FlashAir™ Doujinshi 6

FlashAirの同人誌6

TAKE FREE ￥0

フルスロットルで駆け抜けた6年間！
ありがとう FlashAir Developers！
FlashAirの歴史はまだまだ止まらない！

FlashAir
応援キャラクター
閃ソラ

閃ソラ（ひらめき そら）は、FlashAir の非公式応援キャラクターです。
とある航空会社の CA をしています。フライトでたびたび不在にしますが、
オフにはミラーレス一眼で写真を撮ったり、電子工作したり、
アプリ開発したり、忙しい毎日を送っています。

■プロフィール

名前	閃ソラ（ひらめき そら）
年齢	23 歳
職業	新人キャビンアテンダント
趣味	電子工作、アプリ開発
Twitter	@Hirameki_Sora

ありがとう、FlashAir Developers ！

高田

とても残念なお知らせをお伝えしなければなりません。2019 年 9 月 17 日をもちまして FlashAir Developers を閉鎖することになりました。FlashAir 同人誌もこの 6 号をもって最終号となります。FlashAir Developers ユーザの皆様には 2013 年 8 月の公開以来、6 年にわたり FlashAir の「普通でない使い方」を試し、Developers のコミュニティを盛り上げていただき、本当にありがとうございました。

FlashAir Developers はカメラ以外で FlashAir を使っていただく用途を増やすことを目的に開設しました。IoT 市場の出現にあわせて、FlashAir の技術情報をできる限り公開し、ユーザの皆様がお持ちのアイデアを自由に試していただけるようにと運営してきました。その結果、FlashAir Developers サミットには多くの参加者をお迎えして盛り上がりました。企業を支えている一人一人のエンジニアが個人の顔で参加してくれたのが Developers コミュニティでした。そんな FlashAir Developers に携わるのは、やりがいのある楽しい時間でした。「楽しい」という思いが新しいものを作り出すと信じて続けてきました。

その FlashAir Developers を閉鎖するのは正直つらいです。本気で取り組まれた方には申し訳ない気持ちでいっぱいです。また、Developers を通じてつながった仲間を失う寂しさもあります。ビジネスを超えたコミュニティの熱量が私たちの原動力でした。

FlashAir Developers、エンジニアの行動力とパワーを教えてくれてありがとう。貴重な経験と思い出、そして仲間をくれてありがとう。

東芝メモリ株式会社は 10 月 1 日から「キオクシア株式会社」に変わります。記憶で人々の暮らしに新しい価値をもたらしたいという願いが込められている社名です。社名は変わりますが、FlashAir Developers の経験は引き継ぎ、今後の取り組みにつなげてまいります。

FlashAir Developers に携わった皆様の未来が輝かしく広がることを願ってやみません。新社名のイベントで、またお目にかかれる日を楽しみにしております。

高田 (@M_Takada)

展示会やイベントなど FlashAir Developers のプロモーションを担当。5 月から会社全体の技術プレゼンス向上の仕事につきました。バブル世代の山女。

FlashAir は超ミニマイコン！

撮ったらシェア！

カメラに入れて使うことで
写真を撮ったらその場で
友人とシェアできる！

web サーバ機能搭載！

会議で資料を共有できる！
マイコンと繋げて
データロガーとしても使える！

GPIO が使える！

FlashAir の信号端子を自由に使える！
L チカからゲームまで
用途は多彩！

Lua スクリプトを実行可能！

FlashAir がマイコンのように使える！
メールを送信したり
ツイッターへの投稿が簡単にできる！

v4.00.03 新 API

fa.serial **fa.pwm**

FlashAir で音楽を鳴らせる！
サーマルプリンタを動かせる！

v4.00.04 新 API

fa.udp

FlashAir で UDP 通信ができる！

Contents

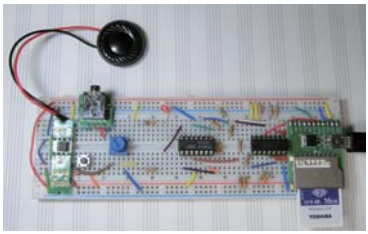
FlashAir 同人誌 6 みどころ紹介

FlashAir の歩み

ありがとう、FlashAir Developers！
FlashAir Developers was here
大事なことはすべて Developers が教えてくれた
Maker Faire Tokyo の 6 年間の戦い
「マイクロ SD でも負けません」の巻

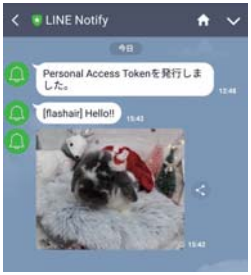
新 API 作例

帰ってきた FlashAir 活用テクニック
fa.pwm を使ってみよう
FlashAir が音源チップに？PWM で矩形波を演奏！
fa.serial でサーマルプリンタ
fa.udp で遊ぼう



FlashAir で作ってみた！

FlashAir ブラウザアプリの Android-9 対応
ファイルシステム不整合を回避する USB-SD リーダの製作
FlashAir + LINE で簡単に写真をシェアしよう !!
FlashAir と Azure IoT Central で簡単 IoT
FlashAir で写真をリアルタイムに表示する
世界初？ FlashAir 羽ばたき飛行機！
FlashAir の電波強度レベルを確認しよう
FlashAir で FPGA を操れ！
Airio-Base で Mbed OS 5
Mbed de iSDIO



思い出を語る

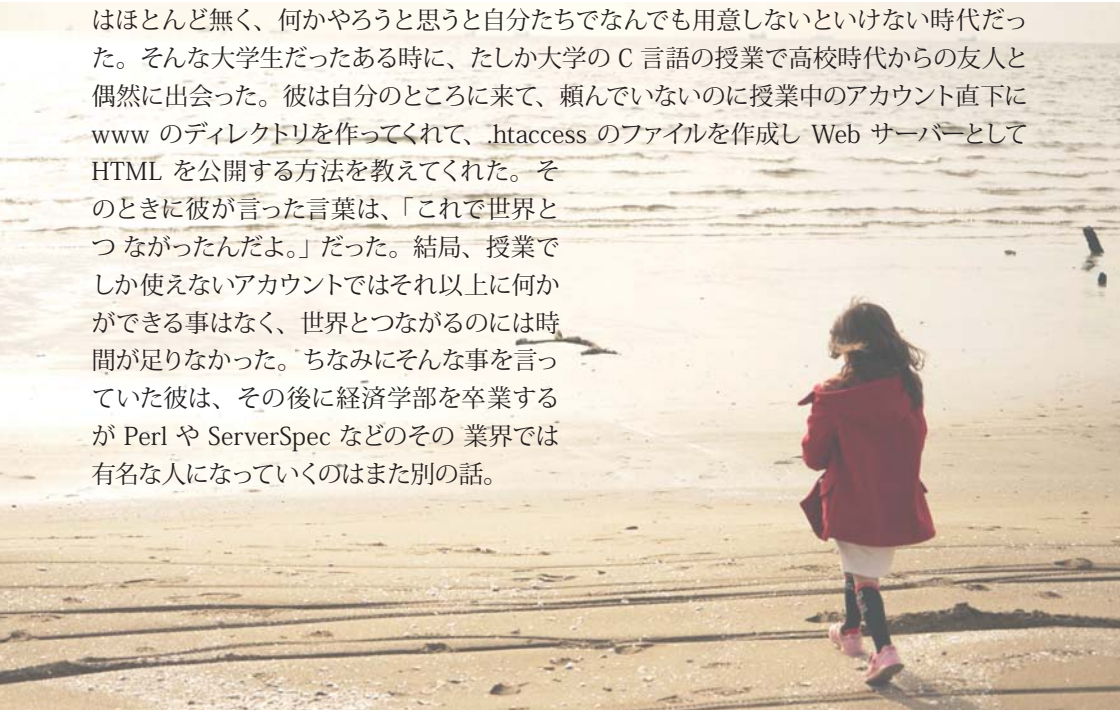
FlashAir の進化（AirLapse 編）
FlashAir と Lua と自作のツールと
煌めく空との九年史（ノンフィクションとフィクションの間に）
悲しみは思い出とともに
FlashAir 同人誌の長い編集後記

FlashAir Developers was here

伊藤 晋朗

我が家には 10 才の女の子がいる。学年で言えば小学５年生だ。最近になってその小学校の近くに分譲住宅用の土地が２箇所もできて、たくさんの一戸建てが建てられている。そのため、入学当初は１１人しかいなかったクラスだが、あっという間にクラスの人数は２４人になった。一年生の頃は女の子が４人しかいなくてうちの子がリレーの選手になっていたけども、今では用具係担当である。本人に聞くと用具係は運動会を運営しているようでやっていて楽しいようだ。そんな変な娘が１才だった頃にシンガポールに出張した。初めて訪れたシンガポールは思ったよりも暑くはなかったが昼の１２時に自分の影が真下にあるのをみて、「おお、赤道みたいじゃないか。」と思ったのは覚えている。今にして思えば、おそらくこの出張が本格的に無線 LAN 搭載 SD カードである FlashAir と付き合いはじめたのだと思う。ちなみに似たような物を作っている会社は世界中には複数あって、そのうちシンガポールとアメリカの西海岸の会社のエンジニアのリーダーには会ったことがある。勝手な思いで申し訳ないのだが、彼らとはこの手の製品特有の苦労やジレンマを共感できる気がして、どこかでつながっているような親近感を抱いていたりもする。

FlashAir の最初の世代の CPU クロックは 160MHz だったが、自分が大学時代に最初に買った PC も同じぐらいのスペックだった。そんな大学生の当時はインターネットの最初の頃で、世の中の Web サイトもまだまだな時代で、無料で使えるサービスなんてものはほとんど無く、何かやろうと思うと自分たちでなんでも用意しないといけない時代だった。そんな大学生だったある時に、たしか大学の C 言語の授業で高校時代からの友人と偶然に出会った。彼は自分のところに来て、頼んでいないのに授業中のアカウント直下に www のディレクトリを作ってくれて、.htaccess のファイルを作成し Web サーバーとして HTML を公開する方法を教えてくれた。そのときに彼が言った言葉は、「これで世界とつ ながったんだよ。」だった。結局、授業でしか使えないアカウントではそれ以上に何かができる事はなく、世界とつながるのには時間が足りなかった。ちなみにそんな事を言っていた彼は、その後に経済学部を卒業するが Perl や ServerSpec などのその 業界では有名な人になっていくのはまた別の話。



そんな出来事が、その後に自分でもサイトを作りたいと思うきっかけをくれた。そして、大学の研究室で Web サーバーを立ち上げて、その後に勝手に当時所属していたサークルの掲示板を作っていくのだが、やはり自分の子供と一緒で、「用具係」として運営側にいていろんな人が参加して場が盛り上がっていくのが楽しかったのかもしれない。それから数年後に大学院を卒業する時、引き継ぐ人もいなかったのでサイトは閉鎖することになった。当時はまだ十分に機能しており、後輩達が利用しているサイトを閉じるのには忍びなかったが仕方がなかった。サイトを閉じる報告のための最後のページを用意したところ、後輩が「記念に保存しときました！」と言ってくれたが Web を保存するのはなんとなくこそばゆい。

FlashAir を売り出して行くには FlashAir Developers のようなオープンな場を構築していく事になっていくのは、今思うとすごく自然な流れではあった。しかし、古い体質の会社にあっては、外部の会社組織の力を借りさせてもらうことでやっと実現したことでもある。その開発者サイトは、当初は色々な人達にスマホアプリケーションを作成してもらう事を目的としていたサイトだったのだが、その後に Lua の API なんかを掲載していくと、サイトに寄せられる質問はあるときからどんどん難しくなっていき、大変な思いをすることにもなった。しかし、FlashAir を気に入ってくれた人たちがフォーラムに書いてくれて、ユーザー間で問題が解決していく場面があると、こちらとしてはとても助かり、さらに仲間が増えていく事が嬉しかった。本当にありがとうございます。

Web の力は働いていないのだが、分譲住宅ラッシュのせいで友達が増えた娘のクラスでは、今度、女子だけで近くの市民プールに行く計画を立てているらしい。ちなみに誰かさんが作ったであろう、ちょっと漢字が間違っている計画書なんかも出来上がっている。「プールの後には、コンビニでアイスを買うの。」と娘に言われた。こっちはお金を用意すればいいだけだ。決して後をつけては行けない。まあ、そんな感じで遊ぶ事ができるのは彼女に仲間が増えていったからだ。そして、そのうち世界の誰かともつながって、もっと仲間を増やして面白いことができるようになっていくんだろう。

FlashAir がそうだったようにね。

伊藤 晋朗 (@ikainuk)
FlashAir の「中」の人。色々あって、細胞レベルで必死に自分を書き換えて対応している最中。

大事なことはすべて Developers が教えてくれた

土居

こんにちは！FlashAir Developers サイト（以下、Developers）の運営を担当しているフィックスターズの土居です。同人誌への登場は 2 号以来となります。Developers の中の人として、立ち上げ前から携わってきました。そんな Developers もとうとう終わりを迎えるということで、いろいろな思い出話を取り留めもなく語ってみようと思います。

FlashAir Developers が生まれたきっかけ

2012 年頃から、FlashAir に潜在する可能性を開花させるため東芝メモリ（以下、TMC）の FlashAir チームの皆様が何でもやってみようとアクションを取り始めていたそうです。その流れでフィックスターズにも話が来たのですが、他にも開発者コミュニティを運営した豊富な経験を持つ弊社蓮見とともに Developers の原型を提案しました。TMC としてはかなり尖がった活動に違いないものでしたが、FlashAir という製品を作っているチームの皆さんにとってはもう一つチャレンジが増えるだけ。TMC の皆様とともに API 公開とコミュニティによる自発的な知識の集積をイメージして構想を磨き、2013 年の春ごろから実際に動き出しました。

その後、サイト公開少し前の 2013 年 6 月には Pochio さんとの出会いがありコミュニティ活動力が一気に増強[1]。2014 年 10 月には余熱さんがジョインして基板や同人誌も出せる体制になりました。今は、この同人誌に寄稿してくださっている皆様もいます。いつも思うことですが、これだけのタレントが揃うというのは奇跡でしかないですね。

リファレンス 4 種＋チュートリアル 3 種を日英で書く！

あらためて振り返ると、Developers のコンテンツのうち、特にリファレンスやチュートリアルのような技術記事の 50% くらいは私が執筆あるいはドラフトをしていると思います。

Developers 立ち上げ時のコンテンツは今の半分くらいで、CGI/CONFIG/ ブラウザユーティリティのリファレンス、iOS/Android/Web ブラウザのチュートリアル、現在は「開発をはじめよう」にまとまっている FlashAir の開発向け面白機能を紹介するページ群、アプリショーケース、そしてフォーラム、という感じでした。また、オープン当初から現在まで一貫して続けてきたこだわりの一つに、原則として日本語と英語で提供するということがあります（途中で中国語にも対応しました）。

1　このとき Vine Linux の中の人& TeX 界隈で仕事をされているムネピさんという、私と同名の方とも出会いました。共通の知人がいてお互い名前は知っていたのですが、まさか FlashAir を通じて会えるとは！

図 1: 立ち上げ当初の Developers

記事のスタイルも試行錯誤しながらこれだけの量を揃えたというのは、よくやったと労いたいですが、実は秘密があります。ちょうどこのころ、私のチームで Anisha というアメリカ人女性をインターンとして受け入れていました。フィックスターズのインターンはみなプログラマーなのですが、チュートリアルを書いてもらいました。彼女は英語で書き、それを私が日本語に直す、という感じです。また逆に私が日本語で書いたチュートリアルを英訳もしてもらいました。彼女は日本語はわかりませんが、Google 翻訳で文意をつかみ、あとはネイティブの英語で清書する、といった感じで次々とこなしてくれました。彼女がインターンに来るという偶然がなかったら、立ち上げ時のコンテンツはもっとさみしくなっていた・・・かもしれません[2]（図1）。

動かない展示物！ ― Maker Faire Bay Area @2014 年 5 月

図 2: Maker Faire Bay Area の様子

この同人誌は 2014 年 11 月の Maker Faire Tokyo 出展以来、目玉として毎回配っているものですが、実は Maker Faire へはその前にも出展しています。しかも、本場 Bay Area！フィックスターズの米国子会社を通じて実現しました。

ストリートミュージシャンが、CD を売る代わりに首に下げたペンダントに内蔵された FlashAir の Web サーバーから音楽を配信する、というストーリー仕立ての展示計画です。展示員には、本物のミュージシャン（子会社社員ですがプロ活動している）をアサイン、さらに蓮見も解説に立ち当日の注目度はバッチリ！（図２）

そして FlashAir ペンダントの開発は私ということで、この時初めて独力で回路設計から基板製造、3D プリンタによる外装の制作まで行いました。外注したアートワークの検図でバグを見つけて冷や汗を書いたりしながらもなんとか進め、出来上がった基板に電池を入れてスイッチオン！すぐ止まります…。実は、ボタン電池では FlashAir が要求する負荷に耐えられずすぐに電圧降下してしまうためでした。よく考えれば当たり前ですね…。

2　Anisha、上部、土居の 3 名で技術記事は書きました。

焦りましたが、実は USB バッテリーバージョンというのも考えていて部品だけは買っていたので、無理やり改造して展示には何とか間に合わせたのでした（図３）。当日は無線が多くて接続性がイマイチだったのと、ペンダントが小さくてよくわからず、ただミュージシャンがチェロを演奏しているだけ、という感じになってしまったようです（笑）。

図 3: FlashAir ペンダント

iSDIO 規格書との闘い

iSDIO という、FlashAir をマイコンボードの拡張カードのように使える機能があります。これは、初代 FlashAir から搭載されていて、ある意味 FlashAir のアイデンティティーの一部ともいえる機能です 。しかし、Developers へのリファレンス掲載はかなり遅く、OSC 関西で Arduino チュートリアル開始を告知したのが、2014 年 8 月のことです。

なぜ遅れたのか。それは iSDIO の規格書の読解に私が手こずったからです！規格自体は十分な情報量でしたが、命令やメモリマップといった要素ごとの説明のため、この値をあのメモリに書き込んでからその命令を呼ぶ！というような手順を見出すのに大変苦労しました。理解が進まずずるずると時間が過ぎていたのですが、iSDIO API を公開しなければ Developers は完成と呼べない！と思っていたので、気持ちを奮い立たせて解析に取り組みました。そうしてようやく完成したのが、iSDIO API と Arduino チュートリアルです。世界でもっともわかりやすい iSDIO の解説であると自負しています（笑）。

マツダ SKYACTIV に涙した ー SWEST16@ 下呂温泉

iSDIO の勉強で必死だった頃、SWEST16 という組み込み系の学会イベントに参加するため FlashAir 生みの親の伊藤さんと一緒に下呂温泉に出張しました。ここでは、京都大学の高瀬先生らが作ってくださった、iRobot 社の研究目的向けロボット（ルンバから掃除機能を除いたもの）を FlashAir で制御するデモを中心に、展示と講演を行いました。実は、ホテルで伊藤さんに Arduino チュートリアルのデバッグに協力していただいた思い出も…。（出張は 8 月 28 日で、チュートリアル公開は 8 月…あれあれ？？）

さて、この時の基調講演で、マツダの人見光夫常務が Skyactiv という新世代エンジン

の開発秘話を語ってくれました。当時噂に聞いていた Skyactiv がどういうものか、そしてどうやって弱者が強者と闘うか、などが語られ、伊藤さんと一緒に涙を流して感銘を受けまくりました。それ以来、Skyactiv の記事を読むのが私の日課です。人見常務やマツダの泣ける苦労話はググればすぐに見つかるのでぜひご覧ください。

初めてのカプセルホテル ー DentalAir 導入事例の取材

ビジネス応用の事例の一つに株式会社ビーエム東洋様の DentalAir があります。Developers でご紹介できている中では、完成したソリューションとして実際にお客様に導入されている事例として貴重なものです。

図 4: DentalAir 開発者のお二人

2016 年 4 月に開院した歯科医院への導入事例を取材させていただいた（図 4 ）のですが、このとき私は兼務していたハードウェア製品のトラブルで大変な時でした。お客様へ謝りに行き対策を検討して報告、オフィスに戻れば開発チームと対策の実施。徹夜になる日も多くありました。この記事を書くためにメールを掘り返してみましたが…すぐに閉じました。

そんな中でしたので、取材を延期しようかと直前まで何度も葛藤したのですが、ビーエム東洋様が作ってくださったソリューションや受け入れてくださった歯科医院の皆様にもこちらが報いなくてはいけないと自分を奮い立たせ、前日に空いていた写真がきれいなホテルを予約して京都へ向かいました。京都への到着は 24 時近く。明日に向けて早く寝ようと綾小路通を歩くこと 10 分。到着したのはなんとカプセルホテル！初めての経験で、宿泊客もほとんど外国人。戸惑いましたが、設備は簡素ながら清潔で静粛でしたし、豊かな国際色で世界中を旅行する人ってこうなのかなー、などと結構楽しめました。

翌日、取材は和やかに終了したのですが、前述のトラブルが長引いたせいで記事の執筆に時間がかかってしまってご迷惑をおかけしたのが、申し訳なく思っています。

土居 (@munepi)

株式会社フィックスターズ在籍の FlashAir Developers の中の人。コミュニティ、同人誌、電子工作、ど根性…。大事なことはすべて Developers が教えてくれました。形を変えつつも、ここでの学びや出会いを生かしていきたいと思います。

Maker Faire Tokyo の 6 年間の戦い

Pochio

FlashAir 芸人の Pochio です。この原稿を執筆している今日は 2019 年 7 月 23 日でして、入稿直前のこの時期は FlashAir 同人誌の校閲作業に追われるのが恒例行事となっています。しかしご承知のとおりではありますが、残念ながら FlashAir Developers をこの 9 月に閉鎖することになりました。したがって今年でこの作業も最後になるわけです。そう思うと、大好きなお祭りがなくなるようで大変残念でなりません。

そんな FlashAir Developers にとって最大のイベントが Maker Faire Tokyo（MFT）への出展でした。毎年 5 月ごろにスポンサー出展の申し込みを行い、6 月に展示内容とおまけ基板企画について考え始め、7 月は同人誌制作で原稿集めから製本までを一気にやりきります。MFT には連続で 6 回出展させていただきました。これまでユーザーの皆様の独創的で様々な作品を多数展示してきました。一方、裏方の我々にとって MFT への出展は、同人誌とおまけ基板製作の戦い（？）の歴史でもあります。そこで今回が最後の FlashAir 同人誌になりますので、この 6 年間の製作物を振り返ってみたいと思います。

2014 年の初出展から始まった同人誌とおまけ基板製作

おまけ基板企画がなければ同人誌も生まれなかった

2014 年に販売されていた第 2 世代の FlashAir W-02 には GPIO 機能が搭載されていたのですが、現在秋月電子さんが販売されているような便利なブレイクアウト基板は存在しませんでした。そこで FlashAir 同人誌の表紙イラストを担当されているじむさんが、「GPIO 機能を活用しやすくする基板があれば便利だろう」、と考えたのです。初参加となる MFT2014 での配布を検討したのですが、基板の組立に失敗した人のクレーム対応が気になり、同人基板の製作に詳しい人にアドバイスを伺ってみることにしました。

