arm® Mbed™ 開発環境や、Raspberry Pi Pico 開発環境でも、便利なライブラリが用意されていますが、どのような仕組みで動作しているかについて概要を説明します。

USB デバイス初期化

USB デバイスが USB ホストに接続されると、Endpoint0 という仮想通信路を利用して初期化処理が行われます。ストレージ機能を提供する Bulk-Only Transport を利用するために、以下のような処理が行われます。

1. **固有アドレス割り当て**
   - **USB デバイスに固有アドレスを割り当てる。**
2. **各デスクリプタを介して以下の情報をホストに通知**
   - **デバイスクラス（USB Mass Storage）、**
   - **サブクラス（SCSI 透過型コマンドセット）、**
   - **インターフェイスプロトコル (Bulk-Only Transport)、**
   - **通信に使用する仮想通信路情報など。**

Bulk-Only Transport

初期化が完了すると、Bulk-Only Transport プロトコルに従って、コマンド、データ、ステータス通知のやり取りが行われます。Bulk-Only Transport プロトコルでは、右図のような Command Block Wrapper (CBW) コマンド発行ステート、データ転送ステート、Command Status Wrapper (CSW) ステータス通知ステートを遷移することでストレージ機能を提供します（図 3）。

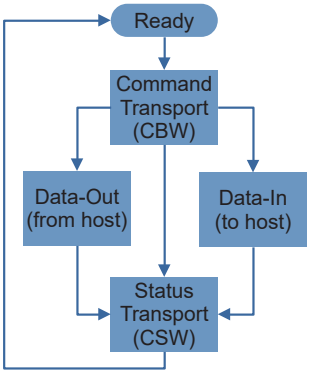


図 3: Bulk-Only Transport

Raw NAND の特性

Raw NAND には、次に説明する特性があるため、ブロックデバイスとして使用するには、Flash Translation Layer (FTL) という変換階層が必要になります。

部分的な書き換えが不得意

Raw NAND を部分的に書き換えるには、事前に、データ退避操作、消去操作 (全ビットを High にすること) が必要になります。その後のプログラム操作で所望の書き換えが完了します。自作 SSD のコンセプト確認では、LPC11U35 の SRAM 容量の制約内でデータ退避可能な容量で実現しているため、図 4 のように 1 ブロックあたり本来 136[KBytes] あるうち 2[セクタ] , 1[KBytes] 分だけを割り当てています。

不良ブロック対応

不良ブロックがあった場合、代替ブロックに切り替える仕組みが必要になります。自作 SSD のコンセプト確認では、初回に不良ブロックを走査しており、代替ブロックへの変換テーブルは、LPC11U35 マイコンの EEPROM に保存しています。

図 4 のように Raw NAND 2 chip 構成の後半 40 ブロック分を、不良ブロックが出現したときの代替ブロックとして確保しておきます。

データ誤り訂正

書き込んだデータが読み出し時に変化する場合に備えてエラー訂正が必要になります。データシートによると、512[Bytes] につき 8bit のエラー訂正能力が必要とされています。自作 SSD のコンセプト確認では未実装です。

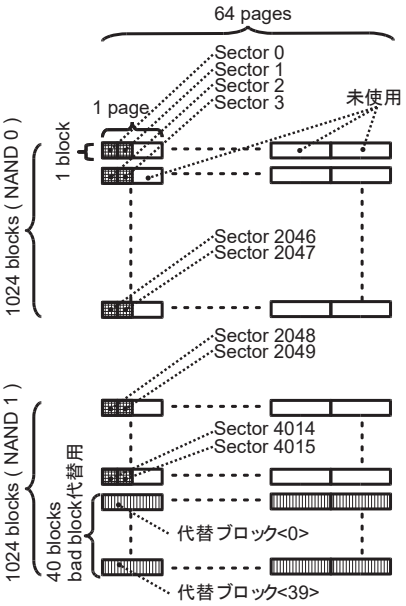


図 4: ブロック割り当て

消去・書き込み回数の平準化

NAND メモリセルは、消去・書き込みを繰り返すことで劣化していきます。特定のセルに劣化が集中しないように消去書込箇所を分散させる操作を行うことが望ましいです。自作 SSD のコンセプト確認では未実装です。

自作 SSD を使うための手順

Airio-Base への firmware(FW) の書き込み手順

Airio-Base は株式会社クレイン電子 の Arm® Mbed™ 互換マイコンボード製品です。開発はブラウザで動作する Mbed™ 環境を用います。Mbed™ の自作 SSD 用ソースコード[1]を「Import into Compiler」を使用してインポートします。main.cpp を選択し「コンパイル」をクリックすると、ブラウザから FW をダウンロードすることができます。

ISP ボタンを押しながら[2] USB 接続すると、Airio-Base が PC に「CRP_disable」という名前のドライブとして認識されます。ドライブを開くと firmware.bin というファイルがあるため、このファイルを削除した後に FW をドラッグ & ドロップで書き込みます。

Airio-Base と PicoSSD を接続し、PC と接続すると、USB ストレージとして認識します。

1 PicoSSD 製品情報ページを参照してください https://crane-elec.co.jp/products/vol-22/

2 ISP ボタンを押しながらリセットボタンを押しても良いです。

Windows で認識させるための手順

FW を書き込んだ自作 SSD を Windows で認識させるためには、Linux™ を使用してパーティション作成・フォーマットする必要があります。root 権限で行ってください。

- **デバイスの認識**

```
# ls /dev
```

/dev/sda があることを確認してください

- **セクタサイズなどの確認**

```
# fdisk -l /dev/sda
```

- **パーティション作成**

```
# fdisk /dev/sda
```

fdisk は対話型ツールです。下記のコマンドで新規パーティションを作成します。
p: 現在の値を確認
n: 新規パーティション作成
w: 書き込みして終了

```
# ls /dev
```

/dev/sda1 があることを確認してください。

- **フォーマット**

```
# apt-get install dosfstools
```

mkfs のフォーマットを用意したので、下記のコマンドでフォーマットを行います。

```
# mkfs -t msdos -n USBDATA /dev/sda1
```

Windows と接続すると「USBDATA」という名前のドライブが認識されます。

自作 SSD のストレージ容量とベンチマーク結果

ストレージ容量

利用可能なセクタ数は 4016[セクタ] で、容量にして約 2[MBytes] です。

ベンチマーク結果

転送速度は、リード 20[KBytes/s], ライト 6[KBytes/s] です。

村口

大いに改良のしがいのある自作 SSD となりました。PicoSSD 基板を利用すると、実用性はともかく、ご自身の手で、お手軽に、SSD の FTL を実装・実験することができます。PicoSSD 基板には USB-UART 変換ありデバッグも容易です。ご興味のある方は、ぜひ、ご活用下さい。

自作 SSD で動かす自作 CPU

にちか

「CPU の創りかた」[1] という名著で、TD4（とりあえず動作するだけの 4 bit CPU）が紹介されています。ところで、この CPU は最大 16 個の命令しかプログラムできません。折角 CPU を自作するなら、もっと沢山の命令を動かしたいと思いました。そこで、各レジスタと加算回路を 2 倍に拡張した 8 bit CPU（名付けて TD8）を作ります。

しかし、ここで大変なのが ROM モジュールの製作です。TD8 の ROM は、12 bit × 256 命令分の配線をする必要があり、DIP スイッチで作るのは大変な手間です。また、その大規模な ROM を都度手作業でコーディングするのも現実的ではありません。

そこで当記事では、PicoSSD を ROM として用いることで、この問題を解決します。

ハードウェア構成

TD8

外観を図 1 に示します。回路設計は、「CPU の創りかた」の 11 章 3 節を参照しました。PCB 設計は、先駆者様の設計[2]を参考にさせていただきました。ROM を中央のピンで外付けする仕様も、先駆者様の設計を参考にしました。

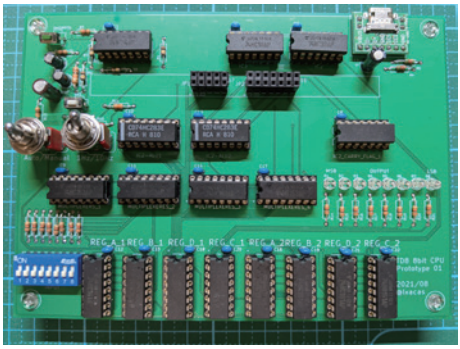


図 1: TD8 の外観

自作 SSD

パーツを図 2 に示します。自作 SSD は、Airio-Base（緑の基板）と SLC NAND シールド（青の基板）から構成されます。I/F 回路（赤の基板）を間にスタックすることで、NAND の読み書きを行いつつ、TD8 の ROM として機能させます。Airio-Base のピンの殆どは NAND 制御に費やしているため、I/F 回路の入出力ピンは、I2C I/O エキスパンダで代替しています。

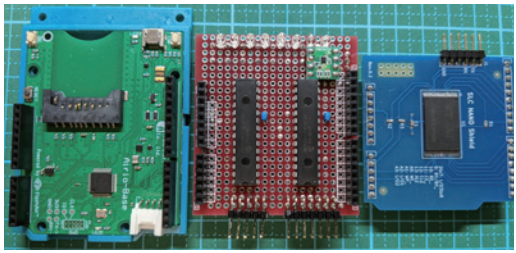


図 2: 自作 SSD の構成パーツ

1　渡波郁「CPU の創りかた」株式会社 マイナビ出版 (2003)

