修士論文

USB システムの構築と応用に関する研究

The investingation of construction and application of USB systems

報　告　者

学籍番号：1225069
氏名：谷脇宇京

指　導　教　員

綿森　道夫　准教授

令和 2 年 2 月 10 日

高知工科大学　電子・光システム工学コース

目次

第 1 章　序論

1.1　研究の背景

「モノがカタチになることは少ない」。これは私の指導教員の綿森先生の言葉である。このことはいったい何を意味しているのであろうか。そもそも最初から形になることを目指さない研究であれば、形になることはあり得ない。私の研究分野はいわゆる電子工作を通してかつての電子立国の成長の過程を追体験しようという側面があり、比較的形になりやすい分野であると思っている。それでも、極度に IOT が発展した現在において、そして安価な 100 円ショップですら製品化された電子機器類があふれている現状において、大学の小さな研究室に所属する一個人が、いくら自由に研究思索する時間があるといっても開発環境一つとっても明らかに現在存在するあまたの企業に比べて劣る現実の中で何かを生むことが可能なのであろうか。何かを生むことを義務付ければ、そもそも大学での電子工作や、回路を作ることが許されなくなるのではなかろうか。電子系の研究室で回路の設計製作が自然淘汰されるというのも、生きるためとはいえ悲しい気がしないでもない。これからの電子立国は、純回路的には大学ではなく企業が自前で人を育てていくのであろうか。

では何故大学の研究室で物が形になることがまれなのであろうか。予算の問題、人員の問題、そして伝統的に積み上げてきたものの蓄積量による問題などが考えられる。何よりも大切なのはアイディアであるというが、これはある面正しくて、ある面勝手な暴言である。アイディアはある程度形になりそうな土壌の上でしか意味をなさず、現状とかけ離れていれば夢を見ているに等しい。そもそも電気が支えている生活をしている現代において、全く新しく、しかもそれを形にできる土壌を有し、明らかに新しい局面を生み出すようなアイディアが、例えば一年に一回程度生まれるものであろうか。とても信じられない。このようなことを考えてみると、最初に挙げた、「モノがカタチになることは少ない」という言葉の意味にも納得できる気がする。

今回の研究の開発段階において、一歩ずつ物を組み上げていく作業を経ている途中で、ある時、綿森先生がモノがカタチになりそうな瞬間を感じたという。もちろんその感じ方には程度差があって、全員が同じ感覚を得るとは思えない。しかしながら、カタチになることは少ないと実感している先生自身がカタチになりそうな予感を得たことに気をよくし、その開発過程を順を追って示していこうと思っている。その過程をたどっていくと、思わぬところからモノというものが生まれてくることを実感できて我ながら楽しく研究開発をさせていただいたと思う。

本研究の概要を一言でいえば、大学で学生実験に使用するオシロスコープが旧型のために、取得した測定データを USB メモリーに保存する際、利用可能な USB メモリーの容量に限度があり、4GB 以下の容量の USB メモリーでなければ認識できないことがあることを別の手段で回避したことにある。USB メモリーの容量制限は FAT システムの上限に由来するものであると思われるが、既に 4GB 以下の USB メモリーは入手困難な状況になりつつある。測定データが保存できなければ学生実験に使用する際には大きな障害となる。さりとて全てのオシロスコープをより大容量メモリーに対応した新機種に切り替えることは、仮に別の方法でこの問題が回避できるならば無駄といえるであろう。新しい新機能が必要となる新たな実験課題が作られるまでは、現状の装置で十分であるといえると思う。私の場合、本来、大学院に入学後、USB の通信規格と通信の仕組みを調べていた [1, 2] こと

がこのオシロスコープの USB メモリー問題に結びつくきっかけとなった。

最近ではパソコンに接続して新たな機能を追加する機器はほとんどの場合 USB 端子を利用している。専用のソフトをパソコンにインストールして USB に接続した機器を操作する。オシロスコープのような測定器や、ロジックアナライザー、アナログ回路測定器などが USB に接続することで、安価にそして気軽に使えるようになっている。身近なところではプリンター、携帯電話や音楽スピーカーなども USB で接続できる。このように USB に接続してパソコンで操作する電子機器類の場合、一般の電子工作愛好家やホビーストなどが開発できるものなのであろうか。単純に考えても、パソコン側のソフトの開発、それも専用のデバイスドライバーを扱う形式のものを作らなければならないし、パソコン側の USB 端子に接続するためのハードウェアーも製作しなければならない。そもそも USB ケーブルを流れている信号電流の形式と通信のプロトコルを知らなければ USB 機器を設計製作するなんて夢のまた夢である。もちろんある種のデバイスを使えば、USB 形式を見かけ上別の信号形式、例えば扱いやすい RS-232C や I^2C のような別の形式に変換してくれることがある。これはこれで便利であり、比較的簡単に USB 機器を設計製作できる利点を有していると思っている。しかしながら USB は便利な接続端子であり、今後簡単にはすたれることは起きそうにないことから、たとえ実用的な物ができなくても USB を勉強しておくことは将来において意味があるのではないかと思って取り組み始めた。

ここに一つの偶然がある。例の学生実験で使用しているオシロスコープであるが、何故か説明書には将来の拡張用で利用できないと書いてありながら USB のポートが存在していた。このポートとは別に USB メモリーをさすための USB type A ホストポートも存在しているので、ひょっとすると拡張用の USB type B クライアントポートも本当は通信できるのではないかと想像した。パソコンに USB ケーブルで接続すると、当然、デバイスドライバーがないのでデバイスドライバー要求画面となる。このことはパソコン側の USB の認識要求信号（エニュミレーションという）にオシロスコープが答えているのではないかという疑いを抱かせ、ケーブルの途中を切断し、USB ケーブルを流れている信号電流を観測した。無事に認識要求プロセスをとらえることができ、USB ケーブルを利用して、パソコン側にオシロスコープのデータを転送するプログラムの開発作業に取り組むことになった。

以上が本研究の背景であるが、開発のクライマックスは学生実験用のオシロスコープを USB ケーブルでパソコンに接続し、自作のソフトで実験データをパソコンに転送できるようになった後、さらなる開発過程において生じる。そしてこの開発過程こそが綿森先生の言うモノがカタチになる瞬間をとらえた一場面であると信じ、その流れを修士論文に記載していこうと思う。

1.2 研究の概要

本研究は学生実験室に置いてある旧型のオシロスコープの USB 端子とパソコンとを USB ケーブルで結び、パソコン側からオシロスコープを制御したり、取得したデータを USB 経由でパソコンに転送し、表示するとともに CSV 形式で保存できるようなソフトを開発することが第一の目的である。背景においても簡単に記述したがオシロスコープの USB 端子は生きており、USB 通信が可能であることが分かった。また、USB 通信に対応した後継機オシロスコープが存在していることもあり、その機器用のドライバが無償配布されている。そこで、USB の通信プロトコルの規格を学び、後継機用の専用ドライバの動作を監視することで、通信内容は分からないもののオシロスコープを

動作させられそうなデータの形式を知ることができたと考えた。自由にオシロスコープへデータを送る手段として、パソコン側のデバイスドライバーを利用方法が明確な汎用 USB ドライバーである WinUSB につなぎかえることにした。WinUSB は専用のクラス番号を想定してインストール用ファイル (inf ファイル) を用意しているので、オシロスコープ自身の製造メーカーコード (Vendor ID) と型番 (Product ID) を書き加えることで登録を行った。