図 1: 同人誌創刊号

図 2: GPIO 基板

この時じむさんが質問したのが余熱さんであり、しかも同じ会社の従業員であることが発覚したので、MFT にお手伝いいただくことになったのでした。その際、余熱さんが「同人誌を出しましょう」というので作ったのが、フルカラー 32 ページの FlashAir 同人誌創

刊号（図 1）だったわけです。この創刊号ですが、当初から「基本的に写真用途のネタは書かない」という明確なスタンスがありました。すなわちデジカメや写真に関するアプリケーションではなく、全く別の活用方法の紹介を目的としていたのです。とはいえこのころ販売されていた W-02 は、まだ Lua スクリプトに対応していませんでした。

さて先ほどの GPIO 基板（図 2）ですが、50 枚作って MFT 会場で販売する FlashAir のおまけにすることにしました。展示スペースは長机 2 つ分で、お隣は CPU を作っているかの有名な半導体メーカーでした。WBS の大江さんが取材に来られていましたね。我々のブースは入口近くのポジションをいただいたこともあってか、開場後 FlashAir 販売コーナーにあっという間に行列ができ、おまけ基板付きの FlashAir を 45 分で完売してしまいました。MFT 初参加で全く想定していない事態だったので、大変驚きました。

MFT2015 に向けた同人誌第 2 号と 2 種のおまけ基板の製作

翌年の同人誌第 2 号（図 3）では、新しい試みとして社外の FlashAir ユーザーを執筆者に迎えました。おなじみ綾瀬さんの鉄道模型記事はこの第 2 号が最初で、第 6 号までの 5 年間、休まず連載いただきました（感謝です！）。

また 2015 年 3 月には Lua スクリプトに対応した第 3 世代の FlashAir W-03 が発売され、4 月には「FlashAir ハッカソン」を開催。その概要と作品のレポートがこの第 2 号に掲載されました。そのほか MSX、某ホビーロボット、AppleII のフロッピー・エミュレータの話題など盛沢山でした。とはいえ、ページ数が倍増したため予算の関係で 2 色刷りとなってしまったのでした。

図 3: 同人誌 2 号

前年に大人気だったおまけ基板に味をしめ、この年は欲張ってスマホスタンド基板（図 4）と丸いコースター基板（図 5）の 2 種類を製作しました。スマホスタンド基板は、組立てると FlashAir 評価基板の「Airio」とほぼ同等になるのですが、丸い基板が珍しいのかコースター基板のほうが圧倒的な人気でした。このコースター基板を組立てるとラジコンカーを作れるのですが、部品の入手が難しく製作難易度も高めだったので、実際に組立てた方は少なかったのではないかと思います。

図 4: スマホスタンド基板

図 5: コースター基板

フルカラーの同人誌が復活した MFT2016

編集の負荷が増えるのを知りながらも懲りずにページ数と発行部数を増やしていく FlashAir 同人誌ですが、順調に第 3 号（図 6）を制作することになりました。前号は 2 色刷りでしたが、今回は印刷会社を変更してコストを抑えることに成功し、見事予算内でフルカラー印刷ができることになりました。

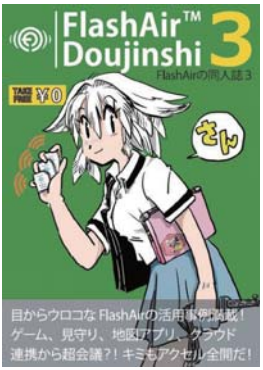


図 6: 同人誌 3 号

遠方で冊子版を入手するのが困難な方のために、FlashAir 同人誌は PDF でも公開しています。その PDF 版はカラーなのですが、冊子版を 2 色刷りにするとデータの加工が二度手間になってしまい、編集担当の負荷がとても増えてしまいます。そのため冊子版のカラー化は、製作コストの削減という観点でとても重要なのでした。

さて、第 2 号でじむさんが「FlashAir でゲーム？！」と題した漫画を掲載したのですが、FlashAir で JavaScript のゲームを動かすというネタが本当に持ちあがり、余熱さんがゲームコントローラ基板「Airio Play」を製作しました。これとほぼ同等の機能を持つコースター基板（図 7）が、この年のおまけ基板でした。

FlashAir の GPIO は 5 つの端子しかないので、ゲームのコントローラとして上下左右のボタンと AB ボタンの 6 つを用意するのは不可能に思えます。それで最初は AB ボタンのどちらかをあきらめようとか、色々議論したのを覚えています。とはいえやはり、AB の 2 つのボタンがないとゲームにならないと考え、やっと編み出した実現方法を秋月電子近くの某コーヒー店で余熱さんから聞いたのを覚えています。

肝心のゲームについては enchant.js という HTML5 でゲームが作れるライブラリがあり、便利なので活用したいと考えました。そこで 2016 年の 6 月ごろに FlashAir 用のゲームおよび MFT でのデモにこれを使うお許しを得るため、ライブラリを作られた会社を伺いました。その際ご相談した方が、某コミック誌で 30 年前に大活躍されていた有名模型メーカーの元社員の方で、お会いできて大変うれしかったのを覚えています。

このほか同人誌第 3 号には猫の見守り、地図アプリ、FlashAir IoT Hub、そしてアルコールガジェット TISPY のクラウドファンディングネタが掲載され、創刊当時では考えられないネタの広がりがありました。

図 7: ゲームコントローラになるコースター基板

突然訪れた試練

スポンサーから陥落した MFT2017

2017 年は原子力事業の損失により、我々 FlashAir 関係者が所属するメモリ事業部が分社化されることになりました。この損失起因で予算が削られ、MFT2017 のスポンサー出展が困難になりました。せっかく 3 年間続けてきて、言い過ぎかもですが同人誌が MFT 名物になりつつあった 4 年目の試練でした。しかしこの年を乗り切れば、来年はスポンサーに復帰できるかもしれない。そこで一般企業枠で参加させていただき、同人誌もこれまで通り発行して来年につなげようとしたのでした。

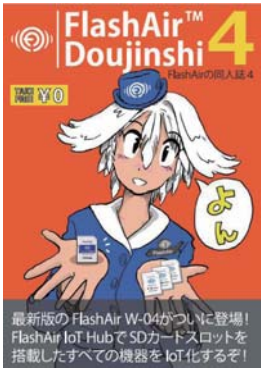


図 8: 同人誌 4 号

残念ながら予算がないのでおまけ基板を作ることはかなわず、昨年作ったゲームコントローラ基板の在庫を引き続きおまけにさせていただきました。一方、第 4 世代の FlashAir W-04 が発売されてから最初の同人誌となる第 4 号（図 8）は、表紙を含めて 72 ページになりました。20 本の記事の大部分で IoT ネタが充実し、FlahAir が IoT デバイスとして認識されるきっかけになれば、と期待を寄せる 1 冊になりました。

図 9: 同人誌 5 号

スポンサー復帰とおまけ基板復活の MFT2018

無事にスポンサーに復帰した 2018 年、記念すべき 5 冊目の FlashAir 同人誌（図 9）は表紙を含めて 80 ページになりました。IoT ネタはもちろん、時代を反映して（？）VR ネタが 2 本掲載される一方、昔のパソコンや家庭用ゲーム機に搭載された FM 音源ネタが 2 本、ペンプロッタへの応用や FlashAir の活用テクニックなど、総著者数 20 名、合計 22 本の記事が掲載され過去最大の盛り上がりを見せました。出展ブースも盛況で（図 10）、たくさん用意した同人誌は 2 日間で予定数を配りきってしまいました。

図 10: MFT2018 出展ブースの様子

一方おまけ基板の開発は、試行錯誤から抜け出すのに苦労しました。新しい試みとして子供向けにハンズオン企画をすることになり、子供が自分で何かを作って体験できるネタを考える必要がありました。そこで最初は某老舗花札メーカーのほにゃららラボを手本に、FlashAir

とおまけ基板と振動モーターでリモコンカーを実現できないか検討を重ねたのですが、実験できちんと動くものがなかなか作れず企画倒れとなったのでした。

このままでは展示に穴があいてしまいますから、急いで代わりのネタを考えなければなりません。そこで自分が小学生の頃に友達とどんな遊びをしていたか思い出すことにしました。漫画やゲームを持ってくることが禁じられているなか、自由帳にすごろくを描いて遊んでいたことを思い出しました。それも普通のすごろくではなく、山登りのすごろくでした（図 11）。特定のマスに止まると一気に登れたり、がけから落ちてしまったりしながら、一番早く頂上にたどり着くのを争うのです。この山登りすごろくをベースにすごろくのテンプレートを用意し、2 マス進むとか、3 マス戻るといったイベントを自由に考えてもらい、オリジナルのすごろくを作るハンズオンを思いついたのでした。

図 11: 山登りすごろく

そしておまけ基板は、すごろくのルーレットになる人気の丸型基板を企画しました。この同人誌の執筆陣の一人、さやさんのご協力で、5 つしかない GPIO 端子を駆使してやたらと機能が充実した FlashAir ルーレットができあがりました（図 12）。おまけ基板としては過去最高の完成度といえると思います。おかげさまですごろくハンズオンは、子供たちにとても好評でした（図 13）。

図 12: ルーレット基板

図 13: ハンズオンの様子

ちなみに余談ですが、2018 年の夏は同人誌 5 号と、コミケの新刊と、2018 年 9 月末に発刊されたまんが「フラッシュメモリのひみつ」の 3 冊の製作が重なり、なかなかハードな日々を過ごしていました。まんがの電子版が無料で読めますので、ぜひ「フラッシュメモリのひみつ」で検索してご覧いただければと思います。フラッシュメモリのことをよく知らない方にも、わかりやすく解説しています。なお、冊子版は全国の図書館にて借りることができますし、全国の小学校にも寄贈しています。

そして、MFT2019・・・

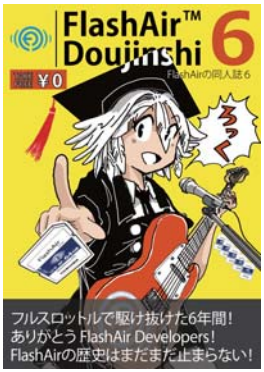


図 14: 同人誌 6 号

そんなこんなで迎えた 2019 年、FlashAir Developers 閉鎖の一報で、再び MFT 出展の危機が訪れました。閉鎖するのに出展はありなのか、相当に悩ましい状況でした。しかし、このまま出展をあきらめ、FlashAir 同人誌がいつのまにか終わってしまうのは寂しいかぎり。6 年間ご支援いただいた皆様に最後に感謝する機会を作りたく、FlashAir メインの展示ではありませんが、関係各位のご協力でどうにかこうにか出展できることになりました。

そして最後の同人誌として、この第 6 号（図 14）を発行できることは本当にありがたい次第です。急きょ過去最大のページ数を狙うことになり、ただ今 4 ページの原稿を 6 ページに増やしている真っただ中の午前 4 時です。果たして第 5 号の 80 ページを超えることができるのか？　もはや何と戦っているのかわからないと言われそうですが、無事に最終号を準備できそうです。なお、今年はおまけ基板がありませんが、フラッシュメモリの動作をモチーフにしたカードゲーム、そしてシリコンウェハを模したうちわなど、新しい試みで出展させていただく予定です。

最後に

6 年間の同人誌総発行部数を計算したところ、なんと 21000 部に上り、延べ 2 万人以上にお読みいただいたことになります（表 1）。これまでの執筆者と読者の皆さまに、この場を借りて厚く御礼申し上げます。FlashAir 同人誌が皆さまのお手元に渡るのは、今回が最後です。ぜひ最終号の感想を Twitter などでお聞かせください。

表 1: 同人誌の発行部数

No	発行部数
同人誌 1 号	3000 部 (1000 × 3 刷)
同人誌 2 号	3000 部 (1000+2000)
同人誌 3 号	3000 部
同人誌 4 号	4500 部
同人誌 5 号	4000 部
同人誌 6 号	3500 部 (予定)
合計	**21000 部**

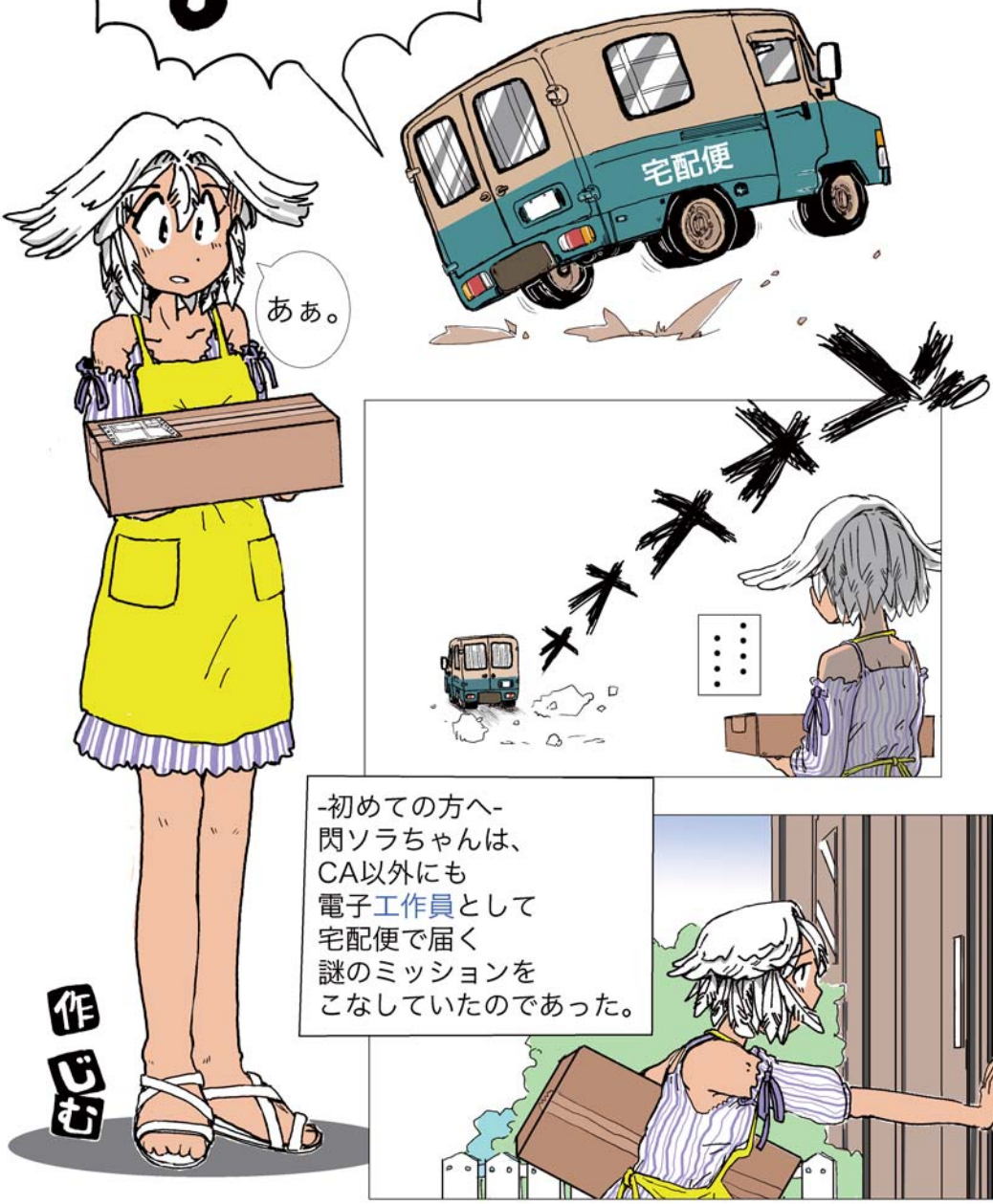
また私事ですが FlashAir 芸人としてハッカソンやイベントで、貴重な経験を積ませていただきました。残念ながら FlashAir Developers 閉鎖に伴い、FlashAir 芸人も引退になります。今後の身の処し方に悩みますが、また面白いことを発信したいと思います。これまで FlashAir 同人誌と当ブースの展示をご愛顧いただきまして、ありがとうございました！

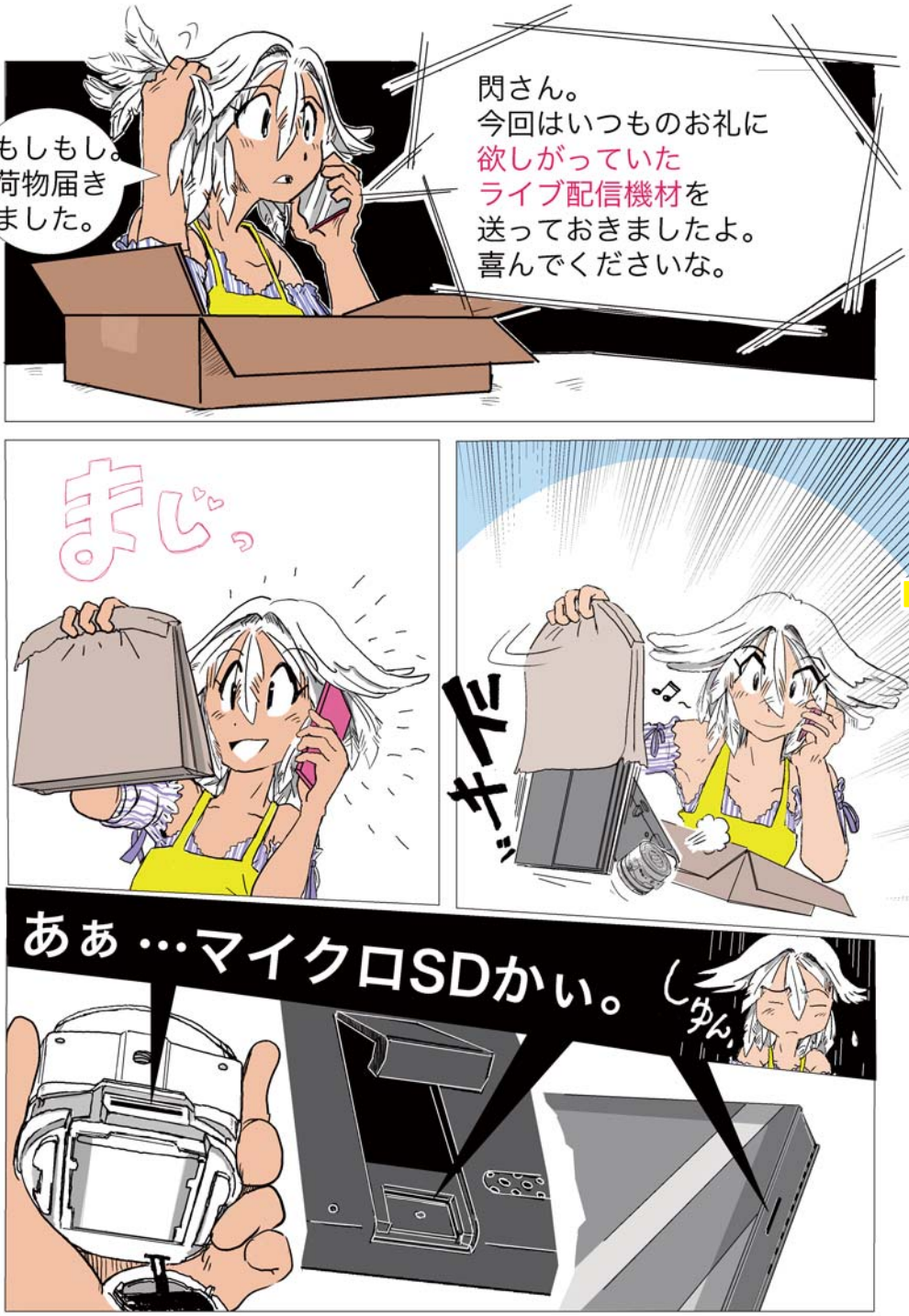
Pochio (@I_love_nintendo)

自称 FlashAir 芸人だが FlashAir Developers の閉鎖と同時に引退予定。昨年はまんが「フラッシュメモリのひみつ」を作っていました。これまでのご支援ありがとうございました。

まいど

「マイクロSDでも負けません」の巻

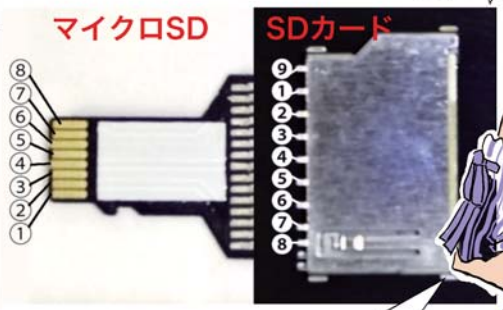



もしもし。
荷物届きました。
閃さん。
今回はいつものお礼に
欲しがっていた
ライブ配信機材を
送っておきましたよ。
喜んでくださいな。
まじっ
ドサッ
あぁ…マイクロSDかい。
しゅん。