2　https://wireless-square.com/2017/07/31/cpu の創りかた。td4 を作ってみました。/

プログラム

PicoSSD の自作 SSD サンプルプログラム[3]を少し改造することで、自作 CPU の ROM として機能するようになります。本章はその手順を紹介します[4]。

自作 SSD（ROM バイナリ書き込み時）

USBMSD_STEP1 クラスを編集し、 UART 出力を仕込みます。

```
int USBMSD_STEP1::disk_write(const uint8_t* data, uint64_t block,
uint8_t count)
{
  uint64_t address = bbt->getTranslatedBlockAddress(block);
  ft232.printf( “Write address is %lx .¥n” ,address);
```

Windows マシンに自作 SSD を USB 接続し、あらかじめ用意した ROM バイナリを書き込みます。この時 UART をモニタし、ROM バイナリの書き込み先アドレスをメモします。

```
Write address is 30 .
```

自作 SSD（ROM として動作させる時）

こちらもサンプルプログラムを基に改変します。まず、前の手順でメモしたアドレスを、起動時に読み出すように変更します。

```
//myusb->connect(); # USB デバイスとして使用しないのでコメントアウト
uint8_t ROM[0x200];
myusb->disk_read(ROM, 0x30,5 0x00); # メモしたアドレスから ROM を読み出し
```

次に、MCP23017 ライブラリをプロジェクトにインポートし、ROM の入出力 I/F として振る舞うようにコーディングします。これで自作 SSD を ROM にする準備は OK です。

```
#include “MCP23017.h”
I2C i2c(P0_5、P0_4);
MCP23017::MCP23017 input(i2c,MCP23017_DEFAULT_ADDR);
MCP23017::MCP23017 output(i2c,MCP23017_DEFAULT_ADDR+8);
(略)
int main()  {
(略)
i2c.frequency(400000);
 input.init();output.init(); //software reset
input.setDirrection(MCP23017_PORTA, 0xFF); //set PORTA Input
input.setPullUp(MCP23017_PORTA, 0xFF);//set PORTA PullUp
  output.setDirrection(MCP23017_PORTA, 0x00); //set PORTA Output
output.setDirrection(MCP23017_PORTB, 0x00);//set PORTB Output
  while(1) {
    uint8_t Pcount=input.read(0);
    output.write(MCP23017_PORTB,ROM[Pcount]);
    output.write(MCP23017_PORTA,ROM[0x100+Pcount]);
}}
```

3 PicoSSD 製品情報ページを参照してください https://crane-elec.co.jp/products/vol-22/

4 当記事で紹介する手順に従って生じた損害について、筆者及び編集者は一切の責任を負いかねます。

5 https://os.mbed.com/users/hsgw/code/MCP23017/

自作 CPU(ROM)

自作 SSD はセクタ（512B）単位で読み書きするため、TD8 の ROM 1個分が丁度1セクタに収まります。セクタの前半 256B は、各ステップの命令コードを保存し、後半 256B は、イミディエイトデータを保存することにしました。

例として、LED をチカチカ光らせ続ける ROM コードを以下に示します。

```
0000000 0b 0b 0b 0b 0b 0b 0b 0b 0b 0f 00 00 00 00 00 00
0000010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
*
0000100 33 66 cc 88 88 cc 66 33 11 00 00 00 00 00 00 00
0000110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

おまけ

自作 CPU & SSD で、“**生命、宇宙、そして万物についての究極の疑問の答え**”を計算します。元ネタは、「銀河ヒッチハイク・ガイド」[6]という小説でスパコンが計算した問題です。物語では、この壮大な問いへの答えが「42」でした、とオチがつきます。そこで当記事では、“生命、（省略）究極の疑問”を、「**答えが 42 になる計算**」と勝手に定義してしまいます。TD8 は 8 bit 以下の整数を扱えるため、まさに打って付けな計算問題と言えます。

計算方法

42 は 5 番目のカタラン数のようです。カタラン数は、以下の式で計算できます[7]。

$$C_n = \frac{1}{n+1}{}_{2n}C_n$$

つまり、$10 \times 9 \times 8 \times 7 \ / \ (5 \times 4 \times 3 \times 2)$ を計算します。しかし、TD8 は加算しかできません。乗算は、足し算の繰り返しで代替可能ですが、いかに割り算をするかが課題です。

そこで、代数学の群論を導入します。TD8 は 0 から 255 の整数しか扱えないため、TD8 が行う割り算は $a \div b$ ですが、$(a \div b) \bmod 256$ であるとも見做せます（a、b は整数）。ここで n mod N という式は、「整数 n を整数 N で割った余り」を表します。

もし b が奇数の場合、256 と互いに素なため、256 を法とした時の b のモジュラ逆数 b^{-1} を求めることができます。ここで「N を法とした時の n のモジュラ逆数」とは、$nx \equiv 1 \bmod N$ となるような整数 x のことをいいます。モジュラ逆数が求まるならば、割り算は、モジュラ逆数を掛けることとして定義できます。例えば $10 \div 5$ を計算したい場合、代わりに 205 を掛けることで、$10 \times 205 \bmod 256 = 2$ と求まります。また、整数 b が 2 のべき乗の場合、右ビットシフトが除算の代わりとなります。

6　ダグラス・アダムス、安原和見「銀河ヒッチハイク・ガイド」河出書房新社 (2005)

7　https://manabitimes.jp/math/657

以上から、÷ 5 は× 205 に、÷ 4 は >>2 に、÷ 3 は× 171 に、÷ 2 は >>1 に置き換えられます。つまり 5 番目のカタラン数は、((10 × 9 × 8 × 7 × 205)>>2) × 171>>1 mod 256 を計算することで求められます。ここで 171 は，3 のモジュラ逆数です。

ROM コード

前節で立てた式を計算し、LED に出力する ROM コードを以下に示します。

```
0000000 03 07 00 05 0e 0f 05 00 0e 0f 00 05 0e 0f 00 0e
0000010 00 05 00 0e 00 05 00 0e 00 05 00 0e 00 05 00 0e
0000020 00 05 00 0e 00 05 03 00 05 0e 0f 00 0e 00 05 00
0000030 0e 00 05 00 0e 00 05 00 0e 00 05 00 0e 00 05 00
0000040 0e 00 05 00 0e 00 05 03 00 05 0e 0f 04 09 ff 00
0000050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
*
0000100 00 0a 09 ff 06 02 cd ff 0a 06 08 ff 0e 0a 80 12
0000110 80 20 40 16 c0 10 20 1a e0 08 10 1e f0 04 08 22
0000120 f8 02 04 26 fc 01 00 ab ff 2b 27 80 2f 80 40 40
0000130 33 c0 20 20 37 e0 10 10 3b f0 08 08 3f f8 04 04
0000140 43 fc 02 00 47 fe 01 00 07 ff 4c 48 00 00 4e 00
0000150 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

結果

無事計算ができました。計算時間は 10Hz 動作モードで 92 秒かかりました。出力ポートに接続した LED に 2 進数で「00101010」と表示されました（図 3）。これは 10 進数に直すと「42」です。

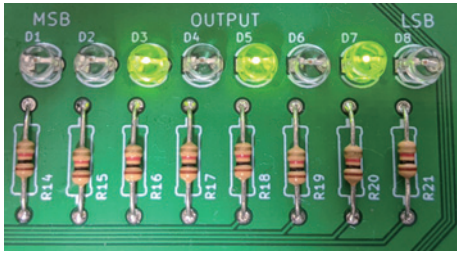
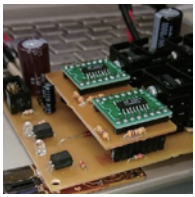


図 3: 生命、宇宙（省略）究極の疑問の答え

おわりに

自作 SSD を ROM にして、自作 CPU を動かすことができました。また、当記事で構成した計算機で、“**生命、宇宙、そして万物についての究極の疑問の答え**”を計算できました。当記事の執筆にあたって、偉大な先人達の作品・知見をいくつもお借りしました。この場を借りて御礼申し上げます。



にちか（@lxacsa）
原稿のネタを考えるのはワクワクして楽しかったです！！
TD8 の基板が欲しい方は（もしいれば）DM ください。

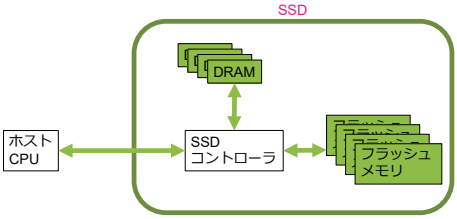
SSD コントローラのお仕事

藤澤 俊雄

みなさん毎日お仕事や趣味で PC やスマホをお使いになっていると思います。PC にはデータを記憶するためにハードディスクライブ (HDD) が多く使われていましたが、近年は筐体が小さな SSD[1] に、徐々に置き換わってきています。SSD は半導体であるフラッシュメモリを使った記憶装置ですが、取り扱いに少し工夫が必要であり、効率良くデータを出し入れするために、SSD コントローラが (中に隠れて) 活躍しています。スマホにも、SSD にとても良く似た記憶装置が内蔵されています。今回、SSD コントローラがしているお仕事について、簡単に紹介します。