利用するデバイスドライバーを決定し、ドライバー登録をすませたら windows パソコン側でデータを送受信するソフトを作る。送信はテキストボックスにセットしたコードを送ることにして、受信に関してはいつ応答が返ってくるのか分からないので別スレッドを立てて信号を拾うこととした。本当に大変なのはこの作業であり、最初は送っているデータ列にどんな意味があるのか、帰ってきているデータには何の意味があるのか全く分からない状態だった。転機になったのは送受信データが部分的に文字列コマンドであることを理解したときである。すなわち、オシロスコープをリモートで動かす時は ASCII コードの”RUN”を特定の手順で送ればいと分かり、受け取っているデータの意味も大よその当たりをつけることができた。これにより、専用ドライバが送っていたコマンドの意味が分かり、無事自作ソフトの開発につながった。

本学の中間発表の前にはこのソフトの最初のバージョンができていて、実際に学生実験で使用してもらって不具合などを洗い出すことにした。中間発表においては何故か今後の研究計画も述べなければならないので今後の研究方針を先生と打ち合わせした。先生は現状で十分で実地に動作テストを行って残りの期間を過ごせばよい。偶然にも左右されたが、早い段階で結果が出ることもあるので敢えて今後の研究計画など発表しないで、実地テストの結果を見るのでいいのではないかという意見であったが、それでは残り 1 年を遊んで暮らすような印象となり具合が悪い。何らかの研究計画をとってつけたように発表することとした。

先生との打ち合わせの結果、現状ではオシロスコープとパソコンを USB ケーブルで結んで通信を行っているが、このケーブルを切断して無線化して通信を行うこととした。この無線化は技術的には大きな意味を持っているが、実際の利用という観点からすると何故無線化するかという意義が分かりにくい。無線化するということは、オシロスコープ周辺に USB 通信を管理する USB ホストデバイスを設計製作しなければならないことである。パソコンのような大掛かりな機器・装置では簡単であったことも、PIC のような組み込み系の CPU を使って実現するなどそれだけで大変であることが容易に想像できる。事実 PIC の製造元のマイクロチップ社のライブラリーには USB ホストと OTG のライブラリーが付録のようについているが、雑誌等の製作記事で解説されたことがなくほとんど無人の野を行くようなものであった。この時、USB ホストのソースを読むために役に立ったのが USB 通信のプロトコルの説明であり、規格というものはきちんと読んで把握すれば、いろいろな応用が可能になることを身に染みて思った一件であった。以下に無線化プロセスがどのような形になったのかを示して、研究の概要を締めようと思う。

無線化プロセスにおいて最初に検討したことは、無線の種類である。しかしながらこれは案外あっさりと決まった。すなわち、既にパソコン側に受信モジュールがあるか、USB に接続するカンタンなドングルのようなもので受信できると便利であるということである。この結果、BlueTooth を採用することとした。この時、無線通信においては必ずしも通信ができるとは限らず、パソコン側からデータ取得のコマンドを送っても無事にデータが取得できないかもしれないという可能性を考慮した。そこで、たとえ無線通信が失敗しても、取得したデータが使えるような方式を考えた。それはオ

シロスコープ側の機器に SD カードをつけることであり、無線通信に失敗してもデータを保存しているという構想である。一歩進めて無線通信が確立していようがいまいが、オシロスコープのデータを SD カードに保存できる仕様とした。例えばオシロスコープから USB メモリーにデータを保存するためには、オシロスコープ本体のいろいろなボタンを押してモードを選び、場合によってはファイル名まで設定しなければならない。それに対して、ちょっとデータが欲しい時、開発した機器を USB 端子に接続し、ボタンを押せばデータが取得できると大変便利である。

この開発した端末でボタンを押すとデータが取得できる仕組みを構築する上で問題になったことはファイルの命名である。話の展開上、ファイル名は機器のボタンを押された際に機器が一意に決定する必要がある。しかしながら単なる連続番号であれば、どのデータが何時のものなのか、本当に必要なデータはどれなのかを判断することが厄介である。そこで機器側にリアルタイムクロックを搭載し、原則としてデータ取得のボタンを押したときにリアルタイムクロックの日付からフォルダを作り、その中に連続番号を有する一連のファイルを保存することとした。バッテリーが完全放電してリアルタイムクロックの情報が失われてしまっても、番外フォルダにとりあえず連番で放り込んでおくことにする。RTC を動かすため PIC の電源はバッテリーから直接供給し、設計した機器の電源を切ったときには、PIC は状態変化割り込みで省電力状態へと移行する。

このような設計方針の下、オシロスコープでデータを測定中にそのデータが必要になったときは製作した機器を接続し、ボタンを押せばファイル保存が行える無線化機器を開発した。これが本当にカタチになったモノといえるのかどうかはわからないが、今までにないものであることは間違いない。第 2 章以降でこの機器を開発していく際に見舞われた想定外の出来事やハプニング、開発の流れについて順に説明していこうと思う。

第 2 章　USB 版オシロデータ取得プログラムの開発

2.1　USB プロトコルの確認

まず始めに、本研究での主題は USB 通信の構造について学ぶことである。通信プロトコルを一通り学んだ後、実際の USB の通信を観測するため、オシロスコープのケーブルにメスのコネクタを接続し、間にロジックアナライザをはさんで、再びオスのコネクタを接続する。ここからオス-メスケーブルでパソコンにつないで、ケーブルを流れている信号をロジックアナライザで観測した。その結果、図 1 の信号が得られた。プロトコル通りに解釈すると LowSpeed の通信で IN,DATA0,ACK の三つのパケットを送っていることがわかる。これは IN トランザクションと呼ばれる一塊のデータである [1]。

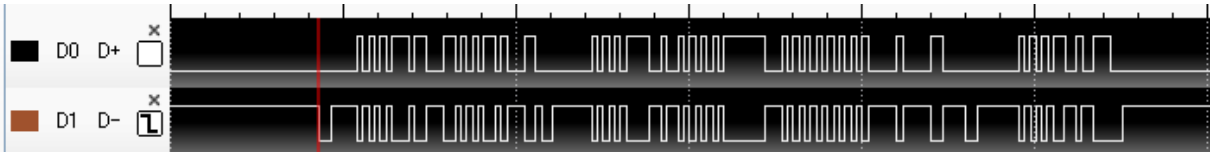


図 1: USB マウスの信号

このようにロジックアナライザを使って実際の通信を解析する事は可能であったが、通信全体を記録する事は難しく、周期的な信号以外を狙って観測する事も難しいために USB デバイスの解析には不向きであると判断できた。次に通信を監視する手段として、フィルタドライバを用いてホストドライバの動きを見る手段を考えた。

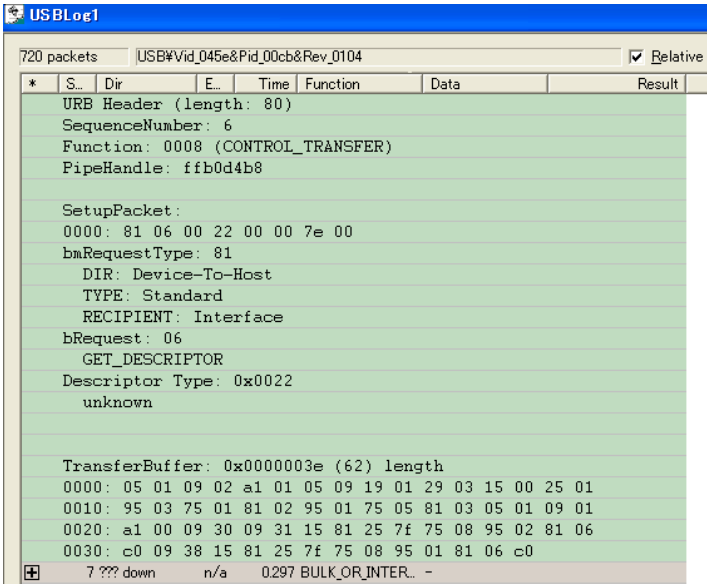


図 2: マウスドライバの監視

図 2 はホストドライバが標準命令の getdescriptor を用いてデバイス情報を読み出す様子である。この内容は HID 特有のレポートディスクリプタと呼ばれるもので、何ビット目に何のデータを配置