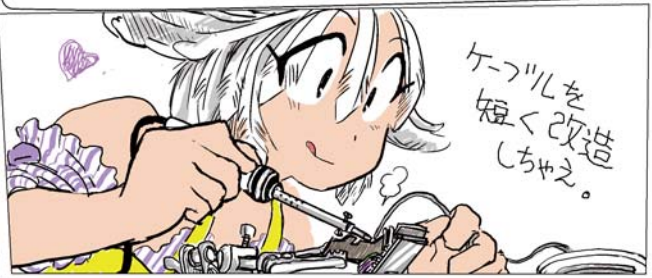
突然！ソラちゃん講座〜♥

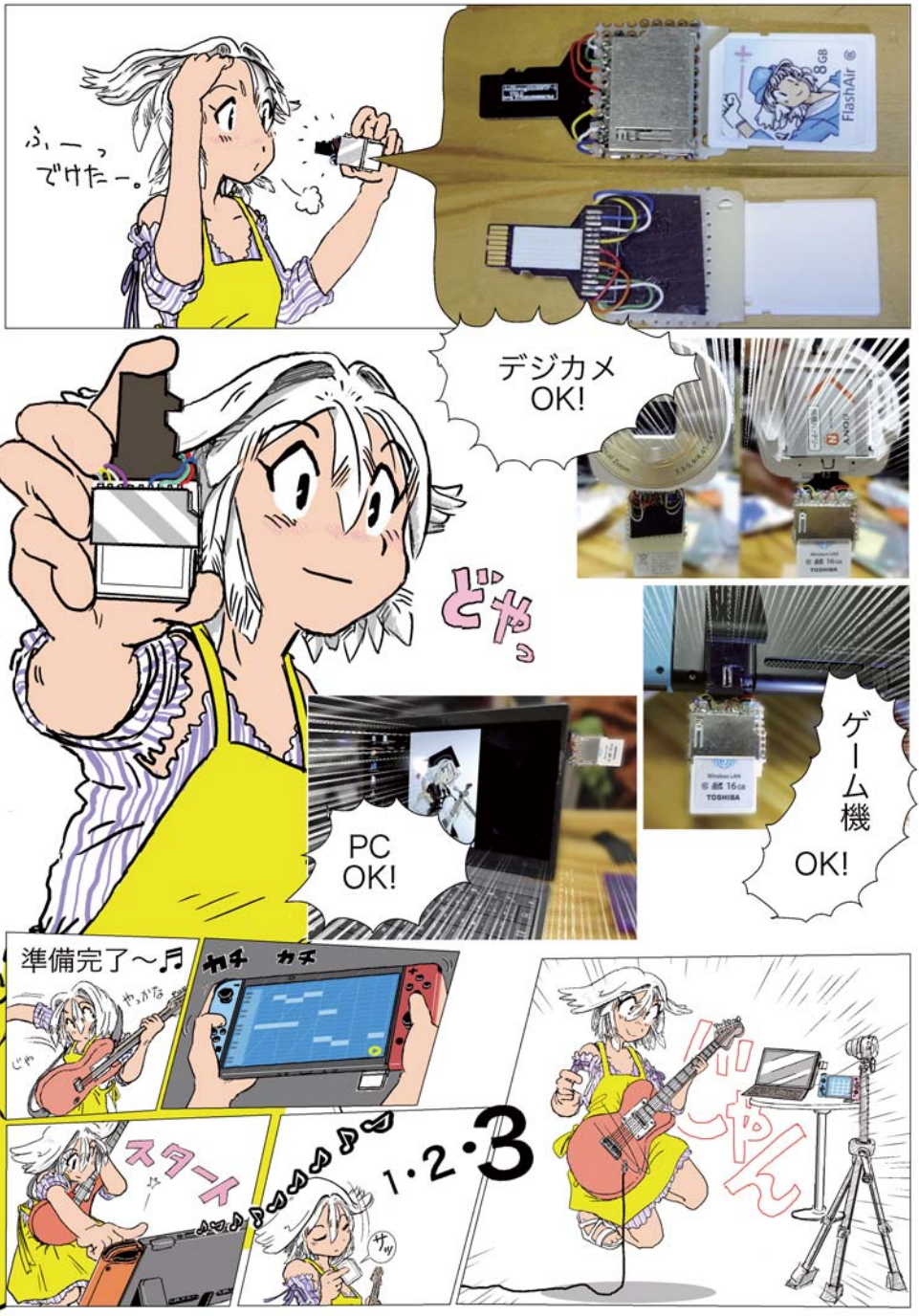
今回は**「マイクロSDの信号線について」**です。
マイクロSDの信号線は「8本」と、
普通のSDカードの「9本」と数が違います。

①	DAT2	⑨
⑧	DAT1	⑧
⑦	DAT0	⑦
⑥	VSS2	⑥
⑤	CLK	⑤
④	VDD	④
~~③~~	~~VSS1~~	~~③~~
③	CMD	②
②	CD/DAT3	①



その1本差も元々2本ある信号なので、互いの
信号線の意味・特性は同等となっています。
なので、それぞれを結線すると、
変換ができちゃいまーす。




ふぃーっ
できたー。
デジカメ
OK!
どやっ
PC
OK!
ゲーム機
OK!
準備完了～♬
やっかな
じゃ
カチ
カチ
スタート
1・2・3
ぎゃん

帰ってきた FlashAir 活用テクニック

あおいさや

FlashAir の内蔵マイコンで IoT 電子工作を楽しんでみませんか？

FlashAir W-04 の最新ファームウェア v4.00.03 の調査結果[1]を紹介するコーナーが帰ってきました。内容は無保証ですが（お約束）、モノづくりの参考になれば幸いです。

端子と Lua 関数の関係

W-04 の汎用端子は 5 本です。同時に使える機能はないか端子機能を調べてみました（表1）。"init" 欄は "init" を実行したときの端子の初期状態を、RSV はリザーブを表しています。濃い黄色セルは、2019 年 6 月に正式公開された関数 (fa.serial と fa.pwm) です。

表 1: 端子機能表

ピン番号	SD 規格			FlashAir オリジナル								
	SD インタフェース	SPI インタフェース	電源投入直後	fa.pio	fa.spi		fa.i2c		fa.serial		fa.pwm	
				GPIO	SPI	"init"	I^2C	"init"	UART	"init"	PWM	"init"
5	CLK	SCLK	I	RSV	RSV	I	RSV	I	RSV	I	RSV	I
2	CMD	DI	I	0x01	DO	O(L)	SCL	I	RX	I	0ch	L
7	DAT0	SO	I	0x02	CLK	O(*1)	SDA	I	TX	O(H)	1ch	L
8	DAT1	RSV	I	0x04	CS	O(H)	PIO	*4	PIO	*4 *5	2ch	L
9	DAT2	RSV	I	0x08	DI *2	I	TX *5	O(H)	RSV	*6	3ch	L
1	CD/DAT3	CS	I(pull-up)	0x10	RSV	*3	RX *5	I	RSV	*6	4ch	L
4	VCC (2.7 ～ 3.6V)											
3	VSS1											
6	VSS2											

*1: "mode" が 0 または 1 のとき "O(L)"、"mode" が 2 または 3 のとき "O(H)"

*2: I/O が "O" の時、"write" や "read" で "L" を出力

*3: fa.pio の設定を反映、事前に fa.pio を実行してない場合は "O(H)"

*4: "setpio" の設定を反映、事前に "setpio" を実行してない場合は "O(H)"

*5: 動作しているが未公開の機能 (2019/7/10 現在)

*6: fa.pio や fa.spi を実行後の初期値は "I", 事前に実行してない場合は "O(H)"

1　参考資料

- 簡易版 SD 規格仕様書 (https://www.sdcard.org/downloads/pls/index.html)
- Lua 関数リファレンス (https://www.flashair-developers.com/ja/documents/api/lua/)
- FlashAir の同人誌 (https://www.flashair-developers.com/ja/documents/books/)
- NXP Semiconductors UM10204「I^2C バス仕様およびユーザーマニュアル Rev. 5.0J」
- FlashAir W-04 v4.00.03 の挙動解析

前回までのあらすじ

下記方法については「FlashAir Doujinshi 5」をご参照ください。

- **fa.spi と fa.pio を切り替えながら使う方法**
- **fa.i2c と fa.serial と "setpio","getpio" を同時に使う方法**
- **fa.serial と "setpio","getpio" を同時に使う方法**

今回は I²C(fa.i2c) と SPI 出力 (fa.spi) を切り替えながら使う方法についてご紹介します。

fa.i2c と fa.spi を切り替えながら使う方法

SPI 対応の部品は、CS（チップセレクト）が High のときにクロックとデータを無視します。また、I²C 対応の部品は、スタート条件およびスレーブアドレスが一致しない限り入力信号を無視します。I²C バス仕様によれば、ファーストモード (400kbit/s) のスタート条件は SDA が High かつ SCL が High の状態が 1.3 μ秒以上続いた後に SDA が High→Low に変化し、SDA が Low かつ SCL が High の状態を 0.6 μ秒継続することです。1.6MHz の SPI 通信であれば、SDA と同じ端子を使う CLK の Low 期間または High 期間のいずれかが 0.32 μ秒未満となるので、正しく作られた I²C 部品ではスタート条件を満たしません。（そうでない部品は、切り替えながら使うことができません。あしからず）。

- **I²C に対応した部品が、ファーストモード (400kbit/s) までの対応でありファーストモードプラス (1Mbit/s) やハイスピードモード (3.4Mbit/s) には対応していないこと**
- **SPI 通信が、1MHz 以上であること**

が共存できる条件と考えています。

実験結果

下記部品をつないで音楽ゲームを作ったところ、期待通りに動かすことができました。

- **I²C 接続の NXP 製 PCF8574 を使った GPIO エキスパンダ**
- **SPI 接続の Maxim 製 MAX7219 を使った 8 x 8 dot 4in1 LED モジュール**

図 1: FlashAir 内蔵マイコンで動くゲーム

あおいさや (@La_zlo)
本職は SoC 開発。まだないものをつくるのが好き。
コードや作例を公開しましたら Twitter でお知らせします。

fa.pwm を使ってみよう

あおいさや

FlashAir Developers 閉鎖の発表と時を同じくして、２つの Lua 関数が公開されました。シリアル通信をサポートする fa.serial と PWM(Pulse Width Modulation) 出力をサポートする fa.pwm です。ここでは情報の少ない fa.pwm の使い方について紹介します。

FlashAir がサポートする PWM の特徴

fa.pwm は最大 5 チャンネル（0ch ～ 4ch）の繰り返しパルスを生成する関数です。W-04 のファームウェアバージョン v4.00.03 で使用可能です。流せる電流は非公開ですが、SD カードの DC 特性の条件である 2mA 程度までと考えるのがよさそうです。PWM は図 1 のような周波数 (freq) およびデューティ比 (duty) で出力され、fa.spi とは違い（内部の割り込み処理の影響を受けずに）常に均一な波形で出力されます。また出力したまま他の命令を実行できるので、好きなタイミングで出したり止めたりできます。止めないと Lua スクリプト終了後も繰り返しパルスを出力し続けます。PWM 機能を使用するには、他の端子機能と同様に CONFIG ファイルに IFMODE=1 を設定する必要があります。

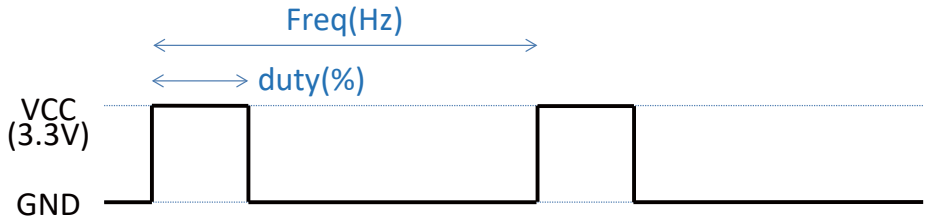


図 1: PWM 波形

fa.pwm の使い方

最初に PWM チャンネルを選択します。例えば、2ch と 4ch を使用する場合は、

```
res = fa.pwm("init", 2, 1)
res = fa.pwm("init", 4, 1)
```

と列挙します。戻り値 res が 1 となれば成功です。もし res が nil となる場合は、CONFIG に IFMODE=1 を設定していないことが原因と考えられます。なお、他の端子モードの状態で fa.pwm("init") 関数を実行すると、使用しないチャンネルも含めてすべての汎用端子が PWM 用に切り替わります。切り替え後の端子の初期状態は Low 出力です。

PWM を出力するときは、周波数とデューティを指定してから PWM をスタートします。デューティはパルスの周期に対する High 期間の割合です。

例えば、2ch に 440Hz でデューティ 30% の繰り返しパルスを出力するには、

```
res = fa.pwm("duty", 2, 440, 30)
res = fa.pwm("start", 2)
```

とします。

```
res = fa.pwm("stop", 2)
```

とすると指定したチャンネルを止めることができます。PWM は Low で止まります。

また下記のようにデューティを 100%にすると出力を High に固定することができます。

```
res = fa.pwm("duty", 2, 1000, 100)
```

PWM によるサーボモータ SG-90 の制御

マイクロサーボ SG-90 は、電子工作でよく使われる小型サーボモータです (図2)。このサーボモータは、20ms(50Hz) の連続パルスで制御でき、High パルスの幅を 0.5 ～ 2.4ms(互換品では値が異なることがあります) の範囲で変化させると、アームの角度が－90 度～＋ 90 度に変化します。そこで、このような関数で制御することができます。

```
function SG90(ch,degree) -- degree=-90..90
  local interval, min, max = 20, 0.5, 2.4  -- [ms]
  local freq = 1000/interval -- [Hz]
  local duty = ((degree+90)/180*(max-min)+min)/interval*100 -- [%]
  fa.pwm("duty",ch,freq,duty)
  fa.pwm("start",ch)
end
fa.pwm("init",4,1)
for i= -90,90,5 do
    SG90(4,i); sleep(200)
end
fa.pwm("stop",4)
```

図 2: マイクロサーボ SG-90

SG-90 の電圧仕様は 4.8V ～ 5V なので、FlashAir につなぐには電圧変換が必要ですが、手元の SG-90 を実験的に 3.3V でつないだところ動作が確認できました。

おわりに

本号では ながしまさんが fa.pwm で音楽再生する例を紹介されています。併せてご覧ください。また、筆者の GitHub（https://github.com/AoiSaya）でも fa.pwm を使用して 5 音同時再生に対応した MML 演奏ライブラリを公開予定です。さらに温湿度気圧センサ BME280 や各種 LCD モジュールを FlashAir で制御できるライブラリや日本語フォントを扱うライブラリも公開しています。皆さんのものづくりの お役に立てれば幸いです。

FlashAir が音源チップに？PWM で矩形波を演奏！

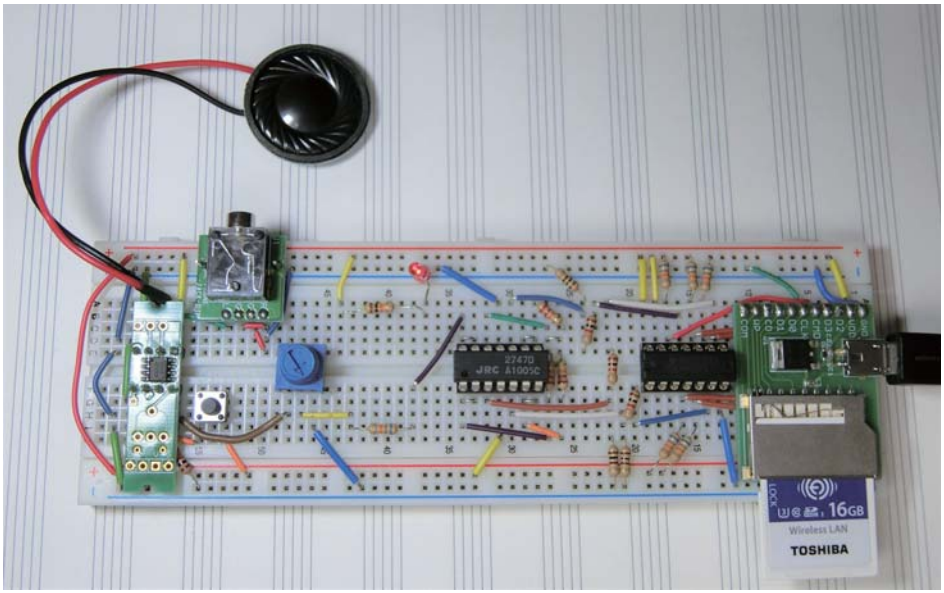
ながしま

FlashAir から音が鳴る！

Maker Faire Tokyo 2018 で東芝メモリのブースに立ち寄った際、FlashAir に YMF825Board や YM2413 を接続して演奏する展示[1]にテンションが上がりました。

一方、FlashAir W-04 ファームウェア v4.00.03 より PWM 出力機能である fa.pwm() がサポートされました。PWM はモーターの制御や LED の明るさ調整に使われますが、シンセサイザーやレトロゲーム機の音源としても使われています。これらの機器では PWM 出力の周波数とデューティー比が音程と音色に対応しています。

fa.pwm() でも同様に任意の周波数およびデューティ比を持つ矩形波を FlashAir のピンから出力させることができるわけです。これは FlashAir を音源化して演奏させるしかないでしょう。というわけで、本記事では FlashAir を使用して製作したチップ音源シーケンサーについてご紹介します。



1　これらの展示内容については FlashAir 同人誌 5 の『小さなゲームミュージックプレイヤー』（高野 修一さん）、および『FlashAir リズムマシン』（せいみ まさみさん）にて紹介されています。また、FlashAir の SPI 動作時のピン出力を使って圧電サウンダを鳴らす例が『FlashAir 活用テクニック－実践編－』（蒼 さやさん）で紹介されています。

PWM と AND ゲートで音量コントロール

FlashAir の PWM 出力をスピーカーに接続すると音を鳴らすことができます。しかし、出力電圧は Vcc, GND の 2 値ですから音量を変えることができません。これでは音楽的な表現に限りがあります。そこで、2 つのチャンネルを組み合わせて、ある出力チャンネルがもう片方の出力チャンネルの音量をコントロールできるようにします。これは AND ゲートにより実現できます。

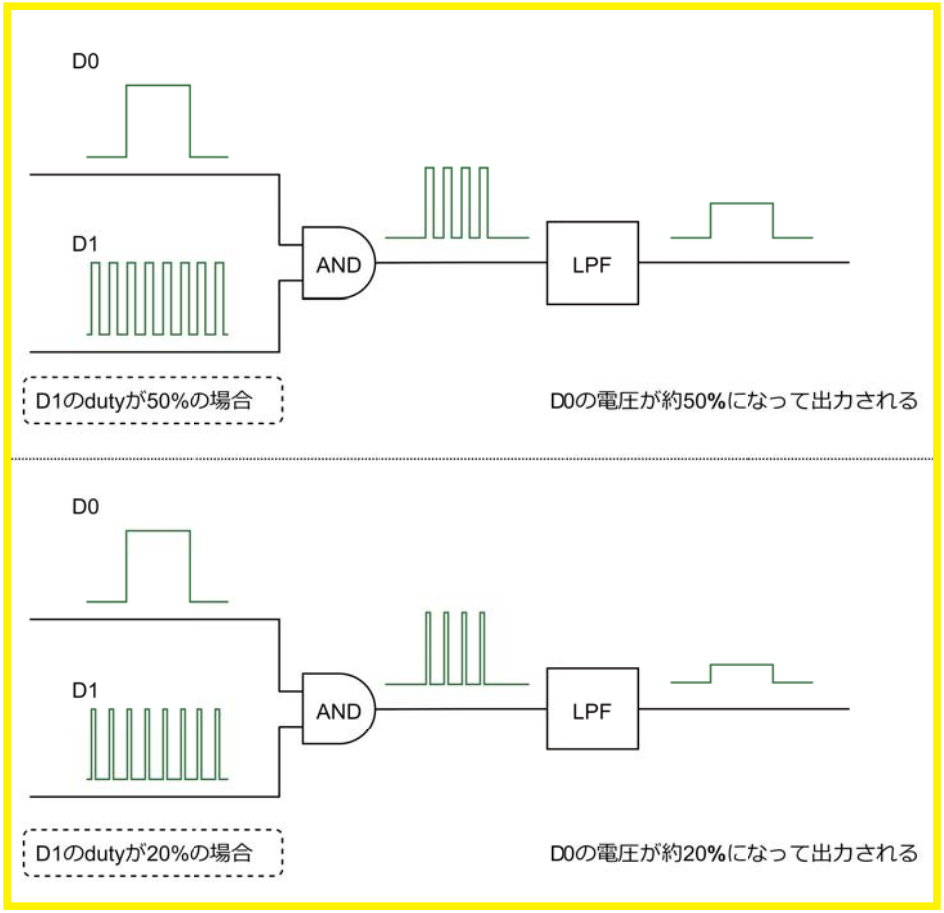


図 1: AND ゲートによる音量コントロール

FlashAir の D0 ピンから元の波形、D1 ピンに音量調整用の波形を出力します。D1 の周波数を D0 に対して高くし[2]、AND ゲートを通すことで D0 の波形が「刻まれ」ます。それをローパスフィルターに通すことで D0 の音量が変化したことになります。なお、実際には回路にローパスフィルターを入れなくてもスピーカーから音を発するまでの過程でフィルターがかかるようです。

ハードウェア

おおまかなブロック図を図 2 に掲載します。FlashAir の PWM は 5 チャンネルを出力できます。今回は 5 チャンネルを 2+2+1 の組み合わせとし、音量コントロール可能な矩形波を 2 つと音量固定の矩形波を 1 つ出力する構成としました。ファミコンや MSX のように、音程が出せるチャンネルを 3 つ作りたかったためです。

後段にミキサー回路、アンプを経由してスピーカーに接続しています。ミキサーは OP アンプを使っています。それぞれの矩形波をボルテージフォロワに通した後、非反転加算回路でミックスしています。

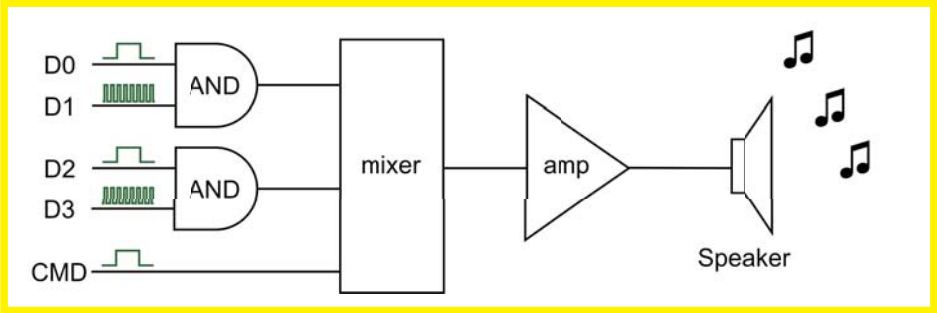


図 2: ハードウェアブロック図

ソフトウェア

os.clock() を使ってタイマー割り込みのような動作をさせる部分は FlashAir 同人誌 5 で紹介されているプログラムを参考にしています。1/60 秒ごとに処理を実行し、曲データの読み込みと発音済みの音に対するエンベロープをソフトウェアで実現しています。

曲データの形式は、Lua の string がバイナリを含められるという点を活かし、バイナリとテキストが混在した偽 MML のようなものを定義しました。曲データは手打ちです。

```
-- 偽 MML による「蛍の光」( 冒頭部分 )
"o¥x05cw¥x28¥x00Lfw¥x3c¥x00fw¥x14¥x00fw¥x28¥x00aw¥x28¥x00"..
"gw¥x3c¥x00fw¥x14¥x00gw¥x28¥x00aw¥x28¥x00"
```

2　実際には 400kHz に設定しています。

番外編 : PCM 再生

番外編として PCM 再生についてご紹介します。これは 1 チャンネルを用いて電圧を DAC のようにコントロールすることで実現できます。ただし、os.clock() の返り値が 1ms 単位のため、これを使ってタイミングを取るとサンプリング周波数の上限が 1kHz と荒くなってしまいます。そこで、CPU の速度に任せてハードループで再生することにします。速度が不安定のため再生中に音程が揺れますがそれも味ということで…。状況によって変わりますが、再生時のサンプリング周波数は 13 ～ 14kHz 程度になっていました。

```
-- CMD ピンから PCM 波形を出力 ; fa.pwm("init",0,1) は実行済みとする
local function wavplay()
    -- グローバル変数とテーブルルックアップを避けることによる高速化
    local fa_pwm, s_byte, s_len = fa.pwm, string.byte, string.len
    -- RAW 形式ファイルを使用 (Unsigned 8-bit PCM, 14kHz)
    local f = io.open("wav/sample.raw", "rb")
    local data = f:read(1*1024)
    local idx, val, s = 1, nil, nil
    fa_pwm("duty", 0, 400*1000, 0.0)
    fa_pwm("start", 0) -- チャンネル 0(CMD ピン ) を使用する
    while data do
        s = s_byte(data, idx) -- 1 サンプル取得
        val = (s / 255.0 * 100) -- 0 ～ 100 に調整
        idx = idx + 1
        if idx > s_len(data) then -- バッファを全て読みだしたら
            data = f:read(1*1024) -- 次を読む
            if not data then break end -- ファイルの最後で終了
            idx = 1
        end
        fa_pwm("duty", 0, 400*1000, val) -- 出力電圧に反映
    end
    fa_pwm("stop", 0)
    f:close()
end
```