SSD の中身

一般的な SSD の中身を図 1 に示します。CPU から受けとったデータを格納するフラッシュメモリ、データの書き込みと読み出しを制御する SSD コントローラ、制御に必要なデータを格納する DRAM[2] から構成されています。記憶容量が大きな SSD では、数十枚～数百枚の半導体チップが使われています。



※ DRAM: Dynamic Random Access Memory, 高速で高価な揮発性のメモリ
CPU: Central Processing Unit, スマホやPCのコントローラ

図 1: SSD の構成

SSD の基本的な動作 (データ書き込み)

データ書き込みの動作を図 2 に示します。SSD コントローラは CPU から書き込み要求と書き込みデータと書き込み先 (論理アドレス) を受け取ると、データをフラッシュメモリに書き込みます。

データの書き込み先として、CPU から論理アドレスを受け取りますが、SSD の物理アドレスに変換します。アドレスを変換する理由は、フラッシュメモリは一か所を頻繁に書き換えると壊れてしまうからです。SSD コントローラは、同じ物理アドレスへの書き込みが連続して来た場合でも、フラッシュメモリの別の場所に書き込むことによって、SSD が壊れることなく長く使えるようにしています。

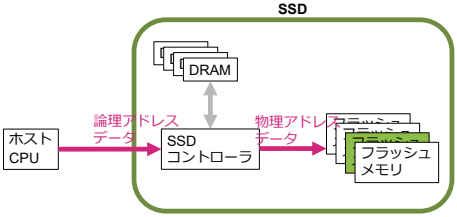


図 2: データの書き込み

1 SSD: Solid State Drive, 半導体不揮発メモリを使った記憶装置
2 DRAM: Dynamic Random Access Memory, 揮発メモリ

論理アドレスと物理アドレスのつながり(組み合わせ)は、論物変換テーブルと呼ばれます。SSD コントローラは、データをフラッシュメモリに書き込んだ後に、論物変換テーブルを DRAM に格納し、データ読み出し時に参照して取り出せるようにしておきます(図3)。

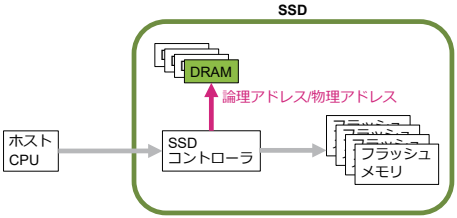


図 3: 論物変換テーブルの生成

また、SSD コントローラはデータだけでなく、DRAM 上の論物変換テーブルも時々にフラッシュメモリに書き込み、もし SSD の電源が切られても、後から論物変換テーブルを復元できるようにしています。

SSD の基本的な動作(データ読み出し)

次に、データ読み出しの動作を図 4 で説明します。SSD コントローラは CPU から読み出し要求と、読み出し先アドレス(論理アドレス)を受け取ると、まず DRAM を参照し、論理アドレスに対応する物理アドレスを探して取り出します(図 4)。

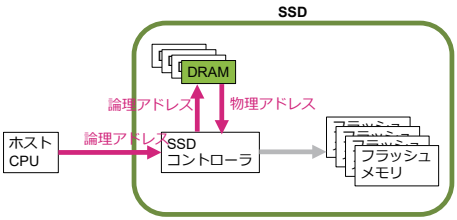


図 4: 論物変換テーブルの参照

続いて、DRAM から取り出した物理アドレスを使って、フラッシュメモリから対象のデータを読み出します(図 5)。そしてそのデータを CPU に渡します。

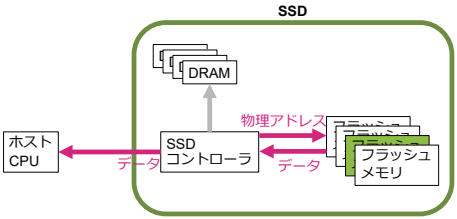


図 5: データの読み出し

フラッシュメモリは、半導体の酸化膜中に電子を閉じ込めてデータを記憶する構造なので、時間経過後に読み出すと、データに誤りが含まれることがあります。SSD コントローラは、正しく読まれたデータを使って、誤ったデータを修復する重要な仕事(エラー訂正)も担っています。

SSD の高性能化

フラッシュメモリはその性質上、1 回の読み出しは数十マイクロ秒、1 回の書き込みは数百マイクロ秒〜数ミリ秒かかります。つまり、フラッシュメモリ 1 つに対する書き込み速度は数十 MB/ 秒であり、HDD の性能とあまり変わりません。しかし、SSD はフラッシュメモリを多数同時に動かすことによって、数百〜数千 MB/ 秒の高性能を発揮します。

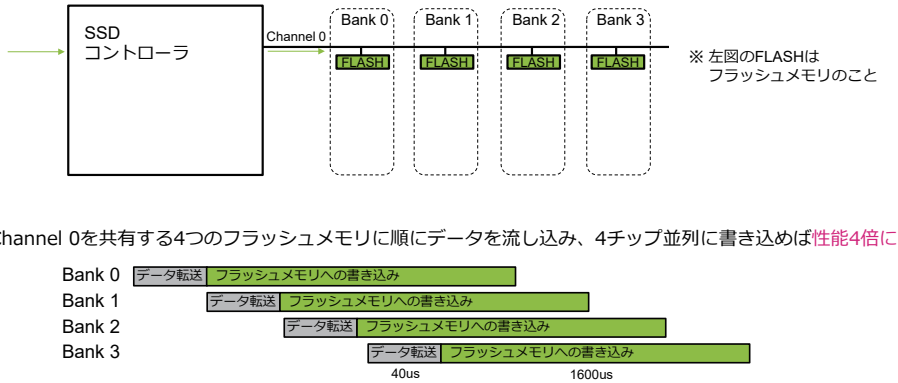


図 6: 並列書き込みによる高性能化（4 倍）

図 6 は、SSD コントローラに 4 つのフラッシュメモリチップを接続して、書き込みを行った例です。SSD コントローラが 1 つ目のフラッシュメモリのデータバッファにデータを流し込み (灰色の時間)、続いて書き込みを要求すると、1 つ目のフラッシュメモリは数ミリ秒かけてデータバッファからメモリセルへの書き込みを行います (緑色の時間)。ここで、SSD コントローラはその書き込み完了を待たずに、すぐに 2 つ目のフラッシュメモリにデータを流し込みます (上から 2 つ目の灰色の時間)。同様に、2 つ目のメモリセルの書き込み完了を待たずに、3 つ目のフラッシュメモリにデータを流し込みます (上から 3 つ目の灰色の時間)。このように制御することによって、4 つのフラッシュメモリチップに対して同時に書き込み動作をさせることができます。この場合、書き込み性能はほぼ 4 倍になります。

SSD コントローラからフラッシュメモリにデータを流し込む経路をチャネルと呼びます。チャネル数を増やすことによって、さらに並列数を上げることができます。図 7 は、2 チャネルを使って 8 つのフラッシュメモリに同時書き込みをした例です。この例では、1

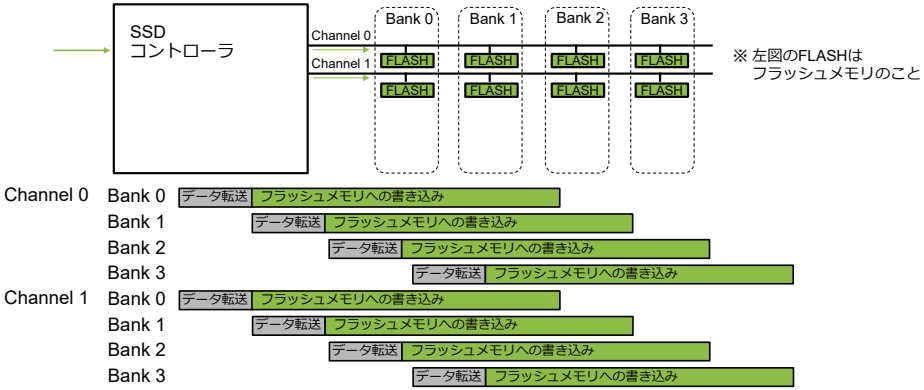


図 7: 並列書き込みによる高性能化（8 倍）

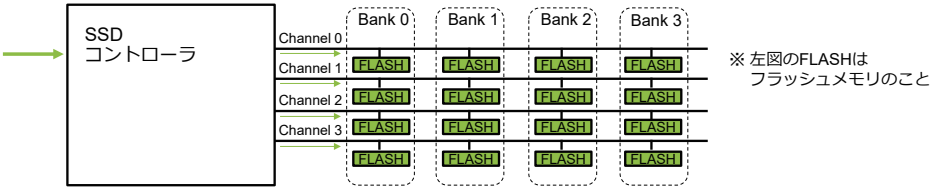


図 8: 並列書き込みによる高性能化（16 倍）

つ目のフラッシュメモリと 5 つ目のフラッシュメモリのそれぞれに、2 つのチャネルを使って同時に書き込みデータを流し込み、さらに、同時にメモリセルへの書き込みを開始しています。このように制御することによって、8 つのフラッシュメモリチップに対して同時に書き込み動作をさせることができます。この場合、書き込み性能は約 8 倍になります。