するのかを大まかに伝えるものである [13]。このマウスはボタンの押下情報と X 方向、Y 方向の移動情報、ホイールの回転量を DATA パケットで伝えてくれる事が分かる。ロジックアナライザに比べるとドライバが呼ばれるまでの通信は見る事ができないものの、処理の順番が把握できて通信の内容が判断しやすくなっている。そのため、本研究での USB デバイス解析は主にこの手段を用いて進めた。

2.2 オシロスコープと PC の接続

図 3: DS-5102 外観

図 4: 背面の USB ポート

図 3 に示すのは現在電子系実験室で利用されているオシロスコープ (DS-5102) である。このオシロスコープの裏面には図 4 のように USB 端子が存在し、説明書ではこの端子は使用されていないとかかれている。しかし、実際に PC に接続してみると図 5 に示す通り不明なデバイスとして認識された。これはパソコンからの標準リクエストをデバイスが正しく処理しているという事であり、USB デバイスとして使用できる可能性があるという事でもあった。勿論オシロスコープの USB 通信モジュールが勝手に標準リクエストを処理しているだけで、オシロスコープとして意味のある処理がなされているかどうかは不明である。

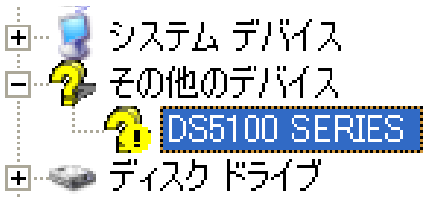


図 5: 不明なデバイスとして認識される

調査のために目をつけたのは同シリーズの後継機 (DS-5105B) である。こちらは USB 通信に対応しているらしく、動作させるための専用ドライバとアプリケーションが配布されていた。これらを使って動作させられないかと考えた。

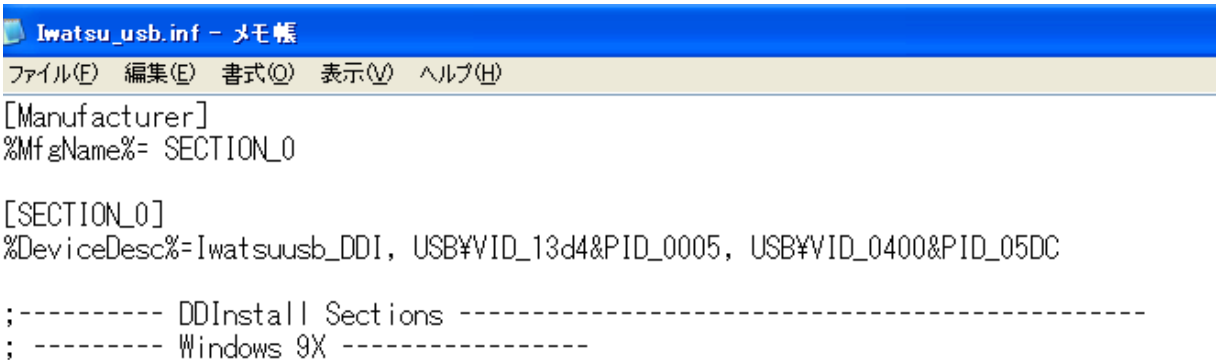


図 6: 改変した inf ファイル

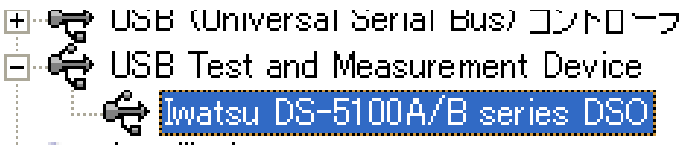


図 7: 専用ドライバをインストール

図 6 のように後継機用のドライバのインストール用情報ファイル (inf ファイル) に DS-5102 の VID と PID を書き加え、強制的に DS-5102 と接続した（図 7 参照)。専用のアプリケーションを立ち上げると、接続扱いにはなったもののデータ取得は全て失敗に終わった。オシロスコープが反応するのは画面更新の状態を切り替える命令のみだった。

2.3 オシロスコープ通信プロトコルの推測

専用ドライバの動作を図 8 に示す。標準リクエストを使って基本的なデバイスの情報を読み出した後はバルク転送でデータを送受信している事が分かる。やり取りされているデータはその特徴から三種類に分類することができた。

一つ目は 0101 からはじまる 12 バイト固定長のデータであり、二つ目は 0202 からはじまるデータであり、そして三つめは 3a からはじまる可変長のデータである。三つめの可変長データは 4 バイト程度の短いものもあり、サイズ情報も末尾情報も含まれていないように見えた。観察してみると 3a

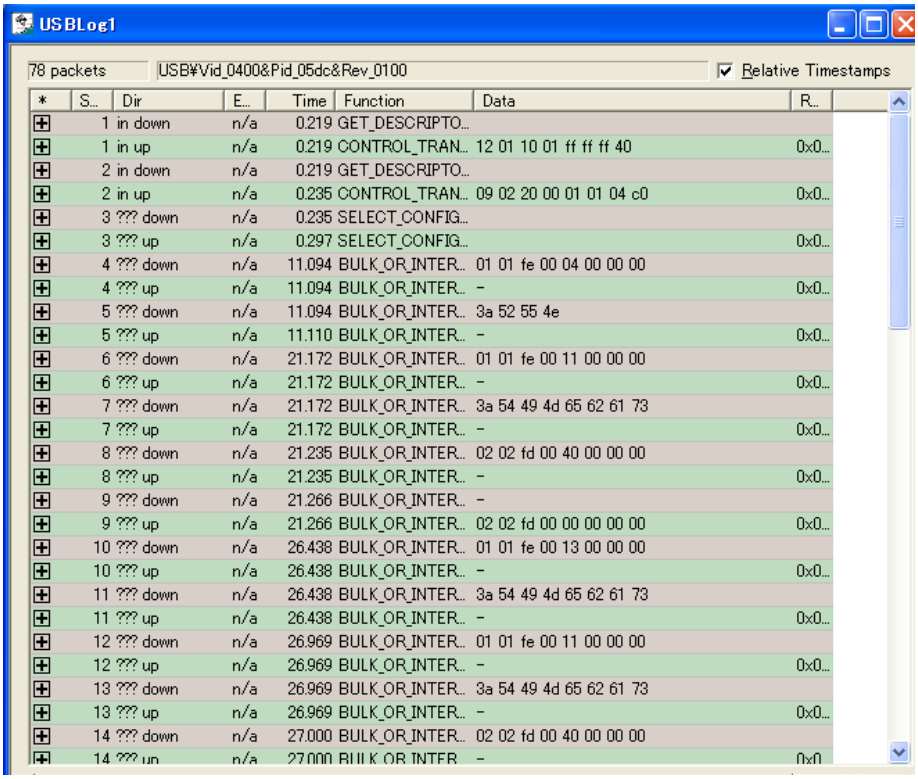


図 8: 専用ドライバの動作

のデータの前には必ず 0101 のデータが送られていることがわかり、0101 のデータの 5 バイト目には 03 のデータのサイズと同じ値が入っている事が見つけられた。このことから 3a のデータがコマンド、0101 のデータがサイズ情報を持ったコマンド送信予告だと推測することができた。同様にして 0202 の命令が読み取り予告だと推測することができる。また、0202 の命令の 5 バイト目は必ず 0x40 であったため、ホスト側が用意しているバッファのサイズか何かだろうと推察した。