おわりに

今回の製作物は Interop Tokyo 2019 の展示物および余熱さんによるセミナーの BGM として使用していただきました。3.5mm ミニジャックを接続してパソコンでライン録音したところ、なかなか良い音質で鳴っていました。ソースコードや動画を github で公開予定です (https://hngsm.github.io/fapwmplay/)。

ながしま (@_hngsm)

組込みソフトウェアエンジニア。FlashAir のおかげでブレッドボードや半田ごて、各種シングルボードコンピュータを触るようになった。

fa.serial を使って Pomera からプリンタに直接印刷

余熱

mbed 祭り 2016 @金沢 にてナダ電子さんからサーマルプリンタ AS-289R2 の紹介があったので購入してみました。シリアル通信で文字を送るとコンビニのレシートのような記録紙に印刷してくれる製品です。FlashAir にシリアル通信機能が付いたため直接駆動させることが可能です。

SD カード内のファイルを印刷するだけでは面白くないので、Pomera に入れた FlashAir からテキストファイルを取得し、自動的に印刷するシステムを作ろうと思います。DM100 は FlashAir に正式に対応[1]しており、無線 LAN 経由でテキストを読みだすことが可能です。そのため今回は Pomera 側の FlashAir を設定する必要はありません。システムを立ち上げておけば Pomera 側からの操作のみで印刷することが可能なため、会議の議事録などをその場で印刷して渡す という使用例が考えられます。

環境構築

配線

FlashAir のシリアル通信機能を使う際には CMD、DAT0 がぞれぞれ RX、TX になります。DAT0 を AS-289R2 の RX に接続し、それぞれの GND を接続すれば配線は完了です。詳細な説明は、あおいさやさんの記事を参照してください。

CONFIG ファイル

FlashAir 内の SD_WLAN/CONFIG に設定を書きます。Lua スクリプトを実行するために LUA_RUN_SCRIPT=test.lua と記述します。また、シリアル通信を行うために IFMODE=1 が必要です。Lua スクリプトを使って AP に接続を行うため、APPMODE などの設定変更は不要です。

Lua スクリプト

fa.Connect によって Pomera に接続し、fa.HTTPGetFile にてファイルダウンロードして fa.serial で印刷します。印刷後に fa.remove でファイルを削除しています。

1　後継機種の DM30 も FlashAir に正式対応しているため、同じように動作すると思われます。

```
fa.Disconnect()

while true do

	-- ポメラに接続
	while true do
		fa.pio(0x1E,0x1E)
		fa.Connect("flashair_pomera", "12345678")
		if(fa.WlanLink() == 1) then break end
		sleep(200)
		fa.pio(0x00,0x00)
		sleep(500)
	end

	-- ファイルダウンロード
	fa.pio(0x1E,0x1E)
	fa.HTTPGetFile("http://flashair/TEST.txt","TEST.txt")

	-- 印刷
	fa.serial("init",9600)
	fa.serial("write",27)
	fa.serial("write",36)
	fa.serial("write",01)
	local file = io.open("TEST.txt", "r" )
	for line in file:lines() do
		fa.serial("write",line)
		fa.serial("write","¥r¥n")
	end
	fa.serial("write","¥r¥n¥r¥n")

	-- ダウンロードしたファイルの削除
	fa.remove("TEST.txt")
	sleep(10000)

end
```

動作確認

あらかじめ FlashAir と接続した AS-289R2 の電源を立ち上げておきます。Pomera 側でメニューから「ファイル転送」を選ぶことで Pomera 内の FlashAir が AP として立ち上がり、自動的に印刷されます。Pomera と AS-289R2 を繋げる記事は以前にも執筆していますが[2]、ナダ電子さんにファームウェアをアップデートして頂いたおかげでシフト JIS がそのまま印刷できるようになりました。Ver2.0 以降の FW をお使いください。

2　サークル「空と月」発行の「兎と亀」参照

余熱 (@yone2_net)

2014 年には 1.8bps の GPIO で遊んでいたはずが、いつの間にか SPI、I²C、UART、PWM が搭載されてすっかり普通のマイコンになりましたね。最近はレトロ PC や 3D プリンタで活用している人も増えているとか。

fa.udp で遊ぼう

GPS_NMEA_JP (@Seg_faul)

2017 年 (!) の FlashAir 同人誌 4 では、UDP 送信機能について取り上げました。その際、「UDP 受信機能の存在は確認しているが動作させられなかった」と書きましたが、新ファームウェア v4.00.04 ではついに API が正式に搭載されました。受信機能も含めて動作するようになりましたので、本章では UDP の送受信機能について取り上げます。UDP 通信の応用例は Scratch、IP Messenger、Wake on LAN などが考えられます。詳細につきましては FlashAir 同人誌 4 をご参照ください。

UDP 送信機能について

UDP 送信機能では、文字列 (バイナリ含む) と、ファイル、共有メモリのデータの 3 種類を送信元として利用することができます。

省略形での送信

単に送信だけであれば、通常の関数呼び出しの形で書くことができます。

ブロードキャストでポート 4000 に対して Hello World を送信する場合

```
stat,msg = fa.udp("192.168.0.255",4000,"message","Hello World")
print(stat,msg)
```

192.168.0.2 のポート 4000 に対して、test.txt の内容を送信する場合

```
stat,msg = fa.udp("192.168.0.2",4000,"file","test.txt")
print(stat,msg)
```

テーブル形での送信

詳細なオプションを指定するには、引数をテーブルにして呼び出します。

192.168.0.2 のポート 4000 に対して、Hello World のうち Hello だけ送る場合

```
stat,msg = fa.udp({mode="send", address="192.168.0.2", port=4000,
 message="Hello World", size=5})
print(stat,msg)
```

192.168.0.2 のポート 4000 に対して、test.txt の内容を 3 バイト送信する場合

```
stat,msg = fa.udp({mode="send", address="192.168.0.2", port=4000,
 file="test.txt", size=3})
print(stat,msg)
```

共有メモリの送信

ファイル名に sharedmemory を指定すると共有メモリの内容を送信できます。その際、ファイル扱いとなり、送信サイズが指定できます。また送信開始位置として、オフセットの指定も可能です。

192.168.0.2 のポート 4000 に対して共有メモリ 0x01 ～ 0x03 の内容を送信する場合

```
stat,msg = fa.udp({mode="send", address="192.168.0.2", port=4000,
 file="sharedmemory", size=3, offet=1})
print(stat,msg)
```

定期送信機能

mode に send の代わりに send_interval を指定すると、自動的に定期送信する機能になります。ファイルの内容の変化を外部から監視したい場合などに使用できます。

送信間隔と送信終了時間が指定でき、省略した場合は 5 秒に 1 度無限に送り続けるようになります。この動作は非同期タスクとして実行されるため、Lua スクリプトが終了しても動作し続け、同じく非同期タスクとして実行される UDP 受信動作と共存はできません。

192.168.0.2 のポート 4000 に対して、hello.txt の内容を定期送信する場合

```
fa.udp({mode="stop"}) -- 前回実行された定期送信・受信をキャンセル
stat,msg = fa.udp({mode="send_interval", address="192.168.0.2", port=4000
,file="hello.txt", interval=1, timeout=180})
print(stat,msg)
```

定期送信が終了したか確認

現在の定期送信のタスクの状態をチェックするには state を使用します。

```
stat, cnt = fa.udp("state")
```

stat には実行中 (1) か否 (0) か、cnt には送信した総バイト数が代入されます。

定期送信の場合は、主に送信完了したかどうかをチェックすることができます。

UDP 受信機能

UDP 受信機能はファイルへの受信と、共有メモリへの受信の 2 種類が利用でき、受信動作は非同期で行われます。(定期送信と併用することはできません)

基本的に、指定サイズのデータを受信するような使い方が想定されているようですが、stat による確認を利用すると 1 バイトでも受信を検知したら受信途中で中止して内容を取り出すような使用方法も可能です。(受信の開始や停止には時間がかかるため、その間受信したデータは取りこぼします)

ファイルへの受信

ポート 4000 で待ち受け、hello.txt に受信する場合

```
fa.udp({mode="stop"})
stat,msg = fa.udp({mode="recv", port=4000,  file="/hello.txt",
size=100, timeout=30})
print(stat,msg)
```

注意として、ファイルを指定して受信する場合は、終了条件を満たすか、stop するまでは、ファイルに対して書き込まれません。(そのためリアルタイムにファイルをポーリングすることはできない)。このサンプルの場合、100 バイトあるいは 30 秒経ったら終了して反映されます。サイズを指定しない場合、空き領域の許す限り受信を続けると思われます。

共有メモリへの受信

ファイル名に sharedmemory を指定すると、代わりに共有メモリへ受信します。
こちらはリアルタイムに共有メモリに反映されます。また、offset の指定ができます。

ポート 4000 で待ち受け、共有メモリ 0x01 ～ 0x03 に内容を書き込む場合

```
fa.udp({mode="stop"})
stat,msg = fa.udp({mode="recv", port=4000,  file="sharedmemory",
size=3, offet=1})
print(stat,msg)
```

受信完了しているかを確認

現在の受信のタスクの状態をチェックするには state を使用します。

```
stat, cnt = fa.udp("state")
```

stat には実行中 (1) か否 (0) か、cnt には受信した総バイト数が代入されます。

サイズを指定して受信する場合に、受信が完了したかを state で知ることができ、また、共有メモリに対して受信している場合は cnt で受信データがあったかを知ることでリアルタイム処理ができるようになります。

受信の停止

受信を停止するには stop を使用します。この際、ファイルのクローズや受信位置のリセットが行われるようです。

```
fa.udp({mode="stop"})
```

連続受信サンプル

これらを利用し連続で受信しようとする場合、以下のような処理になります。

```
require "/FTLE/breakpoint"
port = 4000
size = 100 -- 確保したいバッファサイズ
-- 共有メモリを使用して連続受信する
function StartReceive()
   -- 受信位置を初期化（しないと 2 回目以降ズレるため）
   fa.udp({mode="stop"})
   -- 共有メモリを初期化
   fa.sharedmemory("write",1,size,string.rep(" ",size))
   -- 受信を開始
   fa.udp({mode="recv",port=port, file="sharedmemory",
      offset=1,size=size})
end
StartReceive() -- 初回の受信を開始
while true do
   breakpoint() --FTLE の Break ボタンで脱出する用
   collectgarbage()
   sleep(10)
   _,c = fa.udp("state") -- 受信バイト数を取得
   if(c > 0)then --1 バイトでも受信したら
      r = fa.sharedmemory("read",1,size) -- 共有メモリから読み出し
      print(r) -- 受信内容を表示
      StartReceive() -- 再度初期化
   end
end
```

なお、

```
require "/FTLE/breakpoint"
```

および

```
breakpoint()
```

は、拙作の FlashAir 向け Lua エディタ FlashTools Lua Editor を使用する場合に、Break ボタン押下時に無限ループから脱出させるための機能で、そうでない環境で使用する場合は削除してください。

GPS_NMEA_JP (@Seg_faul)
FlashTools Lua Editor は以下からダウンロードできます。
おまけ程度のツール置き場
https://sites.google.com/site/gpsnmeajp/tools/flashair_tiny_lua_editer

FlashAir ブラウザアプリの Android-9 対応

いしかわゆうじ

はじめまして。いしかわです。FairyQuick という Android で動く FlashAir ブラウザアプリを作っています（図1）。

本記事では、現行最新の Android 9.0 “Pie” に FlashAir 関連アプリを対応させる際の注意点について記します。最初はセキュリティって何それ状態だった Android も成熟が進んで、様々な形でアプリの挙動に制約が加わるようになりました。新しい API セットに新しい権限モデルが導入されることから始まり、ついに 9.0 で古い API の呼び出しにエラーが返るようになりました。ということで、何年も動かして枯れた処理でも安心できません。実際、私のケースでは、突然南米のユーザから修正要求のメールが来ました。FlashAir は遠く地球の裏側でオールドデジカメファンに愛されているようですよ。

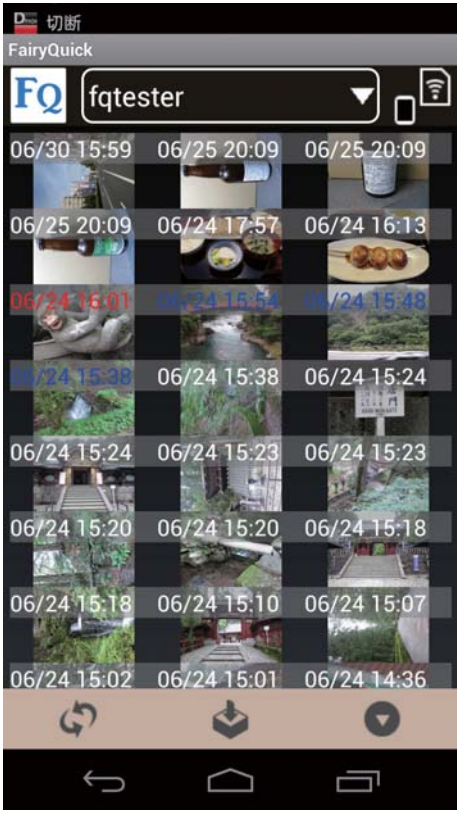


図 1: FairyQuick の画面

FairyQuick

FairyQuick の開発は FlashAir が発売されてすぐに始まり、2012 年 7 月に公開されました。当時からコンデジで撮った写真を SNS に投稿していたのですが、FlashAir は競合製品と違って必要なファイルだけ保存できるのが魅力でした。その一方で、FlashAir を操作している間はインターネットとの接続が切れてしまうことには若干の不満がありました。

そこで、FlashAir にぶらさがる時間を最小化することをコンセプトに FairyQuick の開発を始めました。サムネイルとファイル一覧をキャッシュしておいてオフラインで確認し、必要なファイルをダウンロードするときだけ FlashAir に接続する。これで SNS 投稿が捗りますね。

さて、前述の挙動を実現するためには、以下の技術要素が必要になります。

- **周辺の SSID の列挙 : 複数の FlashAir を使い分けるため。FlashAir が電波を出しているか確認するため**
- **Wi-Fi 接続先の切り替え : 4G 接続や自宅のルータから FlashAir に接続先を切り替えるため**
- **ファイルのダウンロード**

Android がバージョンを重ねるにつれて、これらの機能に関係する API に段階的に制約が入りました。しかし、互換性維持のために古い SDK でビルドされたアプリの挙動は維持されていました。そして時が流れて 2019 年。Android-9 が登場するにあたって、一斉に古い仕様に基づく動作が廃止されました。新端末でアプリが動かず、ユーザは悲鳴を上げます。開発者も遅れて悲鳴を上げます。だって日本は Android OS のアップデートが来るのが遅いのだもの。

以降、Android-9 への対応の要点を記していきます。

SSID の列挙：実行時にパーミッションを要求すべし！

Android 6.0 以降では、Wi-Fi の SSID を列挙するために、権限を実行時に取得する必要があります。以前はマニフェストに記述してインストール時に確認すれば十分だったのですが、プライバシーに関わる権限には追加対応が必要になりました。近傍の SSID を列挙すると GPS を使わずにユーザの現在地がわかるので仕方ないですね。

とはいえ、こちらも FlashAir を探し出す必要がありますので、さくっと修正を行います。関連する権限は ACCESS_COARSE_LOCATION です。サンプルコードの記述は最小限に留めましたが、位置情報を取るという要求に不快感を覚えるユーザもいるでしょうから事前に権限の利用目的を説明するダイアログを出したほうが良いでしょう。

```
/*in MainActivity.onCreate()*/
int perm = ContextCompat.checkSelfPermission(
  this, Manifest.permission.ACCESS_COARSE_LOCATION);
  if(perm != PackageManager.PERMISSION_GRANTED) {
    //TODO: shouldShowRequestPermissionRationale() で既に却下されたか確認
    //TODO: ダイアログを出して、権限を要求する理由を述べる
    //TODO: ファイルを保存する場合、WRITE_EXTERNAL_STORAGE も要求すべし
    ActivityCompat.requestPermissions(
      this, new String[] { Manifest.permission.ACCESS_COARSE_LOCATION },
      0 /* コールバックに渡る値（とりあえずなんでもいい）*/);
  }
```

参考資料 : https://developer.android.com/training/permissions/requesting

FlashAir への接続：アプリに Wi-Fi 接続を明示的にバインドすべし！

Android 5.0 あたりから、インターネットに繋がってない Wi-Fi の優先度が極端に下がりました。4G 回線があれば 4G 回線が優先され、Wi-Fi 経由の DNS は無視されます。ハニーポット対策やクラウドサービスの安定性は重要ですが、ルールはいささか過剰にも思えます。対策としては、bindProcessToNetwork() を呼び出すことでアプリのネットワーク接続を Wi-Fi 優先にすることができます。より細かい制御を望む場合にはソケット単位でネットワーク接続を切り替えられるようですが、そこまでする必要は無いでしょう。

なお、bindProcessToNetwork() を使えるのは、Android 6.0 以降に限られます。これでも 75% 程度の端末はカバーできますが、どうしても Android 5.0 以前をカバーしたい場合は、Build.VERSION.SDK_INT で場合分けして処理を記述しましょう。

```
private boolean bindToWifiNetwork() {
  // 既に FlashAir の SSID につないでいるとします。
  ConnectivityManager connman = (ConnectivityManager)
    MainActivity.this.getSystemService(Context.CONNECTIVITY_SERVICE);
  Network[] allnetwork = connman.getAllNetworks() ;
  for(Network network : allnetwork) {
    NetworkCapabilities cap = connman.getNetworkCapabilities(network);
    if(cap.hasTransport(NetworkCapabilities.TRANSPORT_WIFI)) {
      connman.bindProcessToNetwork(network);
      return true;
  }
}
return false;
}
```

参考資料 : https://developers-jp.googleblog.com/2016/08/wi-fi.html

HTTP アクセス : SSL 強制を回避するためにマニフェストを追加すべし！

Android 8.0 あたりから、ネットワーク経由の通信は SSL が必須になりました。このため、ファイルのダウンロードに https スキームを使わないと通信がブロックされます。しかし FlashAir は http しか喋りません。どうしようもないので、マニフェストに定義を追加して、特定のサイトだけ http アクセスが許されるようにします。

以下の設定ファイルを作成します。本来はアプリ単位でプライベートな証明書を扱うための仕組みのようですが、平文通信を許可することもできます。

```
<!-- res/xml/network_security_config.xml に保存する -->
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
<domain-config cleartextTrafficPermitted="true">
  <domain includeSubdomains="true">192.168.0.1</domain>
  <domain includeSubdomains="true">flashair</domain>
```

```
</domain-config>
</network-security-config>
```

そして、AndroidManifest を変更して設定ファイルを参照します。

```
<!-- AndroidManifest.xml に追記する -->
<manifest>
  <!-uses-permission などのタグがたくさんある -->
  <application
    // 他の属性がいろいろ書かれている。
    // 以下を追記する。
    android:networkSecurityConfig=" @xml/network_security_config" >
  </application>
</manifest>
```

参考資料 : https://developer.android.com/training/articles/security-config

まとめ

以上、FairyQuick の改修で得た知見を手短にまとめました。掲載したソースコードの全体はサンプルアプリという形で、https://sites.google.com/site/radikaiwarehouse/ に公開する予定です。安定して Wi-Fi の接続先を切り替える実装など、紙面の都合で書けなかったノウハウも入れます。

こうして整理すると、随分と簡単な修正に見えますが、デバッグはとてもしんどかったです。海外の製品展開が早いので私の手元には検証用の端末がありませんし、エミュレータでは Wi-Fi の切り替えが再現できません。そんな中で積もり積もった要修正事項が一斉に火を噴いたので、いくら直しても新しいトラブルが湧いてきます。報告をくれた南米のユーザと二人三脚状態で修正しました。こまめに最新 OS に対応すべきでしたね。まとめて改修ダメゼッタイ。

FlashAir Developers が無くなることで一次情報へのアクセスは悪くなるでしょうが、開発者もユーザもまだまだ元気です。これからもコミュニティから新しいアプリが、面白い応用が生まれてくることを期待しています。さて、そろそろ FairyQuick のリファクタリングに戻らねば・・・

いしかわゆうじ (@yuji_iskwrt)

FlashAir を買って、初めて大がかりなアプリを書きました。写真投稿が捗ってツイ廃になりました。凄腕の先輩方に出会いました。いつの間にか地球の裏側にユーザが居ました。同人誌に寄稿しました。全部 FlashAir から始まったことです。本当にありがとうございました。

ファイルシステム不整合を回避する USB-SD リーダの製作

めむ

FlashAir を PC 等のホスト機器に刺した状態で、FlashAir に対して無線経由でファイルをアップロードした場合、ホスト機器が認識しているファイルシステムと、実際の FlashAir 上のファイルシステムに不整合が生じます。このとき、ホスト機器は外部からアップロードされたファイルを認識していないので、例えばその後に FlashAir にファイル書き込みを行った際、外部からファイルアップロードされた領域に意図せず上書きしてしまう、なんてことが生じます。

しかし、「無線経由でファイルがアップロードされるたびに、ホストとの接続をリセットする」ような USB-SD カードリーダがあれば、ホストと無線の両方から、FlashAir を読み書きすることが可能になります。そんなリーダの需要はともかく、この記事では、そのような機能をもたせたカードリーダの構築方法を示します（図 1）。

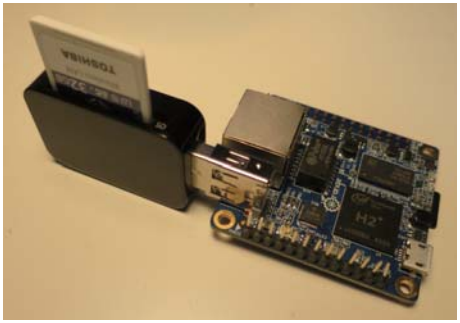


図 1: 構築した SD リーダ

そもそも、ファイルシステム不整合はなぜ起きるのか

単純にはホスト機器が、SD カード等のリムーバブルディスクを「占有的」に利用していることを前提として動作しているためです。FlashAir のように、ホスト機器のインタフェース以外にも、読み書きのインタフェースを持つ製品を、PC 等は想定していません。

このようなホスト機器では、読み書きの際のディスクアクセス数を削減するために、ディスク上のファイル・ディレクトリのインデックスである FAT 領域や、一部のディレクトリエントリ（ルートディレクトリとか）が格納されたセクタを、ホスト機器内にキャッシュします[1]。

このような状況で、FlashAir に無線経由でファイルがアップロードされたとき、FlashAir 内の FAT 領域は更新されても、ホスト機器内にキャッシュされた FAT 領域は更新されないため、ホスト機器はアップロードされたファイルやディレクトリ（厳密にはそれらが新しく占有したクラスタ）を認識できず、それらへ上書きを行い、ファイルやファイルシステムを壊してしまうわけです。