同様に、チャネル 4 つを使ってフラッシュメモリ 16 個に同時書き込みを行い、書き込み性能を 16 倍にした例を図 8 に示します。

エンタープライズ向け SSD やデータセンター向け SSD では、例えばチャネル数を 16 以上、同時書き込み数を 64 以上に引き上げることによって、数千 MB/ 秒の非常に高い書き込み性能を実現しています。読み出し性能の向上については説明を省略しましたが、同様の考え方で性能向上できます。ただしフラッシュメモリの読み出しは書き込みより速いので、SSD コントローラがボトルネックにならないよう、処理能力を併せて向上させる必要があります。

このように、体積のとても小さな半導体チップを多数用いて性能をスケールアップできることが SSD の大きなメリットの一つです。注意点としては、性能向上に伴って、製品コストや消費電力も大きくなることがデメリットにもなるため、コスト・消費電力を抑制する技術開発や、お客様の要求に合った製品仕様企画も重要になります。

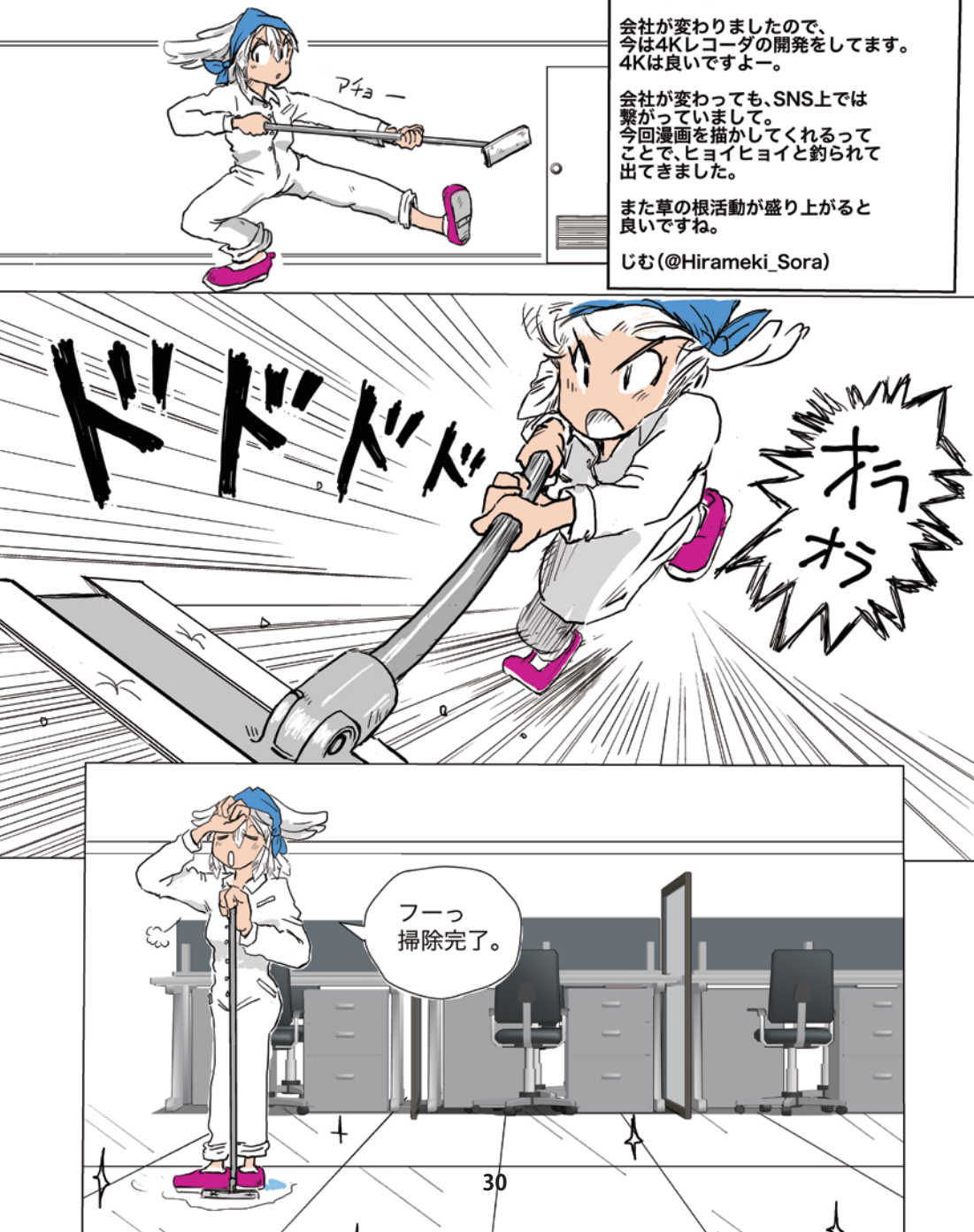
おわりに

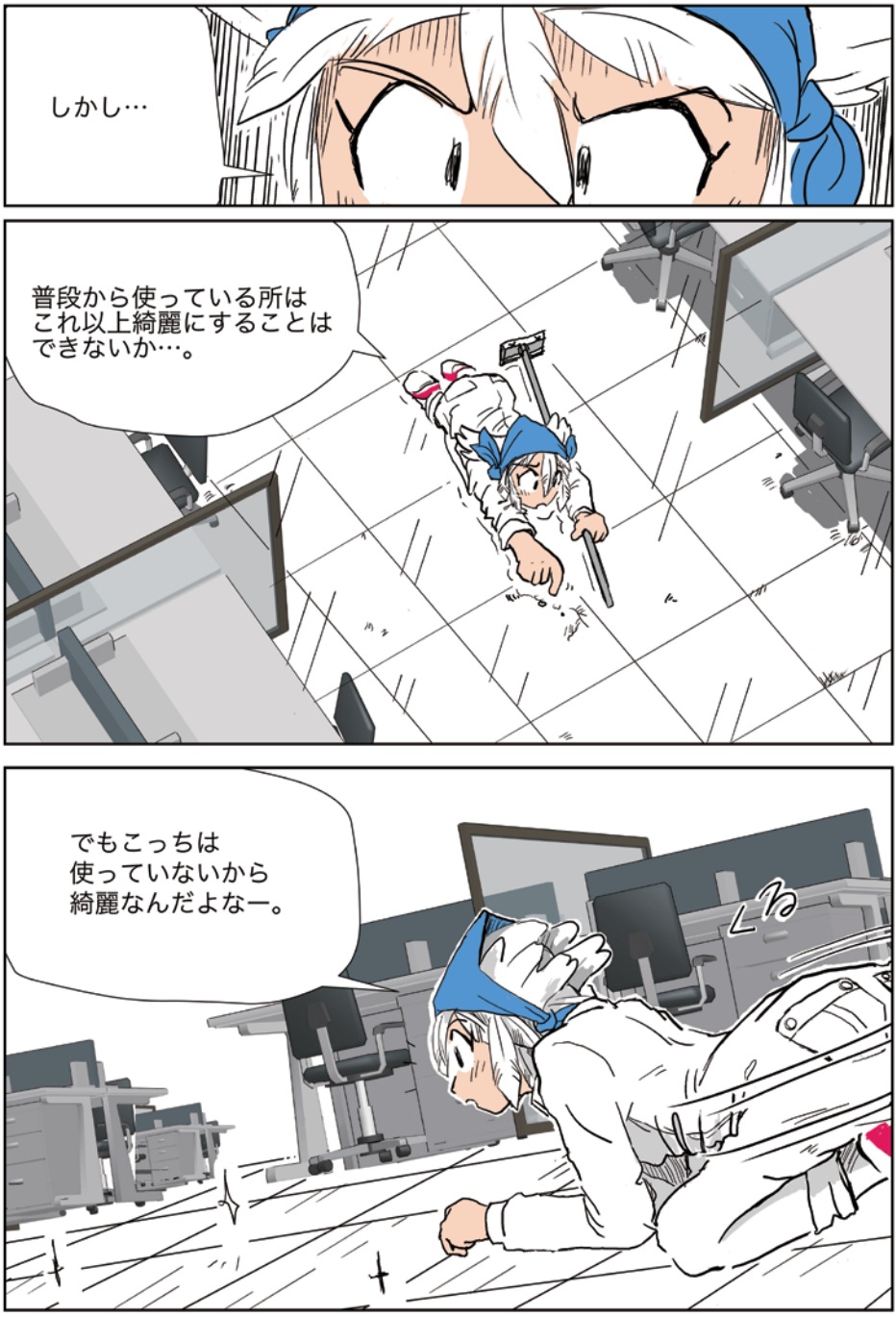
今回は、SSD の基本的な構成要素と、SSD コントローラの基本的な役割について説明しました。これを読んで、SSD を少し身近に感じたり、少し興味を持っていただけたら幸いです。今後も、もしまた機会がありましたら、SSD コントローラのより詳しい機能や、SSD 小型化・大容量化技術、普及させるための規格化活動、将来の SSD に向けた研究開発動向なども、紹介させていただければと思います。では。