これらの推測が正しいかを確認するため自由にデータを送れるソフトウェアを制作した。このソフトを使ってデータの送受信を行っている様子を図 9 に示す。この時点で専用ドライバでは読み出せていなかった情報が読み出せるようになっていた。データ部分を変更すると無反応になるだけだったが、サイズ情報と推測される部分を変更するとドライバが固まってしまうという問題が起こった。この事からより重大なエラーが起こっていると判断でき、推測を裏付ける根拠の一つにすることができた。

2.4　シリアルポートの調査

オシロスコープは USB 端子とは別に RS232C 用の端子も持っている。こちらは公式に動作させるアプリケーションがあったので調査の一環として動作を確認した。図 10 がソフトの外観である。USB シリアル変換ケーブルを用いた通信では波形データを受け取れていることが分かる。

この時の USB-RS232C 変換ケーブルのドライバの動きを図 11 に示す。送信データのほとんどが 3a から始まっており、USB 通信のコマンド部分とよく似ていた。異なっていたのは全てのデータの末尾が 0a で終わっているところである。受信データも始まりはバラバラであるが終了文字は 0x0a

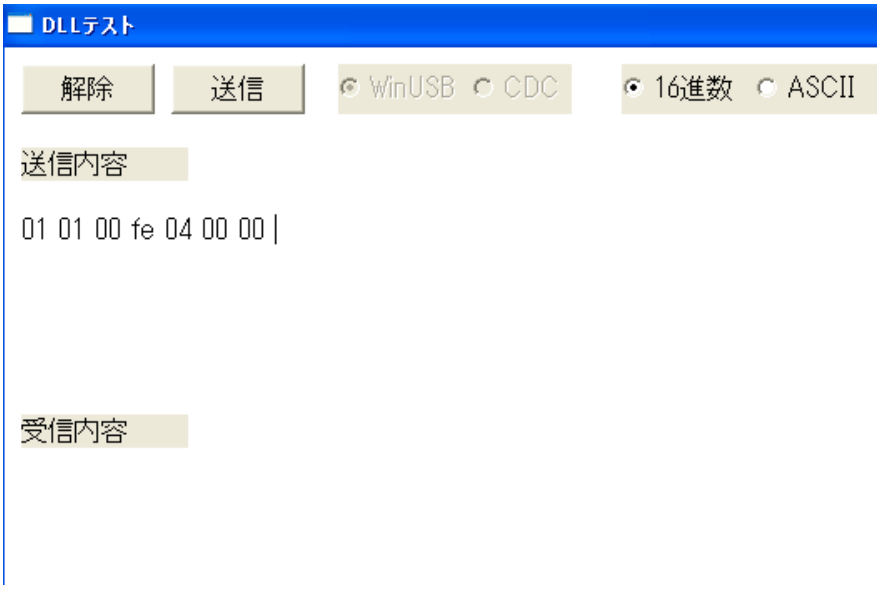


図 9: コマンドテストプログラム

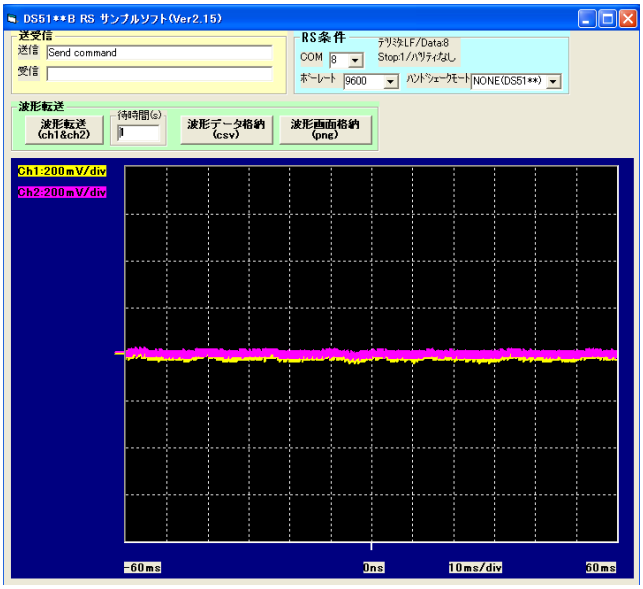


図 10: シリアルポートでの情報取得

で統一されている。更に 0x0a は ASCII 文字の改行コードであることを考えると、可変長のコマンド部分は文字列なのではないかと推測ができた。

ドライバの通信に登場するコマンドを ASCII 文字列へ変換した様子を図 12 に示す。結果「:CHAN1:DISP?」となっており、チャンネル 1 の表示状態を確認するコマンドであることが分かる。このようにして判明したコマンドは、末尾の 0x0a を消した上でサイズ情報を加えて送信予告を行うことで USB 通信で利用できることがテストプログラムにより確認できた。

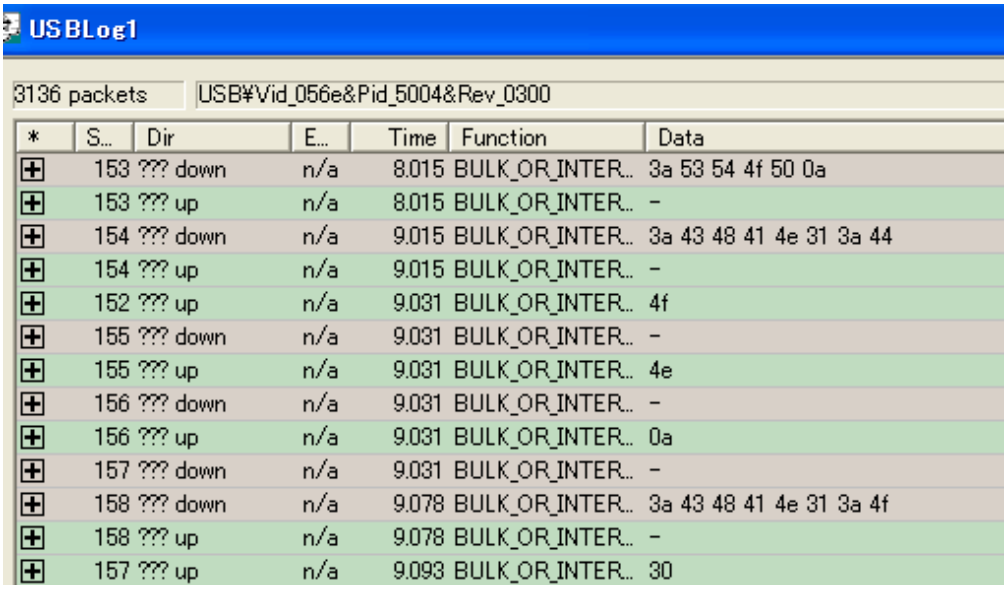


図 11: RS232C 変換ケーブルのホストドライバ

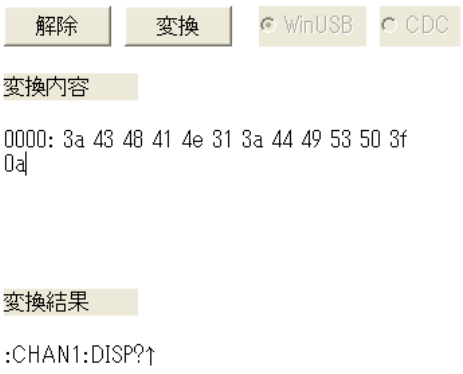


図 12: ASCII 文字への変換

2.5 USB 版プログラムの制作

テストプログラムを制作するのに使用したコンパイラは GCC で、統合開発環境は CodeBlocks を使用した。開発の核になったドライバは WinDDK(Driver Development Kit) に含まれている WinUSB（Windows 用の汎用 USB デバイスドライバ）である。これは一般に USB デバイスドライバ開発に用いられるドライバで、用意された API を使うことで指定したパイプを利用して USB 通信を行うことができる。今回は更新の簡便さを考えて、WinUSB ドライバをそのまま利用したアプリケーションの制作を行った。接続処理としては主に SetupAPI 関数群を利用して以下の処理を行う [7]。