1　FAT 領域は、頻繁にアクセスする必要がある領域です、ここをキャッシュ化しないと、ファイルアクセスの途中等で頻繁なランダムアクセスが生じ、スループットが低下します。

SD リーダの構成

カードリーダの本体

今回作成する USB-SD カードリーダは、ベースにシングルボードコンピュータ (SBC) 使用します。SBC は、以下の条件を満たしている必要があります。

- **USB-OTG が利用可能な USB 端子を持つ**
- **上記と別に USB 端子か、システムディスク用以外の SD カードスロットを持つ**

この記事では、Orange Pi Zero 256MB[2] を利用します。例えば、Raspberry Pi Zero W だと、 1つ目の条件は満たしますが、 2つ目の条件を満たしません。

（Raspberry Pi Zero W では、USB-OTG と USB が1端子の排他利用になります）

材料の一覧

SD リーダを構築するのに必要な材料は、以下のとおりです。

- **シングルボードコンピュータ（SBC）Orange Pi Zero 256MB**
- **USB-SD リーダ 今回は ANKER AR200 を使用**
- **マイクロ SD カード SBC のシステムディスク用 8~32GB 程度 64GB 不可**
- **FlashAir（動作確認及びスクリプト・ソフトウェア転送用。なるべく 64GB 以外）**
  64GB の FlashAir しかない場合、別途 SD カード (32GB 以下) を用意。
- **USB- シリアル UART 変換器＆ケーブル（推奨）**

ソフトウェア構成

USB カードリーダ機能

Linux には、USB On-The-Go 機能[3] の一つに、g_mass_storage という、マスストレージデバイス動作を行わせるものがあり、ファイルやデバイスを USB 接続先の機器にマウントさせることができます。

カード更新の検知

FlashAir の更新検知には、FlashAir 内の FSInfo 構造体 (FAT32)、アロケーションテーブル (exFAT) を利用します。これらは、新しく書き込まれたクラスタ情報等を保持できるところで、ファイル書き込みと同時に必ず更

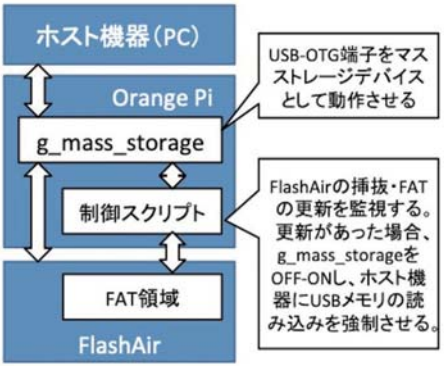


図 2: ブロック図

2 http://www.orangepi.org/orangepizero/

3 USB Device（子機）動作を行わせる機能のこと。

新されます。一方、ホストから SD への書き込みは、リーダ上で iostat を使って監視します。ホストから SD への書き込みがない（書き込みバイト数に変化なし）のに、FSInfo 等に更新があったとき、無線経由のカード更新と判定できます。

カードの更新が検知された場合は、先述の g_mass_storage 機能を無効→有効と切り替え、ホスト側に再度 FlashAir のファイルシステムを読み込ませます（図２）。

構築手順

1. Orange Pi に USB- シリアル UART 変換器、USB-SD リーダを接続します（図3）。あとで Networking サービスを無効化し、起動を高速化するため、シリアル経由でログインしています。（起動が数秒遅くなりますが、Networking 有効でも同じものは作れます）

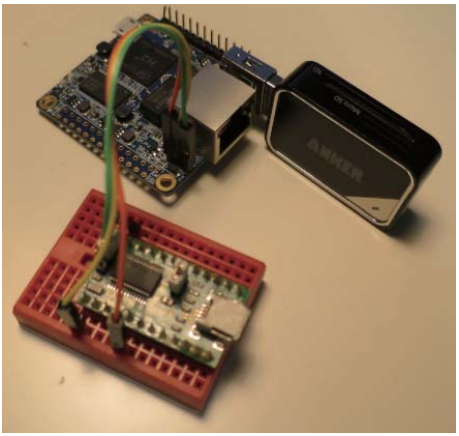


図 3: 構築中の SD リーダ

2. Micro SD カードに、Orange Pi 用 OS を書き込みます。使用した OS は Armbian Xenial (https://www.armbian.com/orange-pi-zero/) です。ページ中 "Other Download Options" から取得します。

3. 拙作のスクリプト (https://github.com/memukuge/StorageBridge4FA) をインストールします。スクリプトは、FlashAir や SD に入れ、Orange Pi に接続した USB-SD リーダから取り込みます。スクリプトの詳細は割愛しますが、おおむね1秒ごとに、カード内の更新および、カードの挿抜を監視し続け、いずれかのイベントを検知すると、ホストとの接続 (g_mass_storage) をリセット、および停止します。

4. /etc/modules を編集します。
- g_serial をコメントアウト（USB に紐付けられたシリアルを無効化）
- xradio_wlan をコメントアウト (Orange Pi の無線は技適不適合のため)

5. OrangePi の USB-OTG を PC 等に接続し、カードリーダに FlashAir を挿入し、主機能用スクリプトのうち、"initialscript.sh" を実行します。PC 上でカードが認識されたら、以下のことを確認します。
- FlashAir を抜き差しすると、マウントされ直すこと
- FlashAir に無線経由でアップロードしたときに、マウントされ直すこと

6. 起動時に自動でカードリーダ動作するよう、crontab に以下を追加します（Path は適宜変更してください）。

```
@reboot sh /root/StorageBridge4FA/initialsctipt.sh > /root/errorlog 2>%1
```

7. 以下のコマンドで不要なサービスを停止し、起動を (数秒程度) 高速化します。

```
# systemctl disable armbian-hardware-monitor.service
# systemctl disable armbian-ramlog.service
# systemctl disable networking # シリアル経由でメンテできる場合のみ
# systemctl disable keyboard-setup.service
```

性能・その他

構築した SD リーダのベンチマークを図 4 に図示します。性能面では、概ねシーケンシャル R/W で 10MB/s 程度出ます。

なお、この SD リーダでも、無線・ホスト書き込みがちょうど同時に行われた場合には、ファイルシステムの不整合を生じます。ネットワークからの書き込み時には、ホストがアクセスしていないタイミングを見計らって行ってください。

今回は、様々なホスト機器で利用できる USB-SD カードリーダの形で作成しましたが、この仕組を応用することで、Linux マシン上で、無線経由の書き込みと、マシンからの書き込みを両立させるアプリケーションも構築可能なはずです。

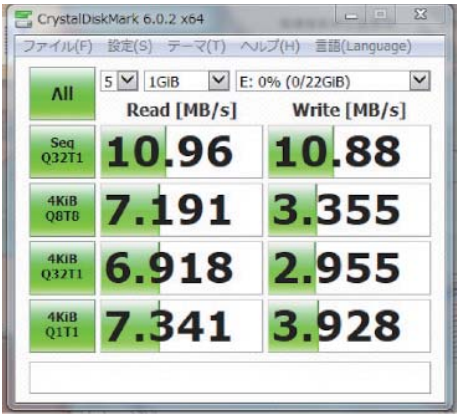


図 4: ベンチマーク

めむ (@memukuge)
いつもは好きなものをニコ動に上げたりする。

FlashAir + LINE で簡単に写真をシェアしよう!!

寺田 賢司

今回はより FlashAir のすごさをご理解頂くため、FlashAir と LINE の連携をご紹介したいと思います。FlashAir を使えば撮った写真を自動でメッセージをつけて LINE にアップすることができます。一度 LINE にアップしてしまえば LINE の機能を使ってシェアしたい方々を招待し、撮った写真を簡単に皆でシェアすることができるようになります。

LINE Notify の登録

皆様は「LINE Notify」というサービスをご存知でしょうか？今回行う FlashAir と Line の連携にはこの「LINE Notify」というサービスを使用します（図 1）。

図 1: LINE Notify

まず初めに LINE Notify[1] に登録を行います。右上にある「ログイン」をクリックしてログイン画面に遷移し、「ログイン」を行うと、先ほどの「ログイン」が LINE で使用している名前に変更されるかと思います。その名前をクリックし、「マイページ」をクリックすると「マイページ」画面に移動し、下部にある「トークンを発行する」ボタンをクリックしダイアログを表示します。

ここはお好みで構いませんが、今回は「トークン名を記入してください」には「flashair」、「通知を送信するトークルームを選択してください」は「1:1 で LINE Notify から通知を受け取る」を選択し、「発行する」をクリックしてください。すると「発行したトークンはこちらです。」と書かれたダイアログが表示されますので、このトークンを保管してから閉じてください、このトークンは今後使用する大事なキーワードとなります。

「LINE Notify」でのトークン発行が完了すると、「連携中サービス」に登録したサービスが表示されるとともに、LINE アプリに「LINE Notify」からのメッセージが送信されていることと思います（図 2）。

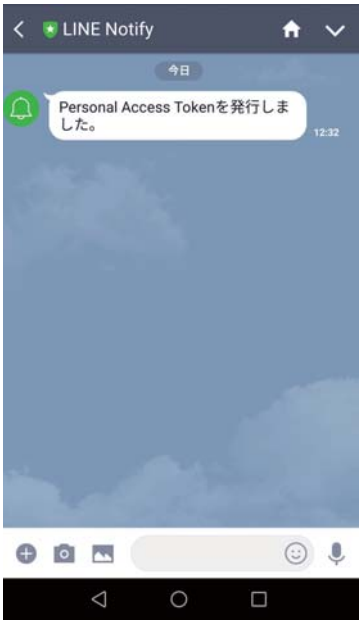


図 2: LINE Notify からのメッセージ

1 https://notify-bot.line.me/ja/

FlashAir の設定

まず初めに FlashAir を PC に接続し、FlashAir の中身を確認してください。(「表示」タブの「ファイル名拡張子」、「隠しファイル」にチェックを入れて表示してください。) そこに「line-notify.lua」というファイルと「images」というフォルダを作成します (図 3)。

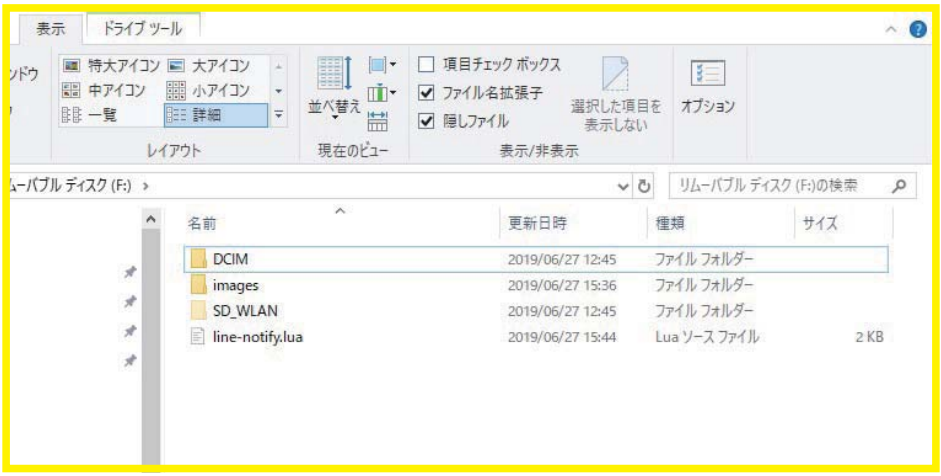


図 3: FlashAir の中身

次に「line-notify.lua」ファイルをテキストエディタで開き、下記のソースをペーストし、「local NOTIFY_TOKEN = ""」の "" 内に先ほど保管したトークンをペーストし保存します。(メッセージを変えたい方は「local MESSAGE = "Hello!!"」の「Hello!!」を変更してください。)

```
local NOTIFY_TOKEN = ""
local MESSAGE = "Hello!!"
local IMAGE_FOLDR = "/images"
local NOTIFY_URL = https://notify-api.line.me/api/notify

local function uploadFile(path, file)
    local filesize = lfs.attributes(path, "size")
    if filesize == nil then
        return
    end
    local boundary = "--------------------------908511f2c01b0981"
    local contenttype = "multipart/form-data; boundary=" .. boundary
    local mes = "--" .. boundary .. "¥r¥n" .. "Content-Disposition: form-data;
name=¥"message¥"¥r¥n" .. "¥r¥n" .. MESSAGE .. "¥r¥n" .. "--" .. boundary .. "¥
r¥n" .. "Content-Disposition: form-data; name=¥"imageFile¥"; filename=¥"" .. file
.. "¥"¥r¥n" .. "Content-Type: image/jpeg¥r¥n" .. "¥r¥n" .. "<!--WLANSDFILE-->¥
r¥n" .. "--" .. boundary .. "--"
    local blen = filesize + string.len(mes) - 17
    local b, c, h = fa.request {
        url = NOTIFY_URL,
```

```
        method = "POST",
        headers = {
            ["Authorization"] = " Bearer " .. NOTIFY_TOKEN,
            ["Content-Length"] = tostring(blen),
            ["Content-Type"] = contenttype
        },
        file = path,
        bufsize = 1460*10,
        body = mes
    }
end

local tgtPath = ""
local tgtFile = ""
local tgtFileMod = 0
local tgtDirMod = 0
local tgtDir = IMAGE_FOLDR

for dirname in lfs.dir(IMAGE_FOLDR) do
    local dirpath = IMAGE_FOLDR .. "/" .. dirname
    local tgtMode = lfs.attributes(dirpath, "mode")
    if tgtMode == "directory" then
        local dirMod = lfs.attributes(dirpath, "modification")
        if dirMod > tgtDirMod then
            tgtDirMod = dirMod
            tgtDir = dirpath
        end
    end
end

for filename in lfs.dir(tgtDir) do
    if(string.sub(filename, 1, 1) ~= ".") then
        local filepath = tgtDir .. "/" .. filename
        local mod = lfs.attributes(filepath, "modification")
        if mod > tgtFileMod then
            tgtFileMod = mod
            tgtPath = filepath
            tgtFile = filename
        end
    end
end
uploadFile(tgtPath, tgtPath)
```

次に CONFIG ファイルの修正を行います。

「SD_WLAN」フォルダの中にある「CONFIG」ファイルをテキストエディタで開き、下記のソースをペーストし、「< 接続先 SSID>」「< 接続先パスワード >」を接続するネットワーク環境（テザリング）に合わせて設定します（接続出来ない場合はステーションモードの利用[2]をご確認ください）。

2 https://flashair-developers.com/ja/documents/tutorials/advanced/1/

```
[Vendor]
CIPATH=/DCIM/100__TSB/FA000001.JPG
APPMODE=5
APPNAME=myflashair
APPSSID=< 接続先 SSID>
APPNETWORKKEY=< 接続先パスワード >
VERSION=F15DBW3BW4.00.03
CID=00000000000000000000000000000000
PRODUCT=FlashAir
VENDOR=TOSHIBA
APPAUTOTIME=0
MASTERCODE=123456789XYZ
LOCK=1
LUA_SD_EVENT=/line-notify.lua
```

保存後「FlashAir」を再起動（抜き差し）し、先ほど作成した「images」フォルダ内にLINE に送信したい JPG 画像を入れてください。

すると FlashAir が書込みイベントを察知して先ほど作成したスクリプトを実行し、自動で LINE に画像が送信されます。送信に成功した場合は LINE アプリを確認すると、下記のように images フォルダに保存した画像が表示されるかと思います。

以上で「LINE Notify」を使用した FlashAir と LINE の連携は完了です（図 4）。カメラなどで撮影した画像を LINE にあげたい場合は、「line-notify.lua」ファイルの「local IMAGE_FOLDR = "/images"」の「/images」をカメラで撮影した場合に画像が保存されるフォルダに変更してください。また、うまくいかなかった場合などや今回の記事の詳細については下記のブログでも公開していますのでぜひご覧ください。

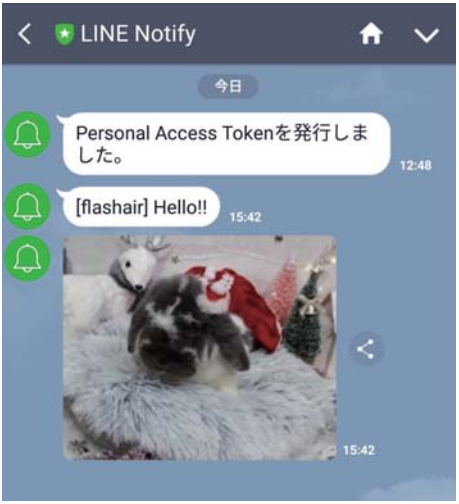


図 4: FlashAir と LINE の連携

からくりブログ (http://www.kara-kuri.jp/blog/?p=2159)

寺田 賢司

FlashAir Developers はまもなく閉鎖となりますが、また機会があればどこかでお会いしましょう。今後とも FlashAir を宜しくお願い致します。

FlashAir と Azure IoT Central で簡単 IoT

綾瀬 ヒロ

いつも FlashAir で鉄道模型を楽しんでいる綾瀬ヒロです。先日、大きなニュースが立て続けに流れましたね。「FlashAir IoT Hub 終了!」「FlashAir Developers 閉鎖!」

思い起こせば 2014 年 11 月の Maker Faire Tokyo で FlashAir 同人誌 1 号を手に取り、その変態度合いを目の当たりにした興奮冷めやらぬまま、量販店で FlashAir を購入し、すぐに鉄道模型制御を検証したのでした。最初は GPIO による制御から始め、共有メモリを用いたマイコンとのデータ共有により制御の自由度を高め、Lua スクリプト対応とともに、独自コマンドリストを解釈できるようにして自動運転までできるようにしました。このあたりの足跡は、これまでの FlashAir 同人誌（２号から各号参加させていただきました）を参照してください。

本号が最後になるのかわかりませんが、そのあたりは特に気にせず、今回も鉄道模型ネタで行きたいと思います。今回は、先日サービス終了の予告がされてしまった「FlashAir IoT Hub」の代わりとなる手軽に使える IoT サービスとして、Microsoft が提供する SaaS 型 IoT サービスである「Azure IoT Central[1]」を使って、FlashAir からテレメトリデータをクラウドに送信し、クラウド上でグラフ表示やテレメトリ値の監視ルールによるアラート送信などをご紹介してみようと思います。

Azure IoT Central とは

Microsoft のクラウドサービス Azure において提供されている IoT 関連サービスはさまざまあります。Azure IoT Hub に代表されるような PaaS 型サービスの部品群では、要件に応じてさまざまに組み合わせて使うことができました。一方、自由度が大きすぎることから「Azure IoT ソリューション アクセラレータ[2]」（旧称:Azure IoT Suite）というリモートモニタリングや予測メンテナンスなどの代表的な要件にあわせてあらかじめ組み合わせた（Preconfigured）ものを展開できるサービスを提供しています。これをさらに進めて、誰でも簡単に IoT ソリューションを構築できる仕組みとして SaaS 型の IoT サービスである「Azure IoT Central」が提供されています。

まずは、「Azure IoT Central」で自分のアプリケーションを作成してください。作成のやり方は、こちらの「Azure IoT Central アプリケーションの作成[3]」を参照してください。

1　ポータルサイト : https://azure.microsoft.com/ja-jp/services/iot-central/
2　ポータルサイト : https://azure.microsoft.com/ja-jp/features/iot-accelerators/
3　公式ドキュメント : https://docs.microsoft.com/ja-jp/azure/iot-central/quick-deploy-iot-central

Azure IoT Central への接続文字列の取得

Lua スクリプトで Azure IoT Central に接続することは、基本的に Azure IoT Hub で接続する場合と同じです。ただ、Azure IoT Central では内部の Azure IoT Hub は隠蔽されているため、Azure IoT Central のアプリケーションポータルサイトで「IoT Hub 接続文字列」を得ることができません。

これは、Azure IoT Central に接続されるデバイスは、「Azure IoT Hub Device Provisioning Service[4]」を使ってポータルサイトで得られる「スコープ ID」「デバイス ID」「主キー」から「IoT Hub 接続文字列」を得ることが前提となっているためです。

今回は簡単に「IoT Hub 接続文字列」を得るためのツール「dps-keygen[5]」を使います。

ツールをコピーしたパソコンで以下のようにコマンドを実行すると、見慣れた IoT Hub 接続文字列が得られます。IoT Hub 名は、Azure IoT Central でアプリケーションを作成した際に自動で命名されているものが表示されます。

```
C:¥dps_cstr>dps_cstr <スコープ ID> <デバイス ID> <主キー>
Registration Information received from service: iotc-XXXXXXXXXXXXX.azure-
devices.net!
Connection String:
HostName=iotc-XXXXXXXXXXXXX.azure-devices.net;DeviceId=XXXXXXXXXXXXX;
SharedAccessKey=XXXXXXXXXXXXX
```

「スコープ ID」「デバイス ID」「主キー」は、Azure IoT Central のアプリケーションポータルサイトの「デバイスエクスプローラー」でデバイス（FlashAir）を追加し、追加したデバイスの「接続」（ポータルサイトの右上あたりにある）の画面に記載されています。

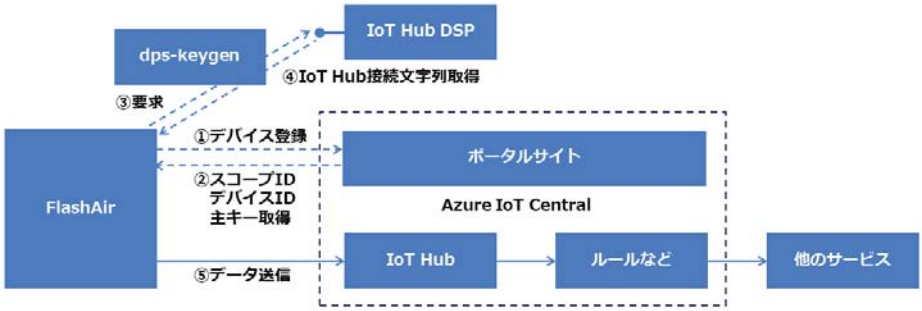


図 1: 全体概要図

4　公式ドキュメント：https://docs.microsoft.com/ja-jp/azure/iot-dps/

5　dps-keygen の GitHub サイト：https://github.com/Azure/dps-keygen

Lua スクリプトによる Azure IoT Central への接続

前段で「IoT Hub 接続文字列」を得たので、あとはいつも通り Azure IoT Hub に接続するように Lua スクリプトを書けばよいのですが、ここで筆者が詰まった点を先に書いておきましょう。Azure IoT Hub へ接続するためには SAS トークンを用いますが、SAS トークンを生成する際に、共有アクセスポリシーを skn=iothubowner などと指定していたのですが、Azure IoT Central に内包された IoT Hub に接続する際には、skn に「iothubowner」、「service」、「device」と記載してみたところ認証エラーとなってしまいました。そこで skn の指定を無くしてみたところ、認証されて接続できるようになりました。