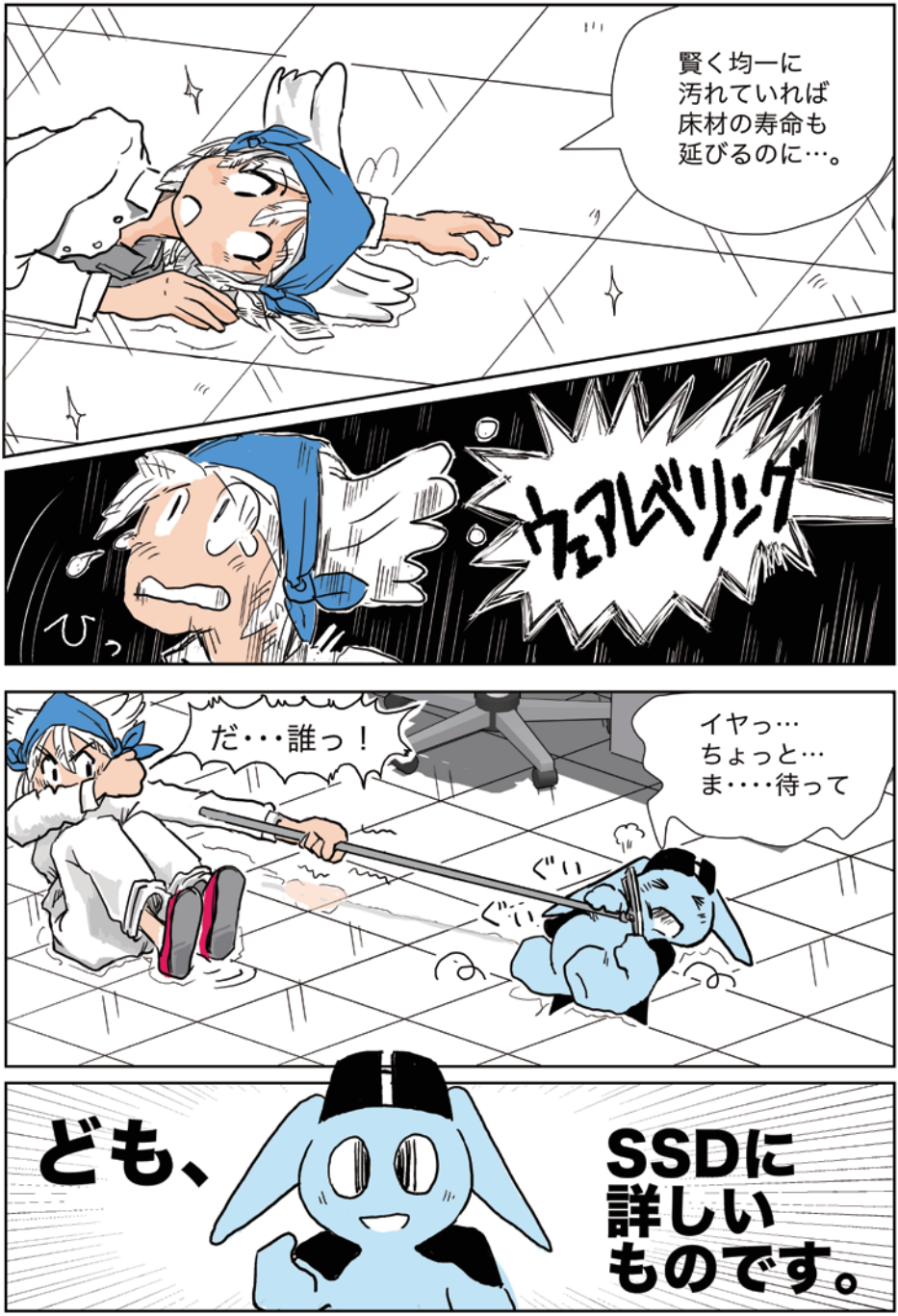
藤澤 俊雄

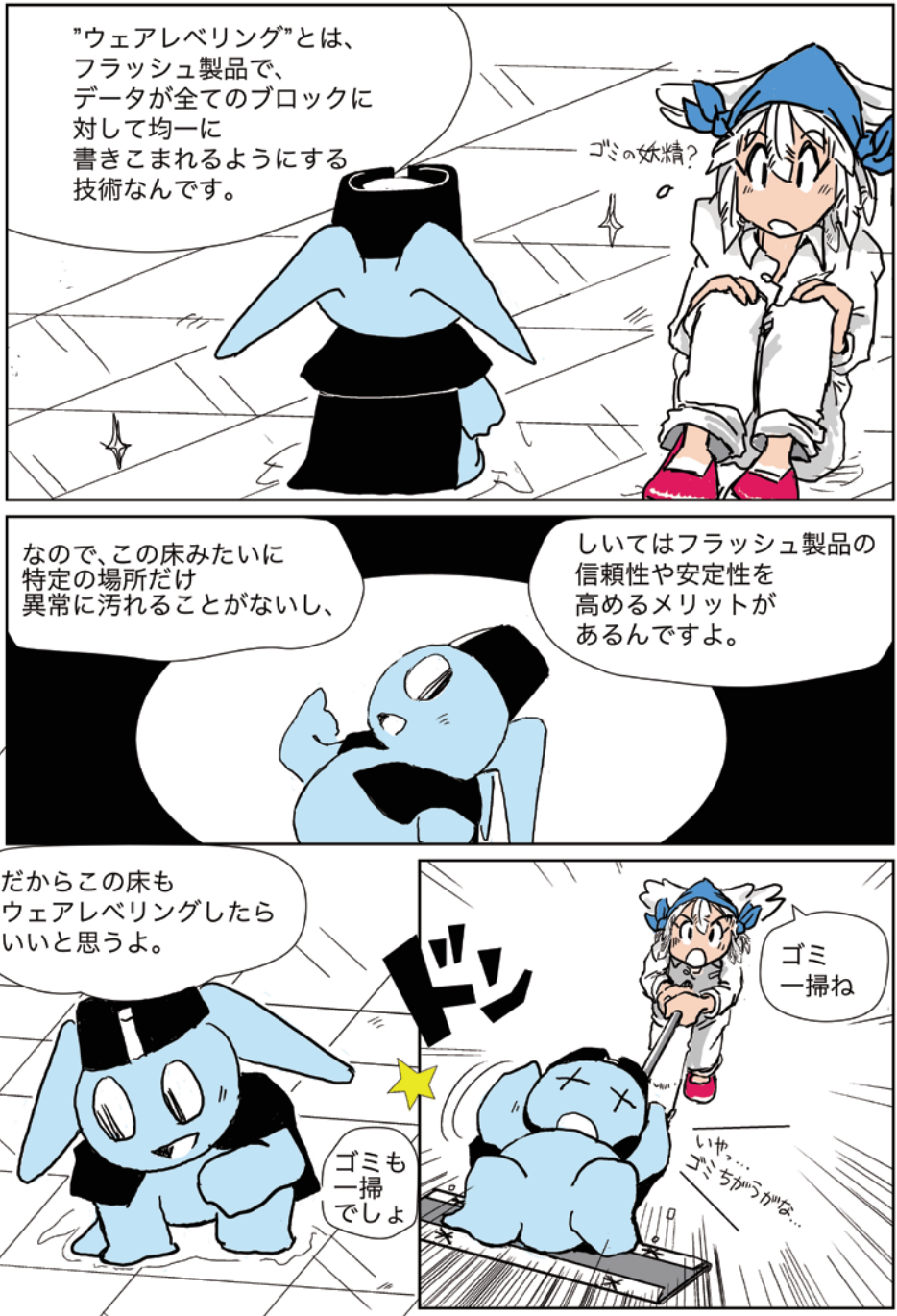
SSD 開発に携わって 10 年以上になりますが、フラッシュメモリは世代ごとに色々な性格を持っていて、たいへん追究し甲斐があります。今後もまだまだ楽しむことができそうです。

漫画「ウェアレベリング」


【自己紹介】
会社が変わりましたので、
今は4Kレコーダの開発をしてます。
4Kは良いですよー。
会社が変わっても、SNS上では
繋がっていまして。
今回漫画を描かしてくれるって
ことで、ヒョイヒョイと釣られて
出てきました。
また草の根活動が盛り上がると
良いですね。
じむ(@Hirameki_Sora)
アチョー
ドドドドド
オラオラ
フーっ
掃除完了。


しかし…
普段から使っている所は
これ以上綺麗にすることは
できないか…。
でもこっちは
使っていないから
綺麗なんだよなー。


賢く均一に
汚れていれば
床材の寿命も
延びるのに…。
ウェアレベリング
ひっ
だ・・・誰っ！
イヤっ…
ちょっと…
ま・・・・待って
ぐい
ぐい
ども、
SSDに
詳しい
ものです。