1. SetupDiGetClassDevs() 関数の第一引数に WinUSB デバイスの GUID を指定し、該当するデバイスのテーブルを作成する。

2. SetupDiEnumDeviceInfo() 関数を使い、指定したインデックスのデバイス情報を DEVINFO_DATA 構造体として受け取る。
3. 受け取った情報を元に SetupDiGetDeviceRegistryProperty() 関数を実行する。引数には SPDRP_HARDWAREID を指定し、ハードウェア ID のサイズを受け取る。
4. 適切なサイズのバッファを用意してもう一度 SPDRP_HARDWAREID を指定し、ハードウェア ID を受け取る。
5. 実験室に存在する二種類のオシロスコープの内、どちらかの ID にマッチしているかを確認する。マッチしていなければインデックスを増やして手順 2 に戻る。
6. 対象デバイスのインデックス番号とテーブル情報を用いて SetupDiEnumDeviceInterfaces() 関数を実行し、DEVICE_INTERFACE_DATA 構造体を受け取る。
7. 受け取った構造体を使って SetupDiGetDeviceInterfaceDetail() 関数を実行し、DEVICE_INTERFACE_DETAIL_DATA 構造体を受け取る。
8. 受け取った構造体の中の DevicePath を対象に CreateFile() 関数を実行する。
9. ファイルハンドルが得られれば、それを元に WinUsb_Initialize() 関数を実行して、データの送受信に必要な WinUSB インターフェースハンドルを作成する。

より汎用的なドライバを作成するならばここでコントロール転送を行い、デバイスが提示するパイプ情報に従うべきであるが、今回は通信相手が DS-5102 である事を前提としてバルク転送のみを行った。これは WinUsb_WritePipe() 関数、WinUsb_ReadPipe() 関数によって実現することができる。専用ドライバの動作を模倣して送信用のパイプは 0x01、受信用のパイプは 0x82 とした。

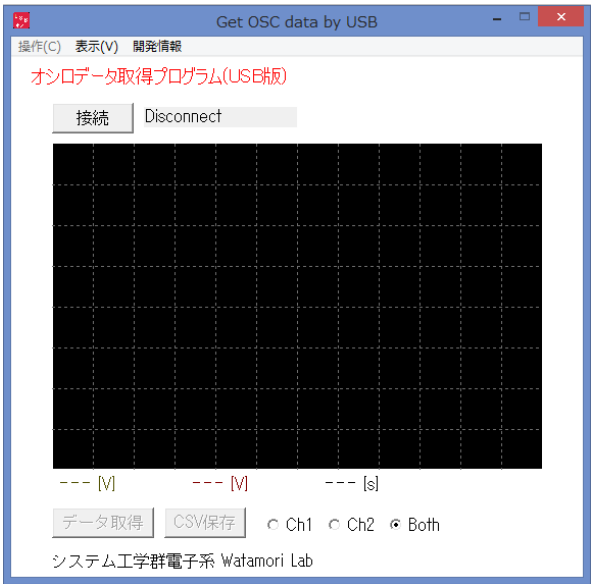


図 13: USB 版データ取得ソフトウェア

作成したソフトウェアを図 13 に示す。「接続」ボタンを押すと準備を行いオシロスコープとの USB 通信を確立する。オシロスコープを取り外した場合は自動で非接続状態へと移ることができる。これは RegisterDeviceNotification() 関数でデバイスの接続、取り外しを検知できるようにしているためである。

「データ取得」ボタンを押すと以下の手順でグラフ表示に必要な情報を読み出し、プロットを行う。このボタンは、接続されていないときは EnableWindow() 関数を用いて無効化している。

1.「:TRIG:STAT?」で更新状態を確認する。
2. 停止中でなければ「:STOP」を送り、データ取得中は測定を停止させる。
3.「:TIM:FORM?」で XY モードか YT モードかを判別する。
4.「:TIM:OFFS?」「:TIM:SCAL?」で時間軸の情報を取得する。
5.「:CHANx:OFFS?」「:CHANx:SCAL?」で電圧軸の情報を取得する。
6.「:WAV:DATA? CHANx」で各チャンネルのプロット位置情報を取得する。
7.「:KEYLOCK DIS」でオシロスコープを手動操作できる状態に戻す。

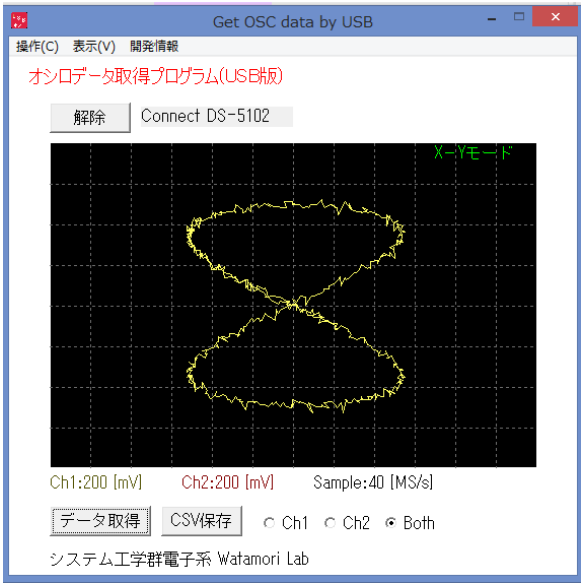


図 14: XY モードへの対応

グラフの描画は Win32API を使用してメモリ上にビットマップデータを作り、線描画でグラフを描画してからウィンドウに転写する形をとっている [9] [14]。グラフ描画のプログラムは USB 重量計を作成したときに作ったグラフ描画クラスを再利用した。実物のオシロスコープに合わせて目盛りや線の色合いは調整している。また、図 14 のように XY モードの描画を行う必要があったため、点毎に X 座標、Y 座標の情報を保存する形に改造した。

「CSV 保存」ボタンを押すとデータを CSV 形式で出力することができる。出力先のファイル指定にはコモンダイアログを使用する。このボタンはデータ未取得のときは無効化している。

他にも判明したコマンドを使えば RUN・STOP の状態を切り替えたり、XY モード・YT モードを切り替えたり出来るが、ボタンを増やしすぎると実験で使用する際に混乱を招くかもしれない。そこでリソースファイル上でメニューを構築して、メニューアイテム「操作」を用意した。これにより後の機能追加を含めて、外観を変えないカタチでマイナーな機能を増やすことができた。このメニュー項目は非接続状態では無効化している。

ここまでで実験に必要な機能はできたと判断し、テストも兼ねて電子系実験室の各パソコンに配布

を行った。

2.6　コントロールパネルの制作

オシロスコープの情報を読み取るだけでなく、時間軸や電圧軸の情報はパソコン側からも設定することができる。そこで、オシロスコープの軸設定を行うコントロールパネルを制作した。