以下に、Azure IoT Central にデータを送信する Lua スクリプトを示します。

```
-- require
require "FaTimestamp"
require "FaUtil"
require "FaAzureIoTSAS"
-- config
local resourceUri = <HostName> --dps-keygen で得られた文字列
local signingKey  = <SharedAccessKey> --dps-keygen で得られた文字列
local policy = "" -- 今回は不要
local expire = 86400
local devicename = <DeviceId> --dps-keygen で得られた文字列
-- create SAS
local timestamp = getTimestamp()
local expires = timestamp + expire
local auth = sas.create(fa, resourceUri, signingKey, expires, policy)

local function sendAzureMsg(msg)
  local len_s = tostring(string.len(msg))
  local b,c,h = fa.request{
    url = "https://" .. resourceUri .. "/devices/" .. devicename .. "/messages/
events?api-version=2018-06-30",
    method = "POST",
    headers = {["Authorization"] = auth, ["Content-Type"] = "application/json",
["Content-Length"] = len_s},
    body = msg }
  if(c == 204) then
    〈正常系処理〉
  end
  return
end
```

いつもお世話になっている cocteau666 氏のライブラリ を利用していますので、require で必要なライブラリを指定しています。ただし、前述の通り SAS トークン生成時に skn を指定しないように FaAzureIoTSAS.lua の８行目を以下のように変更しています。

オリジナル：

```
return "SharedAccessSignature sr=".. resourceUri.."&sig=".. hmac:base64en
code():urlencode().."&se=".. expires.."&skn=".. policy
```

変更後：

```
return "SharedAccessSignature sr=".. resourceUri.."&sig=".. hmac:base64en
code():urlencode().."&se=".. expires
```

ここまで紹介した内容を簡単に図式化したものを図 1 に記載しました。またこの仕組みを使って、鉄道模型車両に搭載した 3 軸加速度センサーの情報を FlashAir から Azure IoT Central に上げてみました。その様子を図 2・図 3 に示します。

Azure IoT Central では、上がってくる JSON 形式データのデータ項目を指定すると、簡単にグラフ化することが可能です。また、テレメトリに「ルール」を付加することで、例えば、Z 軸（最大）の値が 10 以上になったら、メールでアラート送信することなどができます。

みなさんもぜひ FlashAir と Azure IoT Central で IoT をお楽しみください。

図 2: FlashAir を搭載した車両の写真

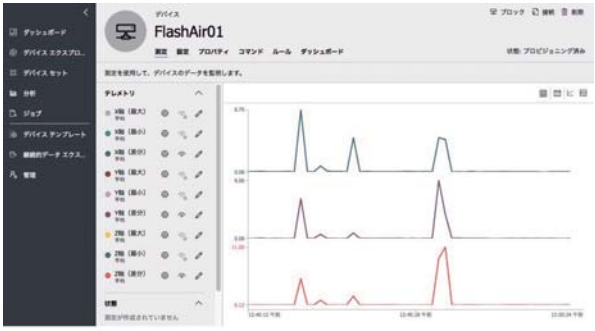


図 3: Azure IoT Central のグラフ表示画面

綾瀬 ヒロ (@ayasehiro)

某 IT 企業の運輸系インダストリーマネージャです。

最近の興味は MaaS - Mobility as a Service です。全国の公共交通の発展に寄与すべく日々飛び回っています。

FlashAir はこれからもずっと使っていきますよ。

FlashAir で写真をリアルタイムに表示する

こたまご a.k.a. ひなたん

FlashAirを無線LAN SDカードらしく、カメラで使う使い方をご紹介したいと思います。今回は、カメラで撮影した写真を即座に PC へ送信し、（できるだけ）リアルタイムに表示する、ということをやってみます（図1）。多くの方の前で撮影をし、その場でプロジェクターに表示するなど、いわゆるライブシューティングのイベントで活用できるのではないかと思います。

図 1: やりたいこと

全体の構成

FlashAir の内部では Lua スクリプトを実行し、PC ではサーバとなるアプリケーションを実行します。Lua スクリプトではカメラから書き込まれた JPEG ファイルを監視します。新しい JPEG ファイルを検出すると、サーバに対し新しく撮影された JPEG ファイルのファイルパスを送信します。 (notify.lua) サーバでは Lua スクリプトから次々に通知される JPEG ファイルを FlashAir から順番にダウンロードします。サーバはクロスコンパイルが簡単な Go 言語を利用して実装しました。 (main.go) 表示部は HTML で構成されており、ブラウザからサーバに接続することで次から次と最新の画像が表示されるようになっています。

現在の設計に至るまでの経緯

FlashAir で Lua を自動実行する手段としては、「起動時に実行する」、「ファイル書き込み時に実行する」の 2 つの手段があります。当初は画像が書き込まれる度にイベント駆動で Lua を実行し、新たに作成されたファイル自体をサーバに送信する設計を試みました。ところが、こちらには課題が見つかり断念しました。画像の送信が次の撮影までに完了しなかった場合に取りこぼしが発生することです。前回の書き込みによって実行されている Lua スクリプトが完了していない場合には新たな Lua の実行は起きないため、早いテンポでの撮影や連写時に Lua スクリプトが必要な回数実行されませんでした。

そこで、この課題を克服したのが現在の設計になります。Lua スクリプトは常時実行とし、無限ループによるポーリングで新しいファイルを検出することにしました。これにより連写撮影を行った場合でも取りこぼすことなく全てのファイルについて処理することができました。また、FlashAir では SD カードの中のファイルを HTTP 経由で取得することができます。これを活用し、JPEG そのものを Lua の request 関数で送信するのではなく、

ファイルパスのみを通知することで Lua スクリプトの処理量を減らし、できるだけ 1 ファイルあたりの処理時間を短くすることにしました。サーバではダウンロードすべきファイルをキューイングし順番にダウンロードしています。

実際に使う

ソースコードは GitHub (https://github.com/chibiegg/airview) で公開しており、使い方は README.md にも載っていますが、簡単に紹介します。FlashAir と PC は相互に通信可能な同じネットワークにつないでください。インターネット上のサーバなど、間に NAT が必要な環境では動作しません。FlashAir はステーションモードでも AP モードでも構いません。FlashAir にはリポジトリに含まれる「notify.lua」を書き込みます。この時、先頭にサーバの IP アドレスを含む設定を書き換える必要があります。FlashAir のデフォルト設定の場合には 192.168.0.11 で問題ないことが多いですが、実際に PC の IP アドレスを確認し、異なる場合には書き換えてください。この Lua スクリプトを起動するために、「/SD_WLAN/CONFIG」に次の一行を追記してください。「LUA_SD_EVENT=/notify.lua」。サーバ側の実行ファイルはリポジトリの Release からダウンロードすることができます。利用時にはこれを実行することで 8000 番ポートにて HTTP サーバが起動します。ファイアウォールが動作している場合には外からの接続を許可する必要がある場合があります。

うまく動作すると、カメラで撮影する度にサーバアプリケーションのコンソールに次のようなログが出力されます（図 2）。ブラウザで http://127.0.0.1:8000/ にアクセスすると、全画面で写真を表示することができます。

図 2: コンソールログ

ソースコードの解説

ソースコードから重要な処理を抜粋し解説します。まず、起動時に監視するフォルダを探します。ほとんどのデジタルカメラでは「カメラファイルシステム DCF」の仕様に従ったフォルダ構成を採用しています。DCF では「DCIM」ディレクトリの下に、数字 3 桁と半角英数 5 文字からなる DCF ディレクトリ (ex: 101CANON) に画像が連番で保存されます。ポーリングで新しい JPEG ファイルを監視するために、スクリプト起動時にはもっとも新しい DCF ディレクトリを探すようにしています。これは DCIM ディレクトリの下にある数字 3 桁のディレクトリを正規表現で探し、もっとも数字の大きいディレクトリを返す関数で実現しています 。ソースコードを示します。

```
function find_latest_directory()
  local latest_dir = nil
  local latest_number = 0
  for dir in lfs.dir("DCIM") do
    c = string.match(dir, "^(%d%d%d)")
    print(c)
    if c ~= nil then
      local n = tonumber(c)
      if n > latest_number then
        latest_number = n
        latest_dir = dir
      end
    end
  end
  return "DCIM".."/"..latest_dir
end
```

ポーリングは 500ms 毎に行うようにしています。DCF ディレクトリ内のファイルを全てチェックし、前回送信した画像番号よりも新しいものを全部送信するという愚直な処理になっています。下記がそのソースコードです。

```
while true do
  for file in lfs.dir(photo_dir) do
    local path = photo_dir.."/"..file
    local suffix = string.sub(string.lower(path), -3)
    local number = get_file_number(path)
    if suffix == "jpg" and number > current_number then
      post_text(NOTIFY_URL, path)
      current_number = number
    end
  end
  sleep(500)
end
```

概ねこの 2 つの処理によって Lua スクリプトは構成されています。

サーバ側は、FlashAir からの新しいファイルパスの通知を受ける処理、順番にファイルをダウンロードする処理、の大きく 2 つの処理があります。

まず、FlashAir からのファイルパスの通知は、POST されてきたファイルパスを一旦キューに追加する処理だけを行っています。この時、HTTP リクエストの送信元 IP アドレスを元にダウンロードすべき URL を生成しています。下記がそのソースコードです。

```
func notify(w http.ResponseWriter, r *http.Request) {
  b, _ := ioutil.ReadAll(r.Body)
  hp := string.Split(r.RemoteAddr, ":")
  u := fmt.Sprintf("https://%s%s", hp[0], string(b))
  q.put(u)
  logger.Info("Enqueue %s", u)
}
```

キューに積まれた URL は Go ルーチン（スレッドのようなもの）で順番にダウンロードされていきます。ソースコードを示します。

```
func download_loop(q *queue.Queue) {
for {
    dl, _ := q.Get(1)
    for _, di := range dl{
      u := di.(string)
      logger.Debugf("Get %s", u)
      dist, err := download(u, "downloads")
      if err != nil {
        logger.Errorf("Download Error: %s", err)
      }else{
        logger.Infof("Download complete: %s", u)
        latestFile = dist
      }
    }
  }
}
```

最後に表示部はほとんど JavaScript で書かれています。こちらもシンプルにポーリング処理によって最新の画像を取得するようになっています。下記がそのソースコードです。

```
setInterval(function(){
  $.get("/latest", function(data){
    if(lastFile != data){
      console.log(data);
      var image = new Image();
      image.onload = function(){
        $('body').css('background-size', 'cover');
        $('body').css('background-image', 'url(' + data + ')');
      };
      image.src = data;
      lastFile = data;
    }
  });
}, 1000);
```

まとめ

WiFi 搭載のカメラは増えてきていますが、今回のように撮影中にリアルタイムに転送できるカメラはまだまだ少ないです。FlashAir は Lua スクリプトで自由な動作をさせることができ、サーバ側の処理と組み合わせることで痒いところに手がとどく機能を実装できるところが嬉しいです。

こたまご a.k.a. ひなたん (@chibiegg, @hinatan_net)
写真と電子工作が好きな男の娘
普段は IT エンジニア女子
インフラ写真集を出しています
https://chofutech.booth.pm/
Instagram: @chibiegg, @hinatan_net

世界初？ FlashAir 羽ばたき飛行機！

高橋 祐介

オリジナルデザインの多葉羽ばたき機に、FlashAir を受信機として搭載した機体です。きちんと確認していませんがたぶん世界初であろうと思います（羽ばたき機マイナーだし）。FlashAir に接続したスマホからブラウザ経由で遠隔操作します。FlashAir の接続端子のうち 2 つを GPIO として使い、オンオフを無線で制御して、モータドライバ IC[1] を経由して左右のモータを制御しています。早朝の公園で怪しく飛んでいる様子を YouTube にアップしていますのでご覧ください。[2]

1　DRV8835 デュアルモータードライバ基板（https://www.switch-science.com/catalog/1637/）
2　世界初？ FlashAir™ 羽ばたき機を飛ばしてみた（https://youtu.be/YJimsU9Oh38）

電力消費の大きそうな Wi-Fi ルータと HTTP サーバを載せて飛ぶわけなのでバッテリの持ちが心配でしたが意外と大丈夫で、1 分ほど飛ばしたあと、そのまま 2 台のスマホ間で数百 MB の動画データ転送も難なくこなしてくれました。イベント会場とかでヘリウムガス風船等にぶら下げて浮遊させておけば Wi-Fi ルータ /HTTP サーバ /FTP サーバの空中基地局として利用できそうですね！（初出記事[3]）

ソースコード（抜粋）

```
（前略）
function gpio_half( data1,data2 ){
  gpio_full( data1 );
  setTimeout( gpio_full, 240, data2 );
}
（中略）
<input type="button" value="Left" onclick="gpio_half(0x18,0x0a)">
<input type="button" value="Up" onclick="gpio_full(0x1f)">
<input type="button" value="Right" onclick="gpio_half(0x06,0x0a)">
<input type="button" value="Down" onclick="gpio_full(0x00)">
（後略）
```

機体スペック

- 全幅：720mm
- 全長：700mm
- 全備重量：34 グラム（150mAh リポバッテリ含む）
- 連続飛行時間：2 分

3　世界初？ FlashAir™ 羽ばたき機を製作しました（https://bit.ly/2xEKWip）

高橋 祐介（flappingwing@goo.jp）
ブログ「羽ばたき飛行機製作工房」主宰
（https://blog.goo.ne.jp/flappingwing）
オリジナル羽ばたき飛行機のデザイン・製作を行っています
「大人の科学マガジン デルタ・ツイスター」（学研プラス 2012）の原型デザインを担当
ファブラボ北加賀屋（大阪市住之江区）運営メンバー
（https://fablabkitakagaya.org/）

FlashAir の電波強度レベルを確認しよう

Takehiro Yamaguchi

ほんまに驚きましたよ。Twitter で FlashAir Developers が 2019/09/17 で閉鎖というTL が流れてきた時、パソコンのモニタを両手で掴みガタガタさせながら「えーー?」ですよ。今は沢山の技術資料がどこかでバックアップされるのを願ってるユーザーの一人です。

振り返れば、私は W-02 で FlashAir デビュー、当時は WEB や同人誌を見ながらFlashAir にひたすら GET リクエストを送り続けました。そして W-03,W-04 では FlashAir IoT Hub に興味を持ったわけですが、FlashAir IoT Hub も 2019/07/31 に閉鎖となってしまったわけです。この状況を「崖っぷち」という言葉で表現するよりも、すでに崖から落ち、たまたまあった小枝に手が伸びたような状況なのかもしれませんが、非力なマイコンボードをよく使う私としては掴んだ小枝を離すわけにはいきません。

非力なマイコンボードでも SD-CARD 用ライブラリが整備されている場合が多く、WEB クライアント機能がある FlashAir には魅力を感じてなりません。WEB クライントと言いたいのは、HTTP クライアントだけでなく FTP クライアントとしても可能だったりするからです。http:// でファイルを取得することもできれば、ftp:// でサーバーにファイルをアップロードが実現できるので、マイコンボードで生成した結果でいきなり WEB サーバーにあるページを更新するなんてことも可能なのです。

しかし FlashAir を WEB クライアントとして利用する場合、ちょっと困った事がありました。それが電波強度レベルが分からないという点です。FlashAir には小さなアンテナが実装されていることは予想でき、めちゃめちゃ遠くまで無線 LAN の電波が飛ぶわけではありません。気が付けば Wi-Fi ルータからの距離が離れ過ぎていていつの間にか " 切断 " なんてことを経験された方は多いのではないかと思います。例えばスマホではお馴染みの Wi-Fi アイコン「📶」で電波強度レベルが表示されるので「バリ3やから安心やわ」っとなるわけです。電波強度レベルを誰もが理解できるアイコンがちょっと羨ましく思っていたりしてたのですが、実は FlashAir にもあったんですよ。

「📶」これ FlashAir にもあったんです

いつも iSDIO はライブラリまかせ、Wi-Fi ルータにはあっさり接続されることもあり、FlashAir のステータスレジスタなんて今までじっくり見る機会がなかったのですが、たまたまマウスのホイールをゆっくりと回転させながら FlashAir Developers を見ていると、

おやおや、これはもしかして……となりました。それがこれ（図1）。

アドレス	サイズ [バイト]	名称	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	読み込み／書き込み
00500h	6	Reserved									
00506h	1	WLAN	Connected	Infra-Direct		AP-STA	Group	WPS		Scan	読み込み専用
00507h	1	WLAN (Reserved)								*1	読み込み専用
00508h	32	SSID									読み込み専用
00528h	1	Encryption Mode*2									読み込み専用
00529h	1	Signal Strength									読み込み専用
0052Ah	1	Channel									読み込み専用
0052Bh	5	Reserved									

図 1: FlashAir のステータスレジスタ

Signal Strength、アドレス 529h、バイト数 1、読み込み専用。間違いなく電波の強さっぽいですよね。しかも説明無しという挑戦的な感じが気になるじゃないですかぁ。トレジャーハンターの気分で下記 Lua スクリプトで取得し、この 1 バイトを追っかけてみました。

```
local lvl = tonumber(string.sub(fa.ReadStatusReg(),83,84),16)
```

Wi-Fi ルータに近づけたり離したり、キッチンのステンレスボールをかぶせたりしてみたのですが間違いないですね。これは電波強度を表すステータスレジスタでした。電波が強い時の最大値が 100 で、値が変化することを確認できました。この値をマイコンボードに渡し LED などで表示させることでスマホのような「📶」が実現できます。この同人誌が配布される時にはもう FlashAir IoT Hub は閉鎖されているのですが足跡を残しておきます（図2）。この取得した値を FlashAir IoT Hub で可視化、ピンクが電波強度レベルになります。FlashAir Developers、FlashAir IoT Hub、お疲れ様でした。ありがとうございました。

図 2: FlashAir IoT Hub で電波強度を表示

NADA ELECTRONICS,LTD.

Takehiro Yamaguchi（@nada_tokki）

次はこれをやってみます

```
fa.serial("init",9600)
fa.serial("write","Hello World")
```

FlashAir で FPGA を操れ！

長船 俊

FlashAir W-04 ではインターフェースが強化されて、SPI/I2C/UART/GPIO を混在させて使うことができるようになりました。また Lua で使えるメモリも大幅増強され、かなり自由にアプリケーションを書くことができるようになっています。ここでは W-04 の I/O をさらに使い倒すべく、Intel FPGA を直接コンフィグレーション・制御できるライブラリ「Canarium Air」を紹介したいと思います。

FPGA コンフィグレーション機能

Canarium Air で FPGA のコンフィグレーションを行うには、まず W-04 と FPGA の間を図1のように結線します。このとき、FPGA 側のコンフィグレーションモード（MSEL ピン）および、Quartus のコンフィグレーションスキームを PS モード（Passive Serial）に設定しておきます。また、"Generate compressed bitstreams" にチェックを入れて圧縮ビットストリームにした方が高速になります。

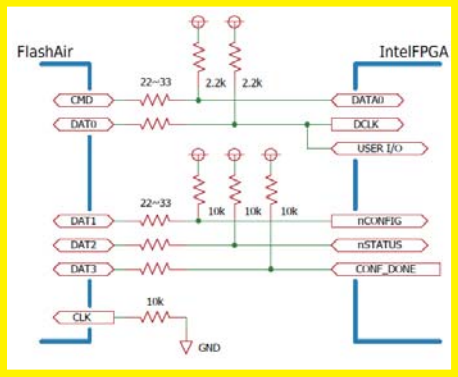


図 1: FlashAir と FPGA の結線図

コンフィグレーションデータは RBF ファイルを使用するので、Quartus の Device and Pin Options → Programming Files タブで "Raw Binary File(.rbf)" にもチェックを入れてコンパイルします。

W-04 側は GPIO モードを使用します。/SD_WLAN/CONFIG ファイルに IFMODE=1 を追加しておきます。

以上の準備ができたら、canarium_air.lua と RBF ファイルを任意のフォルダに格納し、ユーザーの Lua から以下のように呼び出します。

```
require "/foo/canarium_air"
....
res,mes = ca.config{file="/foo/hoge.rbf"}
```

コンフィグレーションに成功すると res に true を返します。Canarium Air ではコンフィグレーションの高速化のためキャッシュファイルの作成を行います。そのため初回のコンフィグレーションのみ時間がかかることに注意してください。

I2C to Avalon-MM ブリッジ機能

Canarium Air では FPGA のコンフィグレーション後、I2C を使って内部へのアクセスを行うこともできます。内部アクセスを行うには、Platform Designer で "Intel FPGA I2C Slave to Avalon-MM Master Bridge Core" を次のようなパラメータでインスタンスします。

- **I2C Slave Address は 0x55 をセット**
- **Byte addressing mode は 4 にセット**
- **Number of Address Stealing bit は 0 にセット**
- **Enable Read only mode は OFF にセット**

インスタンスした Qsys モジュールを作成し、I2C の SDA 信号を USER I/O ピンに、SCL 信号は DATA0 ピンにアサインします。

ca.config でコンフィグレーション後、以下のように呼び出すことで、Qsys モジュール内のメモリ空間に自由にアクセスすることができます。

```
avm = ca.open()
sysid = avm:iord(0x1000)        -- 0x1000 のレジスタを読み込み
avm:iowr(0x1100, 1)             -- 0x1100 のレジスタに 1 を書き込み
avm:close()
```

Canarium Air にはこのような I/O レジスタアクセスの他にも、メモリへの読み書きやバイナリファイル、インテル HEX・モトローラ S レコードファイルのロードのメソッドも用意されています。また、W-04 の CGI 機能を利用してクライアントサイドの JavaScript から各種ファンクションを呼び出すことのできる RPC ライブラリも整備中です。詳しくは GitHub のリポジトリを参照ください。

他にも、W-04 と Canarium Air ライブラリを利用することを前提とした廉価 FPGA ボードも用意していますので、何かの機会に利用していただけたらと思います（図2）。

図 2: Canarium Air 対応ボード

Canarium Air は以下の GitHub リポジトリからダウンロードできます。

https://github.com/osafune/canarium_air

長船 俊 (@s_osafune)

J-7SYSTEM WORKS の FPGA エンジニア。アートワークからアプリケーションソフトまでだいたい全部担当。

Airio-Base で Mbed OS 5

(株) クレイン電子　福屋 新吾

Airio-Base[1] は、Arm Mbed OS 2（以下、OS 2）を利用できますが、今後の主流は Arm Mbed OS 5（以下、OS 5）です。ここでは、FlashAir と通信する iSDIO ライブラリなど、これまでの資産を活かすべく、OS 5 を利用した開発方法を紹介します。

Mbed CLI と bare-metal オプション

以下の URL から Mbed CLI を DL[2] し、セットアップします。次に、作業用フォルダを作成し（例:"D:\mbed"）、公式サンプルコードを PowerShell を用いて DL します（図 1）。

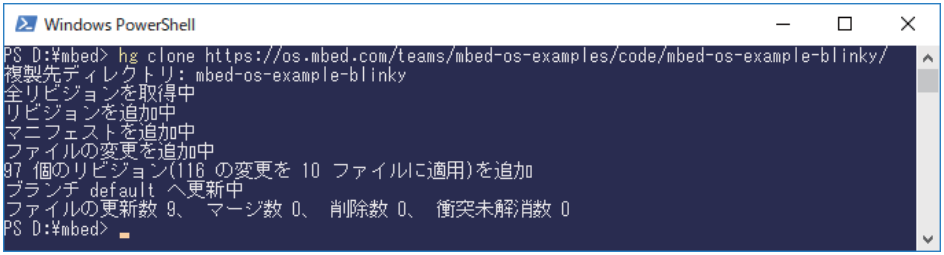


図 1: Powershell でサンプルコードを DL

DL したファイルの内 "mbed_app.json" の 1 行目と 2 行目の間に以下のコードを追加します。

```
  "requires": ["bare-metal"],
```