"ウェアレベリング"とは、
フラッシュ製品で、
データが全てのブロックに
対して均一に
書きこまれるようにする
技術なんです。
ゴミの妖精？
なので、この床みたいに
特定の場所だけ
異常に汚れることがないし、
しいてはフラッシュ製品の
信頼性や安定性を
高めるメリットが
あるんですよ。
だからこの床も
ウェアレベリングしたら
いいと思うよ。
ゴミも
一掃
でしょ
ドン
ゴミ
一掃ね
いやっ…
ゴミちがうがな…

ウェアレベリングとは？

Pochio

「ウェアレベリング」をテーマにした久々のソラちゃんまんがはいかがでしたか？きっと「なんとなくイメージは分かったけど・・・、よくわかりません」という方がいらっしゃるかと思います。そこで、ここからは補足説明をさせていただきます。ちなみに「ソラちゃんって何者ですか?」という方は、検索サイトにて「閃ソラ」で検索してみてくださいね。

ウェアレベリングの目的は？

フラッシュメモリ製品にとって大切な「ウェアレベリング」という技術、インターネット上にもWikipediaをはじめとする解説サイトがありますので、フラッシュメモリに詳しい方ならご存知かもしれません。ここでは言葉そのものをご存知なかった方のために、どういう技術で、どうして大切なのかをごく簡単にお伝えできればと思います。

フラッシュメモリは、メモリセルにあるフローティングゲートに電子を蓄えることで、データを記録します。フラッシュメモリのデータの記録と消去、およびデータを読み出す仕組みについてはここでは割愛しますが、学研プラス刊「学研まんがでよくわかるシリーズ144 フラッシュメモリのひみつ」にて分かりやすく解説されていますので、そちらをぜひご参照ください。小学生向けのまんがですが、理工系の大学の先生から好評をいただいています。なお、この本は非売品ですが、全国の図書館で借りることができます。また、インターネット上にて無料で読むこともできます。ぜひ「フラッシュメモリのひみつ」で検索してみてください。

話を戻します。さて、フローティングゲートに蓄えられた電子は、絶縁膜（ゲート酸化膜）のおかげで勝手に外に出られません。そのため、フラッシュメモリはデータを記録し続けることができるのです。しかしデータを書いたり消去するときは、高い電圧が印加されて電子が絶縁膜を通り抜けます。そして何度も書き込みと消去の動作を繰り返すと、絶縁膜は徐々に劣化していき、最終的には寿命を迎えてしまいます。

ということは、いつも同じメモリセルばかりにデータを書いたり消去し続けると、そのメモリセルの絶縁膜は他と比べてあっという間に劣化してしまいます。劣化してしまうとデータを記録できなくなります。

一方で、全く使用されていないメモリセルがそのご近所にあったとしましょう。そのメモリセルの絶縁膜は未使用ですから、まんがに登場した「普段使っていないピカピカのタイル」みたいな状態と推測されます。となると、こっちはボロボロなのに、あっちはピカピカ。なんともバランスの悪い話です。この偏りを解消できれば、つまりどのメモリセルにもまんべんなくデータの書き込みや消去が行われるようにすれば、メモリセルの寿命を早

めてしまうことを防止できると考えられます。

この「まんべんなくデータの書き込みや消去」をする技術のことを、「ウェアレベリング」とよびます。ウェアレベリングを実現する方法には様々なものがありますが、ここでは簡単な例についてご紹介しましょう。

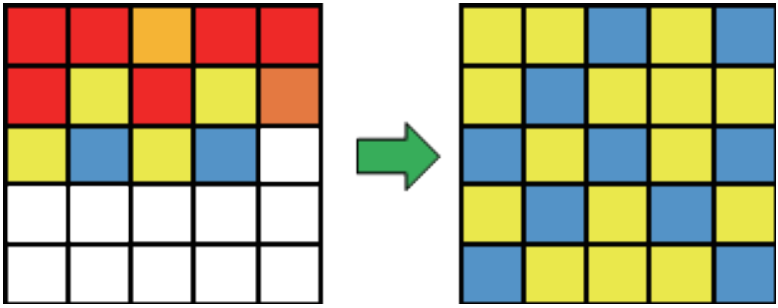


図 1：ウェアレベリングのイメージ。（左）ウェアレベリングなし、（右）あり。1 マスが 1 ブロックを表す。赤いブロックは何度も消去されたブロック。

どうやってまんべんなく書き込むのか？

フラッシュメモリは、記録されたデータをメモリセル単位で消すことができません。メモリセルが集まった「ブロック」とよばれる単位で一括消去します。したがってブロック単位で考えると、「何度も消去されたブロック」ほど寿命を早めてしまうことが心配されます（図 1 左）。そこで、ブロックごとに消去した回数を覚えておくことにしましょう。

突然ですがここでクイズです。ウェアレベリングを実現するためには、どんな戦略でデータを書き込むのがよいと思われるでしょうか。特定のブロックに書き込みや消去が偏らないようにするには、どうしたらよいのでしょう。

いろいろな方法が考えられるかもしれませんが、先ほど覚えておくことにした、「ブロックごとに消去した回数」を参考に考えてみます。もし消去回数の多いブロックにデータを書き込むと、寿命を早めてしまうかもしれません。そこで、消去回数が少ないブロックから書き込むことにすれば、全体でブロックの書き込み消去回数が均等になることが期待されます（図 1 右）。実際には SSD に搭載されているコントローラがウェアレベリングを行うことで、SSD の長寿命化に貢献しています。

ちなみに余談ですが、2019 年の Maker Faire Tokyo において、フラッシュメモリの書き込みや消去の動作をモチーフにして製作したカードゲーム、「Flash Fight」をご紹介しました。各プレーヤーが順に持ち札をゲームマット上の 5 つのブロックに置いていくのですが、カードの置き方にウェアレベリング的な戦術があります。もしこのカードゲームをお持ちの方がいらっしゃいましたら、試しに各ブロックにまんべんなく持ち札を置くようにしてみてください。必勝法ではありませんが、多少は勝ちやすくなると思いますよ。

大容量 SSD と OpenStreetMap

一般社団法人オープンストリートマップファウンデーションジャパン　松澤 太郎

OpenStreetMap(以下、OSM) という仕組みをご存知でしょうか？　多くの場合知っているという人でも背景地図として利用されているのを見ているだけだと思われます。実際、OSM は自由に使えることから多くのウェブサイトで背景地図として使われていますが、その実体は世界中の地理データを単一のデータベースで管理する仕組みであることを知らない方が大半だと思われます。世界中の人が単一のデータベースをみんなで更新するという世界でもあまり例の無い仕組みで、ファイル交換形式としては XML 形式のデータベースとして出力をされます。

基本的には node、way、relation の三要素が主なデータとなり、XML のタグがコミュニティによってガイドラインが決められていて、この要素をうまく使うことによって背景地図を作ったり、経路探索を行ったりします (経路探索時は道路の属性が必要なので way のタグがどうなっているかが重要になります)。基本的には Web サイト[1] のエディタや、JSOM といった高機能なエディタを使って編集をすることになります。Web サイトからも気軽に編集ができるので、興味を持った人は是非アクセスしてみてください。また、データベースのライセンスは ODBL というライセンスで商用利用も可能です。

さて、現在世界中 (Planet と呼びます) のデータで XML にして 1.5TB ほどのデータとなります。そのため、一般的には指定した小さな領域を API で取ってきたり、bzip2 圧縮をしたものや Protocol Buffer 形式でのやりとりをします。なお、現在 Protocol Buffer 形式では 60GB ほどまで圧縮されます。

今回はキオクシア様から PC を貸し出ししていただき、大容量の SSD と OSM のデータについて検証をしてみました。

貸し出し PC の詳細

今回お借りした PC は以下の仕様になっています。

- ROG STRIX Z390-F GAMING
- Intel® Core™ i7-9700F 3.00GHz (8core/8thread)
- DDR4 PC4-19200 CL15 8GBx2
- SSD KXD5YLN13T84 (3,840GB NVMe™)

SSD は一般には販売されていないデータセンター向けのものとなっています。詳細は HP[2] を参考にしてください。

1　https://openstreetmap.org
2　https://business.kioxia.com/ja-jp/ssd/data-center-ssd/xd5.html

OS は Debian bullseye を利用し、Prometheus™ によるストレージの速度計測を実施しています。特に顕著なストレージの高速利用のケースがあった場合はそれを記載します。また PostgreSQL は pgtune[3] をベースにチューニングを行います。

OverPass API のデータベース作成

OverPass API とは OSM で標準的に使われている API 仕様で、呼び出し及びレスポンスがともに XML 形式を利用しています。また、OverPass turbo という API では XML ではなく簡略化した形式でもアクセスが可能です。

今回はまずこの OverPass API に使えるデータベース (独自形式) を作成するベンチマークを取ってみました。OverPass 自体のインストールは OpenStreetMap wiki の手順[4]に沿って行います。インポート自体は init_osm3s.sh を実行するだけです。結果としては以下の通りとなりました。

```
bin/init_osm3s.sh ../data/planet-210816.osm.bz2 $DB_DIR $EXEC_DIR   107634.58s
user 12608.34s system 91% cpu 36:32:35.51 total
```

1.5 日ほどでインポートが行えました。では、インポートしたものを使って検索をしてみましょう。

```
bin/osm3s_query --db-dir=./db
```