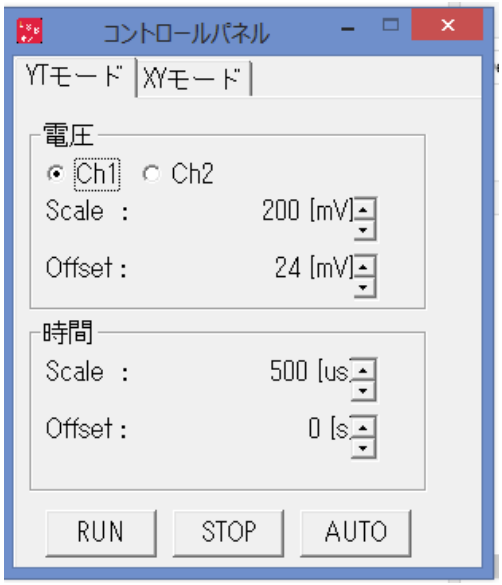


図 15: コントロールパネルの外観

制作したコントロールパネルの外観を図 15 に示す。タブコントロールで YT モードと XY モードの切り替え、スピンコントロールで電圧軸と時間軸の幅とオフセットの操作ができる。スピンコントロールは CreateUpDownWindow() 関数で作成できるコントロールである。デフォルト動作では整数値を扱うもので、増減量を変更したとしても等間隔になってしまう。一方でオシロスコープに設定する軸幅は 50mV、100mV、200mV、500mV のように指数的に変化させたい上、できる限りキリの良い幅に設定したい。そのため、デフォルト処理は使わずに、WM_NOTIFY メッセージ中に UDN_DELTAPOS の通知を拾ってプログラムで直接値を設定した。幅の調整は画面を見ながらしたいだろうと考えて、軸を変更する度に画面状態を再取得している。

一通りの処理を作ってから気づいた事ではあるが、この操作には恐ろしい問題が存在した。パソコンから軸の設定命令を出しても即座に反映されるわけではなく、数十ミリ秒待たなければ正しい軸情報が拾えないのである。軸変更前の波形を表示しようが、変更後の波形を表示しようが大きな問題はない。しかし、軸設定の読み出し処理と波形データの読み出し処理の間で軸の更新が反映されてしまった場合、点の表示位置情報と軸情報から計算して電圧を求めるこのアプリケーションは測定電圧値を誤認してしまうのである。おまけの機能を追加した結果、不正データを取る可能性が生まれるなどという事は絶対にあってはいけない。発生する可能性が低いとはいえエラー検知ができない問題なので、軸情報の設定後は余裕を持って大きなスリープをはさむ事にした。結果、スピンコントロールを押す度にスリープが挟まり、非常に使いづらい機能となってしまった。次案として、UDN_DELTAPOS メッセージで反映処理を行うのではなく、少しの間ユーザーがスピンコントロー

ルの操作を止めたタイミングを見計らって軸設定と波形データの再取得を行うようなプログラムにしていきたいと考えた。しかし、「実験室にある二種類のオシロスコープでそれぞれ設定できる幅の最大値や最小値が異なっている」問題が見つかった他、「二種類のオシロスコープで XY モード中の Sample 時間の設定方法が違う」問題等々、改善するべき様々なハードルが見つかった。これらの問題を時間をかけて解決したとしても、使い心地が悪く、最悪間違ったデータを取りかねないリスクを合わせて考えた結果、実験室のパソコンにアプリケーションを配布する直前で隠す事にした機能である。

2.7 機能追加

実際に実験系の授業で使用して頂いた結果、一応実用できる形にはなっているとの評価をいただいた。フィードバックとして計算機能やカーソル機能などの意見をもらえたため、機能を追加した。

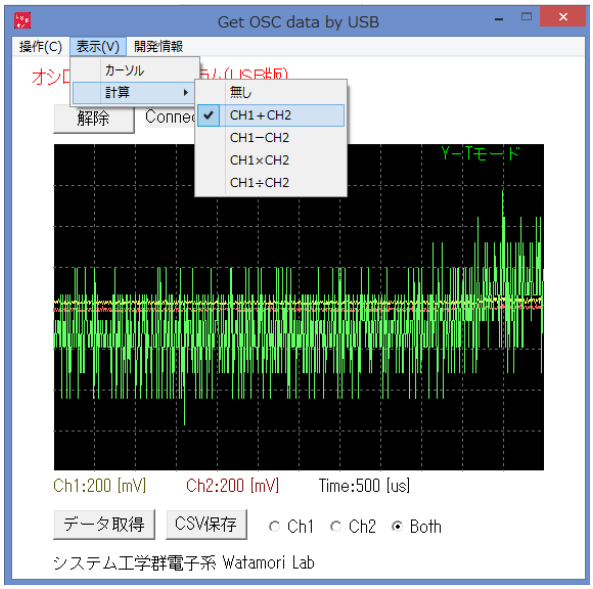


図 16: 計算軸の追加

計算軸機能を ON にした様子を図 16 に示す。2 チャンネル分の情報がそろっている時のみ三軸目が表示される。表示の最大値と最小値をチャンネル 1 や 2 に合わせる方法も取ることができたが、この機能を使う場合は波形の概形をみたい場合がほとんどであろうと考え、計算データの最大値と最小値を基準にしている。

カーソル機能を ON にした様子を図 17 に示す。カーソル情報を表示する必要があったが、機能追加で見た目が複雑になることを嫌って、カーソル機能を ON にしている時だけウィンドウサイズを大きくすることにした。グラフ範囲をクリックするとクリック位置に近いカーソルを操作できる。また各チャンネルのグループボックス範囲をクリックするとカーソルの操作チャンネルを切り替えることができる。各チャンネルの 8 個の情報表示用スタティックコントロールはそれぞれ、チャンネル名付きのグループボックスを親ウィンドウとして登録している。グループボックスのプロシージャは SetWindowLong() 関数の第二引数に GWL_WNDPROC を指定することで入れ替えを行っており [8]、登録された独自のプロシージャが WM_CTLCOLORSTATIC メッセージを拾って

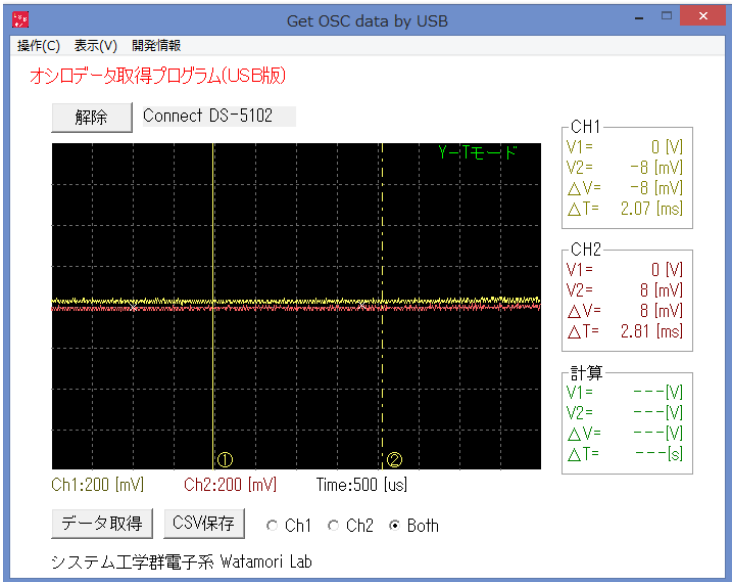


図 17: カーソル機能の追加

各スタティックコントロールの背景色と文字色を変更している。またグループボックスを覆う形で SS_NOTIFY 属性付きの透明なスタティックコントロールを配置しており、グループボックスのプロシージャ内で WM_COMMAND メッセージを拾うことでボタンのように扱っている。こうした透明なスタティックコントロールは、親ウィンドウの WM_CTLCOLORSTATIC 内で LPARAM が対象のウィンドウハンドルだった時に GetStockObject(NULL_BRUSH) を返す事で実現できる。ウィンドウの透明化は WS_VISIBLE 属性と混同されがちだが、WS_VISIBLE 属性を無くした場合はウィンドウとしての機能が全て消えてクリック検知すら不可能になるため注意が必要である。実際にカーソル非表示モードでは ShowWindow() 関数でグループボックスの WS_VISIBLE 属性を無くすことで、カーソル関係の機能を無効化している。