次に、ビルドに必要なファイルを DL するコマンドを実行します。

```
PS D:¥mbed¥mbed-os-example-blinky> mbed deploy
```

以上で、OS 5 で Airio-Base の FW 開発を行う準備が整いました。この bare-metal オプションは OS 2 に比べ大きなリソースが必要となった OS 5 のペリフェラルを一部を取り除く機能です。このオプションの登場により、今まで、主に SRAM 容量が足りなかったボードでも OS 5 が利用可能となりました。

L チカを試す

続いて main.cpp を書き換えます。DL したサンプルコードは bare-metal オプションのない、OS 5 フル機能でのコードなので OS 2 相当のコードに書き換えます。

1　FlashAir 対応の Arduino 互換形状の NXP LPC11U35 マイコンボード
http://crane-elec.co.jp/products/vol-14_airio-base/

2　https://os.mbed.com/docs/mbed-os/v5.12/tools/installation-and-setup.html

```
#include "mbed.h"
DigitalOut led1(LED1);
int main()
{
    while (true) {
        led1 = !led1;
        wait_ms(200);
    }
}
```

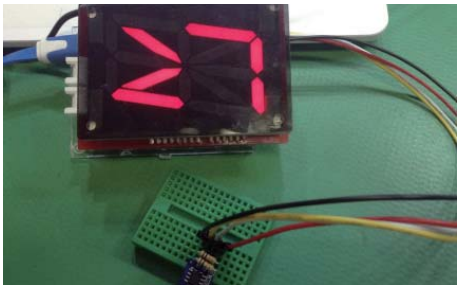


図 2: 温度を取得して表示

編集し終えたら、以下のコマンドを PowerShell で実行し、コンパイルします。

```
PS D:¥mbed¥mbed-os-example-blinky> mbed compile -t gcc_arm -m lpc11u35_401
```

コンパイルが成功すると "mbed-os-example-blinky.bin" というファイルが生成されますので、それを Airio-Base に転送します。無事 L チカすれば OS 5 を用いた開発環境の完成です。

ライブラリは活かせるか

資産を活かすべく「FlashAir Developers Summit ハンズオン @ 大阪 2018 年度」で用いた "FlashAir_iSDIO_16seg_ADT7410_step1" の移植を試みました。このコードは、ADT7410 温度センサから温度を取得して 16 セグメント LED に表示しつつ、FlashAir にその値を書き込みます。そして、書き込まれたデータをスマートフォンから FlashAir にアクセスして読み取るというものです。結論から言うと完全にはうまくいきませんでした。通常のポート制御や I2C 通信は問題なくそのまま動きましたが、OS 5 になってから大幅に変更があったファイルシステム関連のペリフェラルがうまくいかず、結果として SD カード制御が動作しませんでした。Web で調べてみるとこの辺りの実装は、OS 5 のバージョンによって仕様が違ったりするようで発展途上といったところでしょうか。今後に期待したいところです（図2）。

最後に

本稿では、gcc コンパイラを使うために Mbed CLI を利用しましたが、bare-metal オプションさえ使っていれば Mbed クラウドコンパイラでもできるはずです。しかし、現在 (2019/7/18) コンパイル時にエラーが発生してしまいます。これは armcc コンパイラのバージョンの関係で、一部の条件でエラーとなるようです。今後は改善されることを願います。

株式会社クレイン電子（@crane_elec） 福屋 新吾

なんと寄稿二回目！FlashAir がなければ出会うことのなかった人達や、開催することのなかったセミナーなど…。私にとっては、データだけでなく思い出も記録するカードになりました。

Mbed de iSDIO

ほげじゅん

Mbed で iSDIO 使えたらモテるんちゃうのん？ホスト側から無線 LAN のコントロールできたら素敵やん？そのうえ共有メモリで通信できたら痺れるやろ？そう、Mbed 界隈ではやたらモテにこだわるのである。今のとこ、iSDIO 機能を実装したデバイスは東芝メモリ製の FlashAir™ が唯一の存在だな。これは是非動かさねば！モテのために（しつこい）（文中、多大なる脚色が入っていますが何卒ご了承ください。）

iSDIO

下記は FlashAir Developers 内の説明です。

> iSDIO (Intelligent SDIO) をつかえば、マイコンボードなどのSDメモリカードホスト機器から、FlashAirの無線LAN機能の高度で精密な制御が行えます。 FlashAirがHTTP通信コプロセッサになってしまうのです。

FlashAir で使える iSDIO コマンドの仕様（SD カードへのコマンド発行に関するプロトコル）は FlashAir Developers（残念ながらもうすぐ消滅）を眺めればわかるだろうってことで、早速調査を開始。まぁ要するにこんな感じの操作が必要ということ。（より深く知りたい貴方は別途資料[1]参照）

- **CMD48/49 で拡張レジスタ拡張 { データ , ポート } 領域の読み書き**
- **コマンドレスポンスレジスタの読み出しでコマンド完了を待つ**

Mbed で動かす

Arduino のチュートリアル[2]を参考に、Mbed へのポーティングを開始しました。

Arduino のライブラリでは、SD カードへのコマンドの発行に、Sd2Card クラスを利用しています。Mbed ではこれに相当するクラスとして SDFileSystem クラスというものがあり、それを利用して iSDIO 操作クラスを作成（iSDIO）することにします。

また、後の拡張の容易性を担保するため iSDIO 操作と FlashAir の iSDIO 操作に特化した部分は分離しておきます。こうすることにより、将来的に FlashAir のバージョンが更新され実装が変わった場合には、FlashAir iSDIO 操作クラス (FlashAir_iSDIO) のみメンテナンスすればよいという利点があります。

1 SD Specifications Part E7 iSDIO Simplified Specification
2 https://www.flashair-developers.com/ja/documents/tutorials/isdio/

こうして整理されたレイヤを図1に示します。左側が Arduino を表し、右側が今回作成した Mbed 用のライブラリを表しています。

既存のクラスである SDFileSystem を継承して iSDIO クラスを作成し、それをまた継承して FlashAir_iSDIO クラスを作成しています。一部、タイプ量を減らすのと見通しをよくするために C++ のテンプレート記法を使用していますが、ここでは説明しないので詳しくは実装[3]を参照してください。

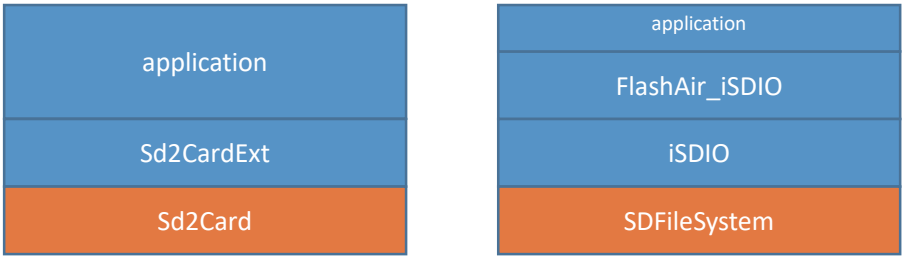


図 1: FlashAir_iSDIO のレイヤ

こうして作成したライブラリをクレイン電子製の Airio-Base およびマクニカ製の TOYBOARD で動作確認を行いました。結論から述べると最初の動作確認では正常に動作しませんでした。鋭い方はライブラリ内および動作確認アプリケーションにシリアル出力を仕込んであるのに気づかれたこととおもいますが、それはそのときのデバッグ作業の名残です。これにより、SD カードとの通信が Teraterm 等のシリアルモニタ用のアプリケーションを使うことにより丸裸になります。なお、今回動作しなかった理由はとても間抜けなもので、FlashAir の FW が iSDIO 対応でなかったというね。。。そのため、せっかく iSDIO のためのコマンドを発行しても、FlashAir はそのようなコマンドは知らないということで無視されていました。みなさんに限ってそんなことはないと思いますが、iSDIO 機能を動かす場合は最新の FW に更新するのを忘れないでください（了）

3 https://www.flashair-developers.com/ja/documents/tutorials/isdio/

ほげじゅん (@hogejun)

半導体業界組み込み界隈と渡り歩いてきたロートル。現在、何屋なのかは自分でもわからなくなっているところ。Mbed 祭りの懇親会で余熱氏から熱烈な FlashAir の売り込みを受けそれ以来プロダクトに携わる方々のファンになり現在に至る

FlashAir の進化（AirLapse 編）

高瀬 秀樹

2014 年 3 月、私は浜松町の東芝本社の会議室に居た。当時、東日本大震災の復興の様子を 10 年以上定点撮影するシステム（AirLapse の前身）の開発を担当していた私。この時点では、Eye-Fi を使用して運用を行っていた。ところがこの Eye-Fi が思い通りに動かない。写真の枚数が増えてくると転送しなくなり３～４ヵ月毎に現場へ行くはめになる。これでは 10 年以上使い続ける運用は出来ない。そんな中たまたま行った OSC で FlashAir が独自アプリを書ける事を知った。早速購入し、テストをしてみるといくつかの問題点が見つかった。そこで東芝の担当者にアポイントを取り、対応の可能性を聞きに来たのだ。

システム構成や現状の問題点、FlashAir に期待する機能を一方的にたたみかけた。担当者が真剣に前のめりに聞いてる様子を見て私のテンションもかなり上がっていた。説明が終わった後、微妙な空気の沈黙が訪れた。数秒だったのかもしれないが数分のように感じられた。そこにいた開発担当者とプロジェクト担当者は、「なんとか協力はしたいけど最低 1000 個、出来たら 1 万個程度の発注が約束できれば何とかなるかもしれないけど、今回の提案程度での数では、とても残念だけれど実現は無理だと思う」と悲しそうな表情でおっしゃった。私は茫然として寂しく帰社した。

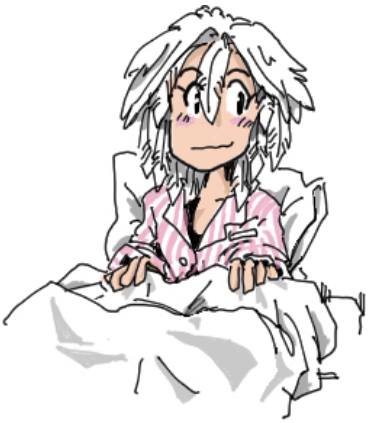
それから、半年以上の時間が経過したある日、担当者から連絡を受けた。FlashAir の次期バージョンを開発中である事、お願いした機能が実装できるかもしれない事を聞かされた。東芝と言う大きな会社が私の様な小さなプロジェクトの為に動いてくれたことに心底感動を覚えた。もちろん、全ての機能が実装された訳ではないが、この時期 FlashAir Developers も充実してきて、Arudino や Raspberry Pi に続く日本産のイノベーションデバイスが誕生するのだという感覚が次第に現実のものとなってきた、凄いプロジェクトの末席に携われることに震えたことをよく覚えている。

しばらくたったある日、開発担当に実装される機能に関しての説明を聞くとともに、再度こちらの要望を伝えた。それほど時間はかからず、テスト版 FlashAir をお貸しいただいた。アプリの開発とテストを進め問題点を開発チームと共有し、問題点をひとつずつつぶしていただき、専用ファームウェアを提供していただいた。このファームをもとに AirLapse のβサービスに入った。ここで開発された機能の多くが、W-04 に実装されることになった。その W-04 は、転送速度向上、無線電波強度向上、てんこ盛りな機能追加をして 2017 年に発売となった。AirLapse もこのバージョンに移行している。AirLapse は IoT とは言えデバイスからクラウドへの片方向の写真転送がほとんどで、クラウド側からの制御は、ほぼ無い。そこで、コントローラ側にも FlashAir を実装する新たなボードの開発に着手した。今までの経験上、FlashAir だけでリアルタイム処理を行うには無理

があるので、省電力 CPU と SD スロットを実装したメインボードに IO 関係をドーターボードとして乗せる構成で開発を進める事となった。

このボードが昨年の FlashAir デベロッパーズサミット 2018 でハンズオンにも使用され、Airio-Base として販売されることになる。FlashAir デベロッパーズサミット 2018 は、とても盛況でユーザのみなさんの熱量を感じ、開発スタッフの方々とも、「MicroSD 化や Bluetooth の実装もいずれはしたいよね〜」などと話していた。その後他の業務で多忙となり、いつ寝ているのかわからないような働き方改革とは無縁の生活が続き新ボードの開発は全く進んでいない。そこへ今年も FlashAir 同人誌の原稿寄稿の依頼がやってきた。

それからどの位の時間が過ぎたのか記憶にないが、携帯に開発担当から電話があり、すぐに会いたいと言われた。その時私は、なぜか海外に居たのだがその日のうちに東京に戻り、開発担当の話を聞いた。次期バーションの FlashAir は基本機能が MicroSD に収まり、各種オプション機能は MicroSD ー SD 変換カードに載せるので、私の要望は多分実装できると思う、とのことだった。12 時間以内にこの機能で十分かを検証して欲しいという事だった。早速翌日の仕事を全てキャンセルして、追加仕様と出来る限りの提案を徹夜で作った。締め切り 15 分前に担当に伝え、3 分後には「ありがとうございます。出来る限り実装します。」と返信があり、記憶を失った。その後目にした MicroSD 版 FlashAir は、SD への変換アダプターが各種用意されていて、GPIO、LTE、Bluetooth、LPWA に対応した夢のようデバイスだった。しかも、プラットフォーム部分のコードはオープンソースで提供されていてこれも素晴らしい！



これで本当に、Arudino や Raspberry Pi を超えるイノベーションを起こせる IoT デバイスになると確信した。すぐ開発担当に謝意を伝えて早速、次期バージョンの AirLapse の開発にかかったところで、遠くで聞こえていた電話の音が、だんだん大きくなって目を覚ました。電話の主は、同人誌の編集担当で「原稿締め切り過ぎてるよ〜、もう間に合わないわ〜」と言われて、どのくらい寝ていたのだろうか、どこまでが現実だったのか、、、FlashAir 万歳！！

（株）タイプ・アール　高瀬 秀樹 (@hide_tks)

子供の頃から電子工作好き、長年放送局関係の仕事を手掛け、現在は IoT ALGYAN の運営委員も務め、Microsoft Azure と FlashAir や、Arudino・RaspberryPi をつないで IoT でビジネスしようと格闘中。カレーとハンバーガーが好物で、スキー・AV・アマチュア無線が趣味

FlashAir と Lua と自作のツールと

GPS_NMEA_JP (@Seg_faul)

早いもので、FlashAir と出会ってからもう 4 年近く経ってしまいました。執筆に当たり、「せっかくなので思い出話などを書いていただけると」とご要望を受けたのもあり、Lua 含めた FlashAir とのあれこれを 2 ページほど書かせていただくことにしました。

実は同人誌 3 でも、出会いと当時の印象を書いていますので、見比べてみると面白いかもしれません。

Lua との出会い

FlashAir と出会うよりはるか前に、Lua と出会っていました。

私は、いろいろなプログラミング言語を手当り次第触った時期があり、その際に単体の Lua に触れる機会がありました。

Lua は皆さんご存知のように、非常に軽量なスクリプト言語で、主に他のプログラムに組み込んで使用することを想定して作成されている言語です。

同系統の言語としては、Tcl 言語や Python、mruby などがあり、最近では JavaScript なども同様の使い方がされるようになってきています。

ただ、Lua はそれらの言語とは違い、単体で注目されることはほとんどありません。単体での実行環境はあるにはあるのですが、拡張に手間がかかることもあり、単体で使用される例は多くはないためです。大体、なにかに組み込まれて使用されています。

もともと私は BASIC 系の言語出身で、そこからアセンブラ、C など触っていったものですから、Lua の文法には馴染みがありました。

しかしながら、前述の通り、単体ではあまり使いみちがない言語のため、忘れ去ってしまうことになりました。

FlashAir との出会い

FlashAir と最初に出会ったのは W-02、ただし情報だけでした。無線 LAN 内蔵の SD カードというものには興味を持っていて、類似の品も含めていろいろ調べていた時期がありました。しかし、W-02 はブラウザ上の JavaScript 主体、当時の自分は JavaScript には興味がなかったため、これもまた忘れてしまうこととなりました。

状況が変わったのは W-03 が登場するというプレスリリースを見た時、「SD カード上で Lua が動く…！？」と衝撃を受けました。

当時、ポメラという電子メモ帳 (というかワープロ) を持っていた私は、無線 LAN に繋

がり、スクリプトを組んで実行できる W-03 にはかなり興味を持ちました。
そして発売されてすぐに電気屋さんへ車を走らせ、買いに行ったのです。

Lua on FlashAir の苦労と自作ツール

FlashAir の Lua 機能を早速使い、ポメラ上でスクリプトを編集できるようにしたり、サーバー上からスクリプトをロードできるようにしたりとしているうちに、開発環境に不満が出てきました。

FlashAir はエラー発生時にそれを知る手段が通常の方法では用意されていません。そのため、単にメモ帳と SD カードリーダの組み合わせだけでは、開発に相当苦労することになります。その上、FlashAir の端子を IO ポートとして使う場合、基板と SD カードリーダの間を行ったり来たりすることとなります。

そこで作成したのが FlashTools Lua Editor (以下 FTLE) と、FlashTools GPIO Tester & Checker(以下 IO テスター) でした。

FTLE は、FlashAir の各種機能を最大限に活かし、ブラウザ上で Lua スクリプトの編集から実行、エラー内容の取得、ファイルの整理までできるようにしたツールです。

IO テスターは、ブラウザ上から FlashAir の IO 機能をリアルタイムに操作できるようにしたツールです。

これらにより、FlashAir でのスクリプト開発の環境を相当改善したと自負しています。

一方で、FlashAir の便利さを伝えるための動画を作り公開したり、ハマりやすい点を共有するために FlashAir 開発者向け非公式 wiki を立ち上げることをしたりしました。

スクリプトのサンプル、開発支援ライブラリなども数多く揃えましたので、これから FlashAir で開発される方も、興味を持たれた方も、これらを是非お試しいただけると幸いです。

おまけ程度のツール置き場 https://sites.google.com/site/gpsnmeajp/
Seg's Homepage https://sabowl.sakura.ne.jp/gpsnmeajp/
FlashAir 開発者向け非公式 wiki http://seesaawiki.jp/flashair-dev/

GPS_NMEA_JP (@Seg_faul)
自サイトにてツールを公開したり、非公式 wiki の管理をしていたりする人。個人的な代表作はいまでも FTLE。
VR の方では VaNiiMenu など作っています。

煌めく空との九年史（ノンフィクションとフィクションの間に）

上岡 裕一

九年前

私が十数年住み慣れた土地。仲間と育てた作物は「世界初」と謳っていたが、ライバルに追い越されてしまい まだまだ品種改良とコストダウンが必要だった。しかし、親からそのお金は無い、と言われて腐っていた。

ある日、親から知らない土地に行ってほしいと言われた。新しい作物を作って売るのには、自分の力が役に立つと言われたので、２つ返事で ok した。

新しい土地に行ってみると、作物は よそから買ってきたモノを手直しせず ラベルを貼り替えて売る と言われ「作るのが仕事じゃなくて、売るだけか」と思い、不満に思った。

まずは、その「よそ」の人達と会った。その作物には、面白い特徴が有った。作物同士を近づけると、片方の色が相手に移って同じ色になるというものだった。「作物同士が同じ色になって、何が面白いの?」と思ったが、実は色だけではなく 味も同じになる と聞いた。これなら、一つ美味しい作物を作れば 隣の人の作物に直ぐに移して その味を味わってもらえる。作物はたくさん売れないかもしれないけれど、美味しさを共有出来る。なんだか楽しくなってきたが、「よそ」の人達は「他の農家の人達も仲間に入れて、誰でも美味しいものが作れる様にしよう」と言い出した。

早速、他の農家を訪ねて回ったが「誰でも美味しいものが作れたら、うちの作物が売れないし、美味しいだけでなく特色を出したい。」と言われ、仲間は増えなかった。

仕方が無いので「作物の美味しさを 相手に渡す方法」という品種改良マニュアルを作ったが、肝心の「よそ」の人達は姿を見せなくなってしまい 結局自分たちだけで開発を続けることになった。ようやくマニュアルが完成したので農家を回ってみたが、ちっとも評判が良くなかった。何故なら、美味しさを渡す相手側も品種改良が必要で、品種改良した同士でないと美味しさの交換が出来ないからであった。「渡す相手側は、普通の作物でも大丈夫な様にしないと、売れないな」と思った。

八年前

ある日、品種改良に取り組んでいた仲間が「ちょっと見てみます?」と話しかけてきた。

それは 美味しい作物の味が、あっという間に隣の作物に移るデモだった。隣の作物は 普通に売られているもので、ただただ驚き「これ、面白い」となった。このデモを持って、他の農家を回り始めた。みんな「これは魔法か?」と驚いて

くれた。これならイケる！と嬉しくなり、仲間と祝杯をあげる日が続いた。

仲間というより大先輩が、品種改良した作物にニックネームを付けてくれた。(この作物が、自分の子どもの様に愛着が沸くものになるとは思っていなかった。) ただ、デモに使った作物は品種改良も半ば。量産するためには お金と時間が必要だった。品種改良に取り組んでいた仲間の疲労はピークに達し、やがて飲みに行く暇もなくなった。

七年前

作物は、ようやく販売出来そうなところまでたどり着いた。ニックネームをデザインしたラベル、ケース、作物の成分表示等 仲間同士でイチから考えた。一番悩んだのは「作物から作物に美味しさを移す手順」をどの様に書くか。手順書が間違っていたり、勘違いされてしまうと 作物が売れなくなったり 訴えられてしまうことも有ることを知った。(味が良ければよい、ではなかった。)

そして販売開始。数日後にクレームが飛び込む。作物の付属品が動かないとの事。付属品を調べてみると、出荷前検査で大丈夫だったものが出荷後に破損していた。大急ぎで付属品を差し替えて お客様に届けた。販売開始から数ヵ月経ったが、売り上げは芳しくなかった。調べてみると、作物を置いてもらっているスーパーの売り場担当の方が「他の作物との違い」や「移す手順」を知らないので、お客様に上手く説明出来ず 結局普通の作物を買っていってしまう事が分かった。もともと、品種改良した分だけ原価が高めで、売値も高めの設定としていたことも有り、「このままでは売れない」と悩み始めることになる。

店頭での販売が伸び悩む中、営業担当が奔走してくれて まとめ買いしてくれるお客様が幾つか現れた。このまとめ買いのお陰で販売数が増え、田畑も広げてフル稼働状態となった。しかし、お客様からのクレームもそれに合わせて増えていった。結局、お客様でも売れ残りが多発してしまい 追加の注文は途絶えてしまった。

お客様からのクレーム。使い勝手に関するもの、買ってすぐ使えなくなった、上手く味が移らない 等々。他の味も幾つか欲しい、という声も。これらの声に応えるため、開発初期のメンバー (7 ～ 8 人の有志) に加えて新しいメンバーに入ってもらった。昔一緒に仕事をしていたメンバー、他の農家やメーカーで働いていたメンバー。新しい知恵を授かりながら、まずは「お客様のご不満の声を全て解消するものにしよう」という目標を立てて、第2世代品の開発に取り掛かった。