あとは入力したい検索条件を入れて、Ctrl+D で結果を見ることができます。

```
<query type="node"><bbox-query n="51.0" s="50.9" w="6.9" e="7.0"/><has-kv
k="amenity" v="pub"/></query><print/>
```

さて、インポートで注目したいのは入力形式に bzip2 を使っていることです。init_osm3s.sh は入力に bzip2 形式を要求しますが、内部的には bunzip2 コマンドをリダイレクトしているだけです。そのため、ちょっと細工をすれば 1.5TB の非圧縮データを入力として入れることが可能となります。

というわけで実験をしてみたのですが、結果としては特に差は出ませんでした (正確な時間は停電で消えてしまいましたが…)。これはインポートプロセスが単一のプロセスしか利用しないという問題があり、非圧縮のデータを入れようが SSD のスピードよりもプロセスのスピードが足かせとなるという動きをしました。なお、bzip2 も非圧縮データでも同じく 500MB/s ぐらいのスピードが出ました。

というわけで、今度はマルチプロセスでの検証をしてみます。

3 https://pgtune.leopard.in.ua

4 https://wiki.openstreetmap.org/wiki/Overpass_API/Installation

Imposm3 による PostgreSQL データベースの作成

OSM のデータは一般的に特殊なデータベースを作るか、PostgreSQL(実際にはその拡張の PostGIS) 上にデータを入れることで様々な仕組みと連携が可能になります。imposm3[5] は PostgreSQL のデータベースを作成する仕組みでマルチプロセスで動作をします。ただし、入力形式が Protocol Buffer 形式に限られるという仕組みなので注意が必要です。では imposm3 で Planet のインポートを行ってみます。まずはデータベースの作成を行います。

```
createdb -O osm imposm3_planet
psql -U osm imposm3_planet
create extension postgis;
create extension hstore;
```

では、インポートを行います。

```
./imposm import -mapping mapping.json -connection   -read   -write   118021.70s
user 28366.90s system 387% cpu 10:29:22.71 total
```

10 時間 30 分ほどでインポートをすることができました。では検索をしてみましょう。

【検索例】

```
# select * from import.osm_amenities where name like '%新宿御苑%';
    id    |    osm_id    |              name              |       type       |
geometry
----------+--------------+--------------------------------+------------------+--------------
----------------------------------
 269070 | 1420771503 | 四谷消防署新宿御苑出張所 | fire_station | 0101000020110F0
00065D887240CAA6D41241AD81F123E5041
(1 行)
```

geometry のレコードには実際の座標が入っています。PostGIS の ST_ASTEXT 関数などで一般に認識できる座標データを見せることが可能です。

ここで、面白い結果が出てきました。実はこの前に日本の地域のみをインポートする実験を行ったのですが、そのときには最大でも 400MB/s ぐらいしかリードのスピードが出なかったのですが、世界単位になると 1600MB/s ぐらいのリードのスピードが出るようになります (図 1)。データの大きさによってインポートが早くなるという実験結果が得られました。これは思わぬ収穫でした。なお、インポート後に利用されるデータベースの容量は PostgreSQL で 350GB、imposm3 自体のキャッシュで 150GB ほど使います。まだまだストレージとして使える容量があるのでさらに実験をしていきます。

図 1: インポートのスピード

5 https://imposm.org/docs/imposm3/latest/

osm2pgrouting によるインポート

pgRouting は PostgreSQL 上で経路探索を行うプログラムです。pgRouting を使えば徒歩や車椅子、点字ブロックを使った目の見えない方向けの経路探索などが実装することができます。pgRouting には OSM のデータをインポートする専用の osm2pgrouting というソフトがあります。今回は時間が無かったので日本のデータをインポートします。

まず、最初に swap 領域を増やします。これは osm2pgrouting が大量にメモリを消費するため、16GB のメモリではすぐに kill してしまいます。日本全域では 512GB ほど swap を確保しましょう (大容量 SSD ならではの使い方ですね !)。

というわけで実験してみたのですが…osm2pgrouting のバグなのか正確なルーティングができないデータベースになってしまいました。

そのため、代わりに関東のみの領域をインポートしてみます。

```
./osm2pgrouting -d pgrouting_kanto -U osm -W osm -c  -f    603.93s user 216.84s
system 22% cpu 1:00:40.31 total
```

一時間ほどで終わりました。これで検索をしてみます。QGIS であれば pgRouting Layer というプラグインを使うと比較的簡単に検索をすることができます。図 2 では地元から自宅で自転車を使った経路を表示しています。

なお、osm2pgrouting も単一プロセスなのですが、SSD の性能としては 200MB/s ほどの書き出しが行われてかなり高速なのがわかります。今回のように大量に swap を消費するケースでも大容量なら気軽に使えるのがとても良かったです。

今回いろいろ検証したものが皆さんの OSM のデータを扱う第一歩になれば幸いです。

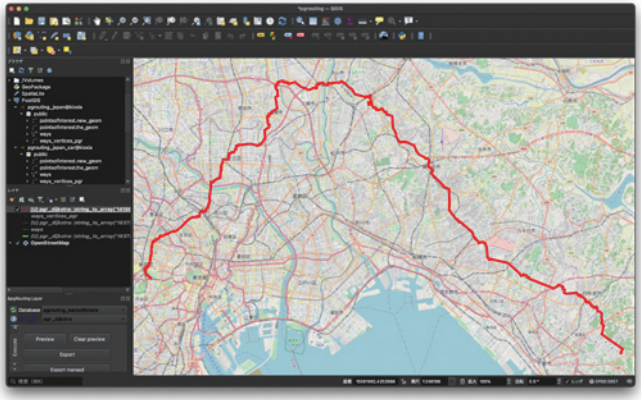


図 2: pgRouting で検索した経路

松澤 太郎 (@smellman)

一般社団法人オープンストリートマップファウンデーションジャパンで技術担当をしています。他にも日本 UNIX ユーザー会会長、一般社団法人 OSGeo 日本支部理事など、地図と UNIX を愛するエンジニアとして活動中。

Raspi は高速ストレージの夢を見るか

田中 瞳

PC にサーバー、IoT 機器。Maker たちの強い味方、Raspberry Pi を筆頭とする SBC(シングルボードコンピュータ)。 最近、SBC に SSD を接続する作例や、NVMe™ SSD を搭載可能な SBC が登場してきました。

SBC に NVMe™ SSD を搭載できるのか。また搭載することで、SSD の性能は発揮できるのか。どのような応用が効果的か。ベンチマーク結果を使い検討します。

NVMe™ SSD が「使える」SBC と、「活かせる」SBC

Raspberry Pi

SBC の代名詞ともいえる Raspberry Pi(Raspi) では、Raspi 4 で搭載する SoC が PCIe® 2.0x1 端子を装備しました[1]。

Raspi 4 Model B では、PCIe® 端子は USB3.0 ホストコントローラに接続され、4 口の USB 端子を搭載するのに使われています。Raspi Compute Module 4(CM4) では、CM4 - I/O ボード - PCIe®-NVMe™ 変換ボード、と接続することで、NVMe™ SSD を利用できます。ただし、搭載している PCIe® の世代が古く、レーン数も少ないため、現在の NVMe™ SSD(PCIe® 3.0or4.0 x4) のスペックを活かすことができません。

非 Raspi 系

非 Raspi 系の SBC 向け SoC には、複数レーンの PCIe® を搭載し、SSD の帯域を活かせるものがあります。Rockchip RK3399 はその一つで、PCIe® 2.1 x4 を搭載します。PCIe® の世代は古いものの、NVMe™ SSD の帯域をある程度カバーします。なお、RK3399 は、Gigabit Ethernet I/F(1Gbps) を備えていますが、PCIe® 2.1x4 の帯域は 4GB/s(32Gbps) です。他の SBC 向け PCIe® ・GbEther 搭載 SoC も同様の傾向ですが、PCIe® I/F の最大帯域よりも、ネットワーク I/F の帯域が小さいです。

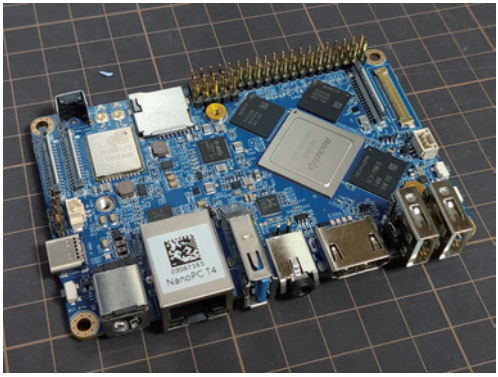


図 1: RK3399 搭載 SBC NanoPI-T4

1 https://datasheets.raspberrypi.org/ より BCM2711 datasheet で確認

RK3399 で、SBCxSSD の限界を探る

RK3399 を搭載する SBC で、NVMe™ SSD のベンチマークを行いました。使用した SBC は、FriendryElec「NanoPC-T4」です。比較用に、2018 年に構築した PC を用います。それぞれのスペックは以下の通りです。