オシロスコープは波形情報として点のプロット位置情報を返してくれるため、配布したバージョンではプログラム内でも位置情報をそのまま利用していた。そして、CSV ファイル出力の瞬間だけ実際の電圧計算を行うという構造だったのだが、この機能のためにはグラフは実際の電圧データを持っている方が都合が良い。そうなるとグラフの最大値と最小値は固定値ではいけなくなり、グラフ描画機能自体に機能を追加する必要が出てくる。地味な機能ではあるがプログラムの構造を大きく変えることになった要因であった。

図 18: 開発情報パネル

更に綿森先生の要望により開発情報パネルをつけることにした（図 18 参照）。このパネルはメニューの開発情報によって呼び出される。通常のパソコンソフトではヘルプやバージョン情報といったパネルが存在するが、このパネルにはバージョン情報ではなく、開発環境や利用した主要な API 関数が示されている。これは、学生実験中に何げなくこのパネルを開いた人に、このソフトは本学で実際に学生の手によって作られたことを感じてもらうために存在しているとのことである。このパネルを見た人の内の何人かに自分でもこの様なソフトが作れるのではなかろうかと思ってもらえるといいという意図である。もちろん、この修士論文を見れば開発過程の抜粋がいわば開発のヒントのように書かれており、おそらく違うタイプの実用ソフトになると思われるが、自作ソフトを開発していくイメージをとらえることができる。しかしながらそれはあくまでもソフトを使って興味をもってこの修士論文にたどり着くといういわば次の段階で、最初はこのような開発情報パネルに開発の入り口を示しておく方がよいという思いである。この様な理由で開発情報パネルをつけることにした。

(a) USB 版　　(b) BT 版

図 19: アイコン

また、このソフト用のアイコンも製作した（図 19 参照）。アイコンは全体的に赤い色で統一されているのが USB ケーブルを使って学生実験室のオシロスコープと接続して使うソフト用であり、後述する BlueTooth を使ってオシロスコープと接続して使うソフトには Blue というだけあって青い色で統一されたアイコンを割り当てている。一目で区別できるためである。

第 3 章　USB ホスト機能を有する BlueTooth 送信ユニットの開発

3.1　USB フレームワーク（デバイス）の解析

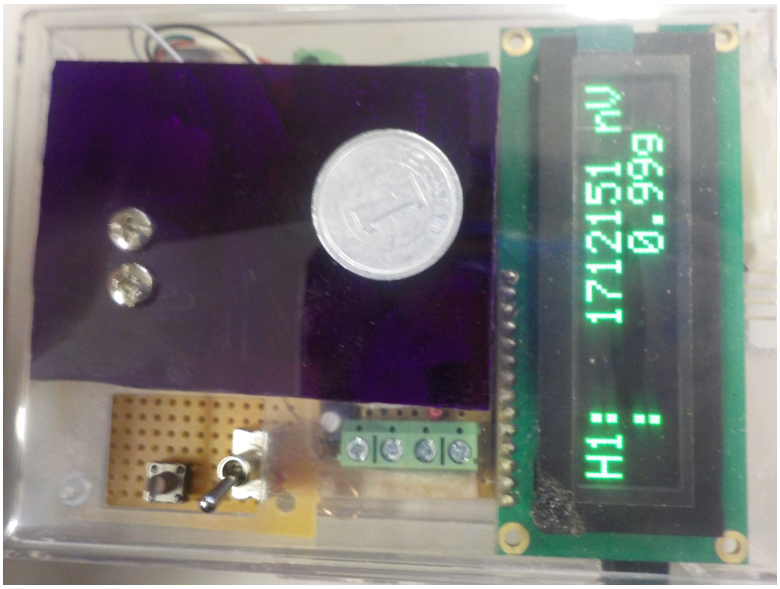


図 20: 自作重量計 (USB デバイス)

序論でも述べたが、無線化のためには BlueTooth 送信方式を採用した。これはデータの受け手であるパソコン側に専用のハードウェア回路を作って取り付けることを避けたいためである。BlueTooth ユニットは、最近のパソコンでは内蔵しているものも存在し、外付けでも市販の BlueTooth ドングルで対応できる。この利点が大きいので無線通信距離が短いという欠点には目をつぶった。BlueTooth での通信を行うためには、PIC 等のマイコンに USB ホストの機能を持たせてオシロスコープから情報を受け取らなければならない。その後、受け取った情報を BlueTooth ユニットでパソコン側に送信する。PIC にはマイクロチップが提供する USB フレームワークが存在するため、これを利用できないかと考えた。必要なのはホスト側であるが、手始めにこれまで作品に利用したことのあるデバイス側のフレームワーク解析からはじめた [2] [11]。チップとして選択したのは PIC18F14K50 である。選択したデモプログラムは図 20 に示す USB 重量計に使用した HIDCustomDemo である。この自作重量計は大学院に入学して最初に設計・製作したもので、24 ビットの分解能をもつ A/D コンバータチップ HX711 を使用してロードセルに重量（圧力）をかけた時に生じる僅かな電圧変化を検知するものである。内部電源 3V または USB バスからの 5V 電圧を定電圧化してホイートストーンブリッジ構成の回路に供給する。図の様に 1 円玉の重量 1g が測定可能である。発生電圧は 1.71mV である。測定した値は USB を経由してホストコンピュータに渡されて利用される訳であるが、今回は重量の時間変化をグラフにしても仕方ないので、手でかけた重量（圧力を）スイッチとする一種のゲームを作った。この作成したソフト自体に関しては修士論文の主題から外れるので、ここでは説明しない。

USB フレームワークは 1 つのソースコードで様々な状況に対応できるようにチップ情報や USB デ

バイス構成情報を元に#ifdef で有効なソースコードを切り替えている。そのため、関数内外を問わず様々なチップ向けの文が入り乱れており、人間がプログラムの流れを追いかける上では非常に厄介な構造になっていた。そこで使用チップを 18F14K50 に絞り、最低限必要なコードに解体を行った。USB フレームワークを構成するファイルの内、HID デバイス用関数をまとめたヘッダや抽象化のためのヘッダなどの細分化されているものは一個のファイルにまとめ、大量に宣言されている関数は使用するもの以外を削除した。union によって大量の別名定義が行われている構造体は、単純な型に直して実際にどのレジスタの何番目のビットを操作しているのかを分かるようにした。最終的には usb フレームワークの内、エニュミレーション処理、接続・切断・エラー等のステート処理、コントロール転送における各ステージの処理、標準リクエストの処理以外は特に必要ないことが分かり、簡単なソースコードに直すことができた。

3.2 USB フレームワーク（ホスト）の解析

BlueTooth 送信ユニットを作る上で採用したチップは PIC24FJ64GB002 である [4]。最初にスタンドアローンでマウスを認識してマウス移動を表示するプログラムを書き込み、HID マウスのホストデモが正常に動くことを確認した。その後、MicrochipSolutions に含まれる Host-MCHPUSB-GenericDriverDemo を雛形として、オシロスコープ操作プログラムを制作した。このデモは同じく配布されている Device-MCHPUSB-GenericDriverDemo を書き込んだ USB デバイスを受けるためのホストプログラムである。デモの内容自体は利用できないが、他のホストデモに比べてパーサ処理等が無い分シンプルで改変がしやすいと考えて雛形に採用した。GenericDriverDemo を改変する上で重要な部分を以下に示す。