使い勝手についての改善点。味を移したい相手と（近くにいるのに）上手くつながらない。相手とつながっても、味が移らない、味が移るのが遅い。

作物側で出来る事。「味のバリエーションを増やす」「味が移るスピードを上げる」「手順書の見直し」等々。とにかく、出来る事は片っ端から試してみて、第2世代品の販売を開始した。少しだけお客様からのクレームが減り、バリエーションを増やした事で 少し売

り上げを持ち直したが決定打とはならず。

味を移したい相手で出来る事。作物側に「操作ボタン」が無いため、使い勝手向上には相手側 がカギと考えた。

当時、急激に普及していたスマートフォン。これを操作に利用しようと スマホアプリを開発。しかし初期バージョンは外注に丸投げで 動作不具合多発。とても世の中に送り出せない出来だったため、自社開発に切り替えた。スマホアプリ側を見直す事で、作物との間のやり取りがスムーズになり、味の移るスピードも向上。やっと使える状態になり、お客様へ届ける事が出来た。この頃から、お客様の声は「使いやすい」「気に入った」「まだ使いにくい」「まだつながらない。」「もう少し早いと良い」と賛否か分かれてきた。

少し盛り上がり始めた感触はあるが、まだまだやることは多いと感じた。

六年前

販売拡大のための模索は続く。TV 通販業界のカリスマにも直接売り込みに行ったが、「商品を買ってもらうこと」の前に、「商品を良く知ってもらうこと」の奥深さと難しさを教えてもらった。私はエンジニアであるが、広報や広告が得意な仲間と会って 一緒に仕事をする機会が増えた。地方局の情報番組、首都圏のラジオ番組、農業業界紙や専門雑誌 等々。飛び込んで行けそうなところには、何処にでも脚を運んだ。

認知度は確実に上がってきたが課題も見えてきた。商品の使い方や良さを説明すれば、「良いね」「面白いね」と言ってもらえるけれど、全国のお客様 一人一人と会ってご説明する訳には行かない。

この頃、商品作物の「味を移す」技術を他の用途に使えないか？という問い合わせが増えてきていた。問い合わせ主の声。「味を移す技術をイチから仕込むと、原価が高くなる」「挿すだけで 味を移す機能を搭載したい」「味以外のモノも移したい」

この頃はまだピンと来ていなかったが、まさに「IoT」への活用の話。

モノがインターネットにつながる技術を搭載するためには、国の認可が必要。一つ一つのモノに ネットへつながる技術を搭載する事は現実的では無かったが、我々の商品であれば挿すだけで使えて 認可取得が不要。この点を上手く PR 出来れば、様々なモノで活用してもらえそうな予感がした。課題は「挿すだけでインターネットにつながる」技術があることだけでなく、移したいモノ によっては「少しずつ使い方を工夫 (= 設定変更) する必要有り」とお知らせすること。これを問い合わせしてくれた人、一人一人に伝えて行く事は大変な労力である事は目に見えていた。

偉い人の計らいで、コンサルタントにこの悩みを聞いてもらった（実は IoT という用語は、このコンサルタントから聞いたのが初めてだった）。こちらのやりたい事と悩みを率直に伝えて、何度かディスカッションした後、一つの提案を受けた。単なる商品紹介だけではなく、「詳細な技術情報を載せた web サイトの開設」である。確かに、お客様から問い合わせ頂いた時の流れとして、毎回「何が出来るのか」という説明を一通りした後で、

お客様から「これをこうしたい」と聞く形がほとんど。これだと、折角会って話をしても大半の時間を「何が出来るのか」の説明に時間を割かれてしまう。情報サイトがあれば最初の「何が出来るのか」の部分の説明が不要になり、理解してもらえれば web サイトに掲載した技術情報を使って、即 検討に入ってもらえる。お客様側で不明点が有り、打ち合わせすることになっても「こういう事は出来ないのか?」という核心に直ぐ入れるメリットもあった。

コンサルタントと会話した 4ヵ月後、試験的に情報サイトをオープン。サイトの運営は、隣の土地の協力会社にお願いすることにした。彼らからは「情報発信だけでなく、情報交換する場にしたい」との提案が有り、「フォーラム」というスペースも開設した。

その様なものに全く馴染みが無い我々も、「先ずはやってみよう」の精神で進めていたが、次第にその役割が分かってくる。フォーラムに書き込まれた、一つの疑問。他のユーザーが、その問いかけに答える。課題が解決する。それが情報として蓄積されて、また他ユーザーが活用できるコミュニティサイトの役割も持つ様になった。このサイトはどんどんコンテンツを増強して、日本語&英語に続いて中国語（簡体中文）もサポート。もはや、試験的というレベルではない本格的なものに進化していった。サイトを起点とした ユーザー向けイベント、SNS でのつぶやき、ユーザー間の交流を通して、活用される用途の「裾野」が広がり、人脈も広がっていく実感も有った。このサイトには「活用事例紹介」もあり、ここに「私の製品を載せて PR したい」というオファーをもらえる様にもなった。まさに、ユーザーやお客様との「共存共栄」。

そんな有る日。コアなメンバーから「同人誌を作って、イベントで配布しようよ」との提案が。同人誌とは、別名「薄い本」と呼ばれるもの（ということを初めて知った）。思いがけない提案だったが、TV での広告宣伝や技術情報サイト開設など 前例の無いことばかりやってきた我々には、それを止める理由など無かった。それ以降 毎年夏に 同人誌発行&イベント出展 というルーチンが始まる事になった。この同人誌は、イベント以外でも秋葉原などの一部店舗でも不定期に配布される様になった。SNS ではコアなユーザー（仲間）がそれを情報発信&拡散してくれる。お客様への細かい説明をする手間を省きたい、という理由で始めた技術情報サイトが こんな形に進化するとは。これは嬉しい誤算であった。

五年前

「挿すだけで簡単 IoT」と称した技術情報 web サイト。

商品作物の「味を移す」技術を、他の用途にでも活用できないか?という活動は、徐々に結果が出始めていた。

コンビニにあるプリント端末、レジ情報収集。自動車車内で使う端末の情報更新。

歯医者の口腔内撮影データの転送。工場内の装置の稼働ログ収集。

デジタルサイネージのコンテンツ更新。アルコール検出用ガジェットの情報収集。

定点撮影システムからのデータ自動送信。街灯監視カメラからのイベントデータ収集。

狙い通りの用途もあるが、想像していなかったが言われれば納得の用途も。ユーザーが、我々の技術情報サイトを見て「思い付いて、考えて、作って」。そして、これらの活用事例を 技術情報サイトに掲載する。またそれを見たユーザーが新しい用途に活用出来ないか考える。また仲間が増えていく。良いサイクルが回り始めた実感が有った。

四年前

その傍らで、本来の用途（作物の味を移す）では大きな壁にぶち当たっていた。インターネットにつながる「モノ」が激増してきた事で、「味が移るスピードが遅い」「スマートフォンとつながらない」というクレームが増え、内部ソフトウェアでの改良やスマホアプリの改良では 根本解決できない程になりつつあった。結果、一度購入頂いたユーザーのリピート購入率が伸び悩み。これは IoT 用途の方も同じで「簡単に導入出来て便利だけれど、スピードが遅くて お客様が増えない」との声。

この「スピードの絶対値を上げる」ため、商品作物のハードウェアレベルからの見直しがスタート。作物の何処がハードウェアか？というと 味を移すための仕組みの部分に 特殊な IC チップを使っているからである。IC チップ、実は９年前まで住んでいた土地の仲間が開発したもの。かつて、「世界初」のモノを作りながら 品種改良のための開発費がもらえず 泣く泣く断念した仲間。新しい土地に移ってからも 仲間との関係は切れず つながっていた。昔の仲間を訪ねて「スピードを上げたい。それも半端なく」と相談した。その答えは予想通りのものであった。「味が移るスピードは上げられるけれど、弊害がたくさん有る」。そして「開発にはたくさんの費用が掛かる」と。開発費については、我々でなんとか捻出するしかないが、技術検討は1日でも早く進めてもらえる様にお願いした。

三年前

数ヵ月後、昔の仲間から検討結果の報告を受けた。スピードは５倍以上になるが、やはり多くの弊害が見込まれるとの事だった。「カタログスペックが大幅にアップする」と謳える事で、我々からの開発費も出しやすくなる。兎に角、弊害を最小限にする事が命題となった。昔の仲間という事も有り、IC チップの細かいレベルの設計レヴューにも首を突っ込ませてもらった。安易に妥協点を見出だす事よりも、ブレずに目標値を目指す事にこだわり、何度も議論は紛糾した。でも、９年前の悔しさを共有している 昔の仲間が意地を見せる。

世界トップレベルの性能が見込める 設計データが完成。数ヵ月後、待望の試作品を入手。机上シミュレーション では計り知れない「産みの苦しみ」との戦いの始まりだが、手に取って評価出来る「モノ」が有れば何とかなる筈。メーカー魂を見せる時が来た。

何せ今回は ハードウェアレベルからの見直しにより「スピードの絶対値を上げる」こと

が目的。これにはスピードを制御するソフトウェア側をハードウェアの特性を最大限に引き出せる様に作り込むが必要。この作り込みを担当する仲間は、９年前この商品作物の品種改良ネタを思い付いて デモを作った仲間（ソフトウェア担当のリーダー）。彼は実の子どもが生まれた日、商品作物の第1回目の試作品が届いたのが同じ頃という、自称「煌めく空 の産みの親」である。今回の性能改善でも、彼が一番初めに「スピードを世界最高レベルにあげて、他が追い付けないところまで持って行けば良い」と言った張本人。周りからは「自分でそんなにハードル上げてしまって大丈夫?」と言われるが、「まあ、何とかする」と返す。特性を引き出すための作り込み。直ぐ目標値の半分くらいまでは到達するが、ここからが伸びない。色々なパラメータを調整して見るがダメ。ハードウェア担当を巻き込んでの調査と打ち合わせが重ねられた。

そして「産みの親」から一報が届く。とうとう目標の性能に到達したとのこと。スピード性能は 従来品の約7倍（産みの親のドヤ顔が目に浮かぶ）。

これなら「カタログスペック」だけで他を圧倒出来る。

二年前

「スピードの絶対値」を上げた商品作物が遂に完成。「カタログスペック」だけで他を圧倒出来るというワクワク感とから商品のラベルも一新。

商品ラベルは商品の顔。ネット販売では 外装（パッケージ）ではなく、商品本体のイメージが載るケースがほとんど。従来品と並んだ時にも 商品品番以外に「明らかに区別できる」様になった。

ここまで紆余曲折有ったが、新商品もようやく船出。自分の子どもの様に 思いを込めて、手を掛けて、世の中の人の役に立つモノを送り出す。本当に素晴らしい経験をさせてもらった。我が子を思う親の気持ちは強く、重い。それが強すぎると、その子のためにならない事も有る。そんな時は、周りの助けをもらえば良い。簡単そうだけれど、意外と難しい事に気づかせてくれたこの商品に感謝。さて、子ども達は 親をどう思っているのかな。この商品「煌めく空」自身が、その思いを話してくれることは無いけれど、商品を買って使ってくれたユーザー（仲間）が それを代弁してくれる。その声を大切にして、また思いを込めた新商品を送り出したいと思う。

上岡 裕一／ykami (@YuichiKamioka)
自称、FlashAir の「義理の父」。
2020 年の東京オリンピックに（裏方としても）FlashAir を送り込むべく活動中。

悲しみは思い出とともに

エヌ氏

エヌ氏は悩んでいた。ここはとある会社のシステム LSI 開発センター。2015 年の冬のことである。開発会議に呼び出されたエヌ氏は製品事業部からある開発計画を聞かされた。無線 LAN 搭載 SD カード向けの LSI を新規に開発してほしいという話だ。目標は高い。単に新機能に対応するだけでなく、ユーザーが体感できる実効性能も高いものを目指すという目標だ。

具体的には、SD カードの状態で

1. **UHS(Ultra High Speed)-I に対応し、読み書き速度は上位モデル並みのメモリ性能を達成する。**
2. **IEEE 802.11b/g/n 対応の無線 LAN 機能とマイコンをワンチップ化し、チャネルボンディングに対応して 100Mbps のスループットを出す。**
3. **イベント会場のように無線 LAN アクセスポイントが乱立する混雑した場所での接続性能を向上させる。**
4. **マイコンの処理能力を向上し、高度な暗号通信や数学関数を使用可能にする。**
5. **キャッシュメモリを増量し、Lua スクリプトで使用可能なメモリも増やす。**
6. **I2C、UART(シリアル通信)、PWM を SD カード端子で使えるようにする。**
7. **以上を、記録するだけの SD メモリカードと同程度の電流で実現する。**

といった内容である。

考え抜かれた仕様だった。限られた電源制約とサイズの中で譲れないものは何かにこだわった仕様である。単なる機能の追加や強化ではなく、ちゃんと体感できる性能を達成したいという思いが伝わってくる。技術者の悲願である。だが簡単には実現できないだろうという物言わぬ合意もまた全員の中にあった。例えば、スループット 100Mbps は、従来製品は 15~16Mbps くらいだと聞いているのでおよそ 6 倍の目標である。単に新しい規格に対応しただけではそんな性能は出ない。ハードウェア開発者とソフトウェア開発者が互いにベストを尽くしてやっと達成できる目標である。ならベストを尽くそう。そんな思いで開発がスタートした。

そこでエヌ氏は開発リーダーに指名される。主担当はデジタル設計。RF 部の開発リーダーとともに LSI 開発を推進する。解決すべき技術課題は何か、ボトルネックとなる工程は？特に重いのは無線 LAN と高速インタフェースの共存と低消費電流化。SD メモリ全体の電流を抑えなければならないのに、NAND メモリの消費電流もメモリアクセスの高速化に伴って増加する。LSI に許される電流は残り少ない。課題の洗い出しと適任者の候補案、課題解決案を複数出して絞り込みアーキテクチャの骨子を作成、開発スケジュールとリソースをまとめて会議に臨んだ。人月の神話は信じていないが、何をするにも人が足り

ない。既存設計や IP の踏襲すべき部分は踏襲し、新たな技術課題に取り組む。最善策はこうだ。日程は守る。共に知恵を出し合いたい。そのような内容だったと思う。責任者が大人の判断を行い開発計画が了承された。そして SD カード開発チームと LSI 開発チームの間でフェース to フェースの会議を幾度となく行い、精力的に開発が進められた。

LSI 設計チームの努力が実り、苦労の末完成したテストチップを SD カードに搭載しての動作試験。ハードウェア設計者とソフトウェア設計者のバトル。目標性能をどうやって引き出すか。製品スケジュールは守れるのか。譲れないところは譲らないが、技術検討や対外交渉は率先して行ってくれる。恵まれた職場だったとエヌ氏は語る。

当初の目標に妥協はなかった。100Mbps の目標については無線環境の問題や通信相手の性能には依存するものの、最後は測定コマンドのソースコードを調べて 12MByte/s の M は 1024x1024 であることを確認し、12MByte/s は 96Mbps ではなく

12MByte/s ＝ 12 x 1024 x 1024 x 8 bit = 100,663,296 bit/s ＞ 100Mbps

であることが判明して一安心する一幕もあった。

そんなエヌ氏には心残りがある。開発で毎日夜遅くなる自分を心身ともに支えてくれた最愛の妻を、新型SDカードの発売を待たずに亡くしたことだ。子宮の病気。前兆はあった。普段から弱音を吐かない人だった。早く歩けない。会社の健康診断では鉄分の不足が指摘されるのみ。更年期障害かもと本人は笑う。不正出血がみられおなかが膨れてきた。これはさすがにおかしいと病院に行った時には、もう手遅れだった。我慢できる痛みでも病気は取り返しがつかないところまで進行していることがある。我慢強い人は病院に行くほどではないという。病院にトラウマがある人もいる。妻も若いころに行った産婦人科医で精神的につらい思いをしたそうで、行かなかったことを後悔とともに謝罪されてしまった…。若さゆえに進行が早く、歩けなくなって２週間後に息を引き取った妻。

ここに書くことではないかもしれない。でも書かせてほしい。普段こんな話はしないけど、最終号ということで少しセンチになっている。見知らぬ誰かを救いたいのかもしれない。どうか世の中のお父さん、お母さん、この記事を読んだ皆さん、病院が苦手な家族や友人や同僚に、体調が悪いときは念のため病院に行くことを強く薦めてほしい。様子がおかしければ無理やりにでも病院に連れていくことを真剣に検討してほしい。

読者の方々とご家族、同僚の皆さんのご健康を心からお祈り申し上げます。

エヌ氏
LSI 設計技術者。
FlashAir Developers 閉鎖を複雑な気持ちで見守る。

FlashAir 同人誌の長い編集後記

余熱

6 年続いた FlashAir 同人誌も今回で一区切りということで、需要があるかどうか分かりませんが自分の FlashAir 関連の思い出を少し書いてみようかと思います。

FlashAir との出会い

僕が FlashAir の「ちょっと変わった使い方」について知ったのは 2014 年の 9 月 21 日、ソラちゃん[1]が「基板に絵を描いちゃった」とツイートしているのを見かけたことがきっかけでした。何気なくソラちゃんのアカウントをフォローしたところ、数日後メッセージが届き、曰く「Maker Faire Tokyo で基板を配りたいのだが取扱説明書に書く注意文が思いつかない、アドバイスが欲しい」とのことでした。改めてソラちゃんのツイートを見てみると、この基板は FlashAir の GPIO 機能を使うための基板だと分かりました。FlashAir は東芝の半導体事業部で作っている製品ですが、当時の僕は同じ会社のヘルスケア事業部でリストバンド型の活動量計の開発を行っていたのです。つまりソラちゃんの中の人も同じ社内の人に違いありません。Twitter の DM に「詳しいことはメールで返信します」と返したところ、メールの署名に会社名が書かれていました。こちらから「僕も同じ会社です」と返したところ「驚きました。社内がざわついています」というメールが返ってきました。数日後、社内便でサンプルの FlashAir が送付していただき、これが僕と FlashAir の出会いになりました。

FlashAir 評価ボード「Airio（えありお）」

FlashAir の GPIO 機能を手軽に試せるボードが必要だと感じたので、会社の同僚に手伝ってもらって基板を設計することにしました。11 月 2 日に「秋これ」というイベントが開催されるということで、そこに合わせて FlashAir 評価ボード「Airio(えありお)」を設計しました（図 1 ）。基板データのガーバアウトは 10 月 13 日ということで、FlashAir を知ってから 1 ヶ月経っていない状況ですでに評価ボードの設計が出来上がっていたことになります。

図 1: FlashAir 評価ボード Airio

Makre Faire Tokyo 2014

10 月 27 日、MFT に向け横浜の肉バルで企画を練ることになり、この企画会に呼んでいただきました。「FlashAir を題材とした同人誌を配って MFT で配るべき！」と密かに思っていたため、この企画会で提案しようと考えていました。タイミングを間違えてボツになってしまうのは避けたいので、皆さんが案を出し切ったタイミングで提案したところ、割とすんなり受け入れて貰えました。企画会が行われたのが MFT の 1 ヶ月前ということで、で

1 https://twitter.com/Hirameki_Sora

きるだけシンプルに作ろうと考えていましたが、最終的に 32 ページの大作になりました。おおよそ 2 週間程度でここまでの記事を書くことができ、FlashAir 関係者の熱量を感じました。これは面白いものができた！ということで、MFT 向けに思い切って 1000 部を作成。大変好評だということで以降毎年 FlashAir 同人誌を作ることになりました。

サークル「空と月」とアルコールガジェット「TISPY」

2015 年 6 月 6 日のお酒のイベント「酒っと」で同人誌を売りたいと Pochio さんから声がかかり、サークル「空と月」を結成しました。お酒のイベントとということでアルコールセンサと FlashAir を組み合わせたガジェット「酔ったー」を製作しました（図2）。

その後、2015 年 8 月に部署異動をし正式に FlashAir チームになりました。10 月には「酔ったー」を発展させたアルコールガジェット TISPY の企画を開始し、翌 2016 年 3 月にクラウドファンディングを開始しました。社内外でも珍しい事例であったため、色々なメディアに取り上げて頂くことができました。2019 年には TISPY2 として再びクラウドファンディングをしています（図3）。

図 2: 空と月の同人誌

そんな FlashAir Developers が閉鎖します

FlashAir Developers の閉鎖を 2019 年 6 月 12 日に Interop の余熱セミナー（通称ヨネミナー）にて発表させて頂きました。ながしまさんが新公開 API fa.pwm を使った「蛍の光」演奏デモを作ってくださったので、悔いの残らない形で閉鎖を発表出来たのがとてもありがたかったです。Developers が終わるということは同人誌も終わるということ。それはやっぱり、寂しいけれど、でも、終わらないものもたくさんあります。同人誌は「熱い想いをそのまま伝える」という編集方針で、素材の良さを引き出すように心がけました。その想いは、終わりません。FlashAir とともに歩んできた 6 年間。いろんな人とお会いすることができ、沢山のありがたい経験をさせて頂きました。その経験も、閉鎖されません。

図 3: アルコールガジェット TISPY2

これからは空と月で FlashAir 関連の製作を続けて行くと思います。空と月は FlashAir 同人誌よりも発刊ペースが速く、すでに 8 冊も同人誌を作っています。同人の活動には終わりがないので、いつまでも続けられそうです。まだまだ色々な切り口で FlashAir のことを語れると思っていますので、引き続きよろしくお願いします。

余熱 (@yone2_net)
Developers は 9/17 で閉鎖ですが、またどこかで会えるかも？

宮内

頻繁にボドゲオフに通うカード / ボードゲーマー。今回 Flash Fight のルール検討・テストプレイに参加し、ゲーム制作の難しさを実感。「『テーマは守る』『面白くもする』両方やらなくちゃあならないのが作り手のつらいところだな」

じむ

企業キャラクタはプロにお金を払っているので、予算や企画が無くなれば消えゆく運命ですが、個人（中の人）で描いているので消えませんねー。w これからもブログや、別の同人誌で細々とソラちゃんを描いていきますよー。

■ FlashAir Doujinshi 6 - FlashAir の同人誌 6

2019 年 8 月 3 日　第 1 版第 1 刷発行

著者 : 高田 / 伊藤 晋朗 / 土居 / Pochio / じむ / あおいさや / ながしま / 余熱 / GPS_NMEA_JP / いしかわゆうじ / めむ / 寺田 賢司 / 綾瀬 ヒロ / こたまご a.k.a ひなたん / 高橋 祐介 / Takehiro Yamaguchi / 長船 俊 / 福屋 新吾 / ほげじゅん / 高瀬 秀樹 / 上岡 裕一 / エヌ氏 / 宮内
表紙・本文イラスト : じむ
表紙デザイン : 余熱 / じむ
編集 : 余熱 / Pochio / ながしま
発行 : FlashAir Developers
連絡先 : support@flashair-developers.com
印刷 : 株式会社 プリントパック

https://flashair-developers.com