表 1: ホストマシン 基本スペック

	NanoPC-T4	比較用 PC
搭載 SoC	RK3399	-
マザーボード	-	MSI Z370 PC PRO
CPU	Dual-Core Cortex®-A72(2.0GHz)+ Quad-Core Cortex®-A53(1.5GHz)	Core™ i5-8600K 3.6GHz (OverClock up to 4.6GHz)
RAM	LPDDR3 4GB	DDR4 PC4-21300 16GB
OS 用 ROM	16GB eMMC 5.1 Flash	PLEXTOR PX-1TM9PG+ (PCIe® 3 1TB NVMe™ SSD)
測定用 PCIe®	2.1 x 4 (NVMe™ key M)	3.0 x4 (NVMe™ key M)
寸法	100mm x 70mm	-

これら2つのホストマシンを使い、EXCERIA SSD 512GB に対して、以下の2つのテストを行いました。

1. **Kdiskmark、CrystalDiskMark を用いたディスク IO 速度テスト**
2. **chia を用いた Plot 速度テスト**

測定時の OS は、NanoPC-T4 では、Lubuntu16.04、比較用 PC では、ネイティブの Windows10 と、WSL 上で動作する Ubuntu 20.04 としました。

図 2: NanoPC-T4 に EXCERIA SSD を搭載

ベンチマーク結果

測定①：ディスク I/O 速度

CrystalDiskmark および、Linux™ 向けに同様の機能を持つ Kdiskmark を用いて、EXCERIA SSD のベンチマークを行った結果を示します。

テストケースは以下の 4 つです。

1. Sequential1MB,Queue8,Thread1
2. Sequential1MB,Queue1,Thread1
3. Random4kB,Queue32,Thread16
4. Random4kB,Queue1,Thread1

NanoPC-T4 における SSD の I/O 速度は、ワーストケースで 1/3 程度となるものの、ある程度 PCIe® の帯域を生かせていることがわかります。

一方、NanoPC-T4 では、SSD のシーケンシャル、ランダム I/O 速度が、搭載するネットワーク I/F（1GbE=125MB/s）よりも高速です。

ファイルサーバーやデータベースサーバーを NanoPC-T4 に構築した場合、ネットワーク入出力に応じて端末内でストレージ I/O 処理が頻繁に生じるため、実質的にはボトルネックにならないと考えられますが、特にシーケンシャルアクセスが優位のケースで、NanoPC-T4 は SSD の性能を活かしきれない可能性があります。

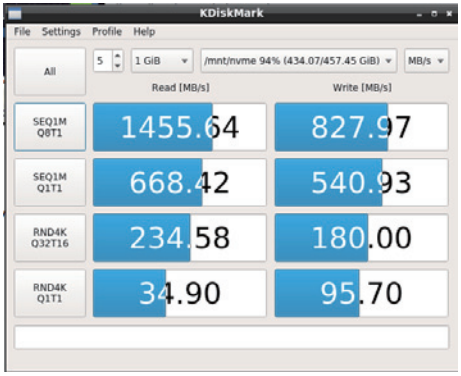


図 3: NanoPC-T4 SSD ベンチマーク結果

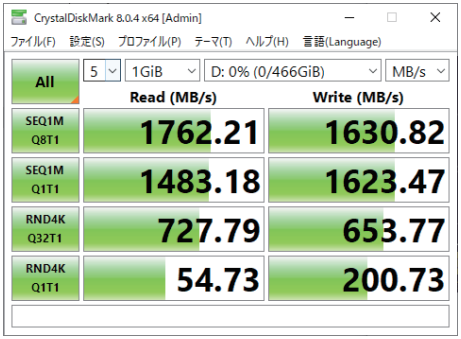


図 4: 比較用 PC (Win) ベンチマーク結果

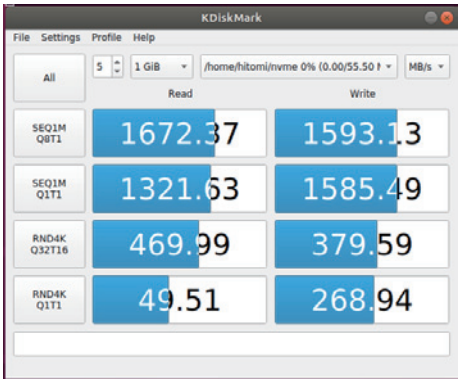


図 5: 比較用 PC（Ubuntu）ベンチマーク結果

測定②：chia Plot 速度

そこで、NanoPC-T4 が、ネットワーク速度に依存しない処理を行うケースを考えます。なるべくストレージを使用する処理として、昨今話題となっている、Proof of Space & Time 型暗号資産 chia の、Plot ファイル作成処理を、それぞれのホストマシンで実行しました。chia blockchain 1.1.2、k=25 プロットを行い、600MB のプロットファイルを生成するのにかかった時間は、以下の通りとなりました。

表 2: k=25 プロット所要時間

ホストマシン	OS	所要時間
NanoPC-T4	Lubuntu 16.04	734.534 seconds.
比較用 PC	Ubuntu 20.04(WSL)	100.886 seconds.
比較用 PC	Windows 10	115.242 seconds

残念ながら、NanoPC-T4 では、大幅に plot 作成時間が延びてしまいました。未特定ですが、原因として以下のようなものが考えられます。

- **DRAM が比較的少ないため、搭載する低速な eMMC にページングが生じた**
- **そもそも CPU が低速であり、ソート処理に時間がかかった**
- **Chia の plot 処理が Arm® コアと相性が悪い（ただし、数倍の差は考えにくい）**

さいごに

Maker 達の強い味方、SBC は、NVMe™ SSD にも対応しつつあります。十分な PCIe® 帯域を持つボードを用意し、搭載する CPU、ネットワーク帯域に見合ったユースケース、例えばファイル /DB サーバーならば、PC に比べ安価に構築ができるといえます。

おことわり

本記事中で示しているベンチマークスコアは、いかなる製品に対しても性能を保証するものでありません。あくまで参考データとしてご利用ください。

田中 瞳

うまくプロットできたら自宅で省電力ノード運用だ！と意気込んだものの、NanoPC のプロットは遅いし K=32 は完走せず。

これからもみんなでいろいろ試してみましょう。

- 記憶容量の定義：1MB(1 メガバイト) ＝ 1,000,000(10 の 6 乗) バイト、1GB ＝ 1 ギガバイトを 1,000,000,000(10 の 9 乗) バイト、1TB を 1,000,000,000,000(10 の 12 乗) バイトによる算出値です。但し、 1GB=1,073,741,824 (2 の 30 乗) バイトによる算出値をドライブ容量として用いるコンピューターオペレーティングシステムでは、記載よりも少ない容量がドライブ容量として表示されます。ドライブ容量は、ファイルサイズ、フォーマット、セッティング、ソフトウェア、オペレーティングシステムおよびその他の要因で変わります。使用可能なストレージ容量（さまざまなメディアファイルの例を含む）は、ファイルサイズ、フォーマット、設定、ソフトウェア、Microsoft オペレーティングシステムやプリインストールされたソフトウェアアプリケーションなどのオペレーティングシステム、またはメディアコンテンツによって異なります。実際のフォーマット済み容量は異なる場合があります。
- 画像は説明用です。実際の商品・サービスとはデザイン・仕様が一部異なる場合があります。
- Arm、Cortex、Mbed は、米国などで登録された Arm Limited（またはその関連会社）の商標です。
- Linux は、Linus Torvalds 氏の米国およびその他の国における登録商標または商標です。
- Raspberry Pi は、Raspberry Pi Foundation の商標です。
- Intel Core i5、Core i7 は、Intel Corporation またはその関連会社の商標です。
- Windows は、Microsoft Corporation の、米国およびその他の国における商標また は登録商標です。
- NVMe は、NVM Express, Inc. の米国またはその他の国における登録商標 または商標です。
- PCIe は PCI-SIG の商標です。
- その他記載されている社名・商品名・およびサービス名などは、それぞれの会社が商標として使用している場合があります。
- 製品の仕様、サービスの内容、お問い合わせ先などの情報は 2021 年 9 月時点の情報です。予告なしに変更される場合がありますので、あらかじめご了承ください。ここに含まれる技術およびアプリケーション情報は、最新の該当するキオクシア製品仕様が対象となります。

本同人誌に掲載の情報や内容については十分に注意を払っておりますが、その利用によって利用者にいかなる損害や被害が生じましても、一切の責任を負いません。必ず利用者ご自身の責任においてご利用ください。

本同人誌記載の部品性能は部品単体での性能であり、自作 SSD の動作、性能等を保証するものではありません。

■ SSD Doujinshi - SSD の同人誌

2021 年 10 月 2 日　第 1 版第 1 刷発行

著者 : ragnag / とだ勝之 / Pochio / 余熱 / 福屋 新吾 / 村口 / にちか /
藤澤 俊雄 / じむ / 松澤 太郎 / 田中 瞳
表紙イラスト : とだ勝之
表紙デザイン : 余熱
編集 : 余熱 / Pochio
発行 : キオクシア株式会社
連絡先 : kioxiahq-Exhibition-SSD@kioxia.com
印刷 : 株式会社 プリントパック

KIOXIA

キオクシア株式会社

〒 108-0023　東京都港区芝浦 3-1-21 田町ステーションタワー S

www.kioxia.com