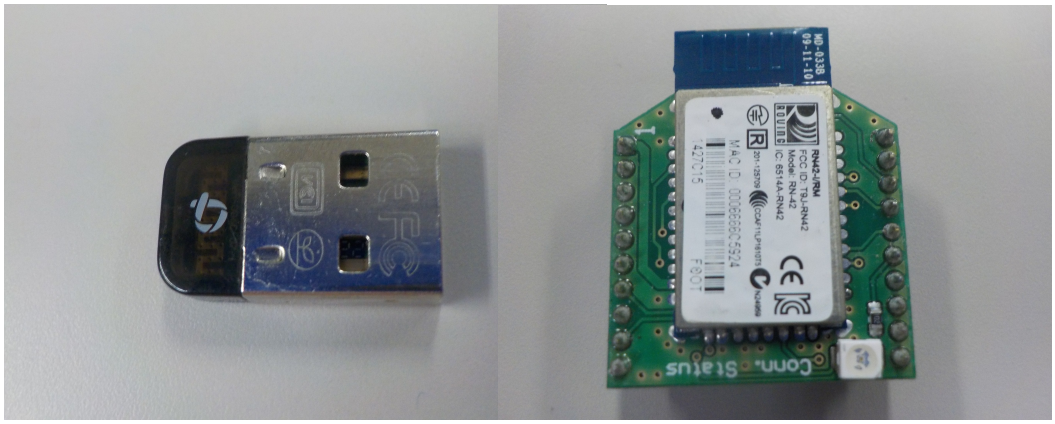
- USBHostInit() を最初に実行する
- USBHostTasks() を定期的に実行する
- usbTPL に接続許可するクラスかハードウェア ID を記述する
- USBHostGenericWrite() でデータ送信準備
- USBHostGenericRead() でデータ受信準備
- USBApplicationEventHandler で接続、切断イベント検知

そのままだと送受信パイプは同じものが使われてしまうので、今回はオシロスコープの仕様に合わせて送信時は 0x01、受信時は 0x82 を対象にするように USBHostGenericWrite() 関数と USBHostGenericRead() 関数の内容を書き換えている。

3.3 テスト用 PIC プログラムの制作

USB 版オシロスコープデータ取得プログラムではオシロスコープ操作コマンドのサイズをアプリケーション部分で調べて送信予告や受信予告を送っていた。BlueTooth 版ではこの機能は自作ホストに持たせたいと考えた。つまりパソコンは文字列コマンドのみを送り、受け取った PIC がオシロスコープに送信予告を送ってから文字列コマンドを送るということである。このときコマンドに「?」が含まれていれば読み取り命令と判断して、コマンド送信後に読み取り予告をオシロスコープに送信する。読み取りが成功したら PIC はパソコンにデータを送って命令待ち状態へと戻る。オシロス

コープがハングアップしていた時のためにウォッチドッグカウンタを用意しており、いつまで経っても命令待ち状態に戻らなければリセット動作をするようにした。



(a) BT-MicroEDR1X　　(b) RN42XVP

図 21: 使用した BlueTooth モジュール

パソコン側の BlueTooth モジュールは図 21a のドングルを使用した。PIC をつなぐ BlueTooth モジュールには図 21b の RN42XVP を使用した [16]。RN42XVP は UART 通信でデータを送受信できる。PIC が USB ホストの処理をしている間に UART でデータを次々に送られると、すぐに UART モジュールがエラー状態になってしまうので、UART を受信した時は一旦割り込み処理でリングバッファにデータを蓄積させておき、PIC の状態が命令待ちに戻ってからリングバッファを読み出すようにしている。また RN42XVP 自体が BlueTooth 通信が確立されると UART で接続情報を送ってくるため、これをオシロスコープに対するコマンドと誤認して PIC 経由でオシロスコープにデータを送ってしまわないように先頭文字が「:」でなければ読み捨てる事にした。このモジュールで通信を行うときの注意点として、文字データ'N' を送信すると BlueTooth 通信の切断コマンドになってしまう事をあげておきたい。シリアル通信を行う上で TeraTerm を使用してコマンドを送っていたのだが、TeraTerm は入力された端から即座にデータを送っていく。「:RUN」と入力した途端に BlueTooth 通信が中断されるため、結構長い間 BlueTooth 通信が不安定なのだと勘違いしていた。順番に送られてくる:RUN の文字列の中の 4 番目の文字'N' に反応していたのである。TeraTerm を使うなら貼り付け機能を、自作プログラムを使うなら WriteFile() 関数等で一度にデータ文字列を送るとこの問題は回避できる。

無線通信で PIC とパソコン間のデータを受け渡す構想をした時点で、PIC 側にもデータを保存できるような仕組みがほしいと考えていた。そこで SD カードに FAT 形式でファイルを構築し、データを保存できるプログラムを作った。また PIC のリアルタイムクロックモジュールを利用してデータを取得した時間をファイル名にできるようにもした [10]。これは、無線の通信が 100% 成功するとは限らないことを考慮してである。無線通信においては通信距離が離れると急激に通信失敗の確率が高くなる。測定コマンドや停止コマンドの様な情報量の少ない通信であればまだ失敗の確率は低いが、データ取得というサイズの大きいデータを扱う一番大切な処理においては再送機能がなければ途中でデータの欠損が起きる可能性が高くなる。実際に採用した BlueTooth においては再送機能が存

在しているが、通信状況が改善されなければいつまでも失敗し続けることには変わらず、通信時間がかかりすぎると各モジュールのタイムアウトや、それに伴うリセット処理によりデータ欠損が起きる可能性が生じる。通信環境が悪い場合の備えはプログラムとして組み込んでおきたいと考えた。オシロスコープから取得したデータをパソコン側に送信するだけではなく、SD カードに念のために保存しておく構成にしておけば、通信失敗の状況に柔軟に対応できる。この際に SD カードにデータをたとえ一時的であれ保存する場合には、ファイル名を PIC 側で勝手に決めて保存しなければならない。この時、ただファイル名の衝突を避けるための連番のファイル名にしてしまうと、どのファイルがいつとったデータなのか全く分からない状態になってしまうため、リアルタイムクロックを搭載して時刻情報をフォルダ名やファイル名に使う事にした。

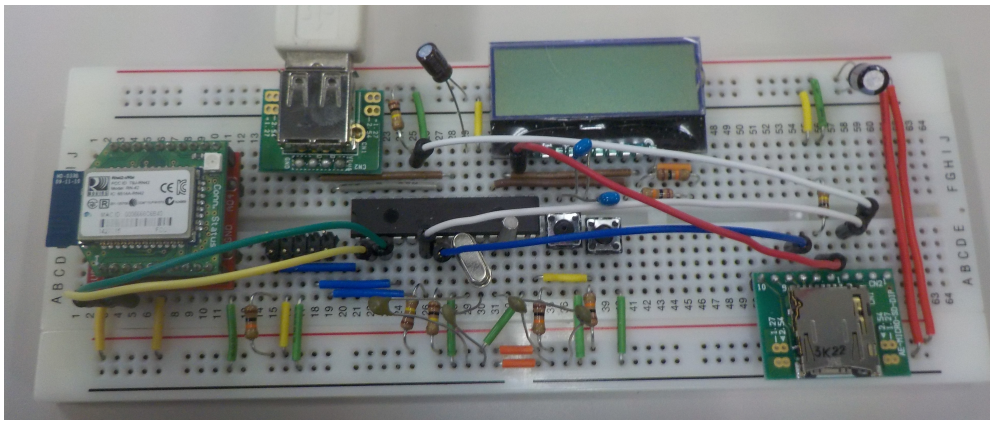


図 22: PIC プログラムのテスト回路

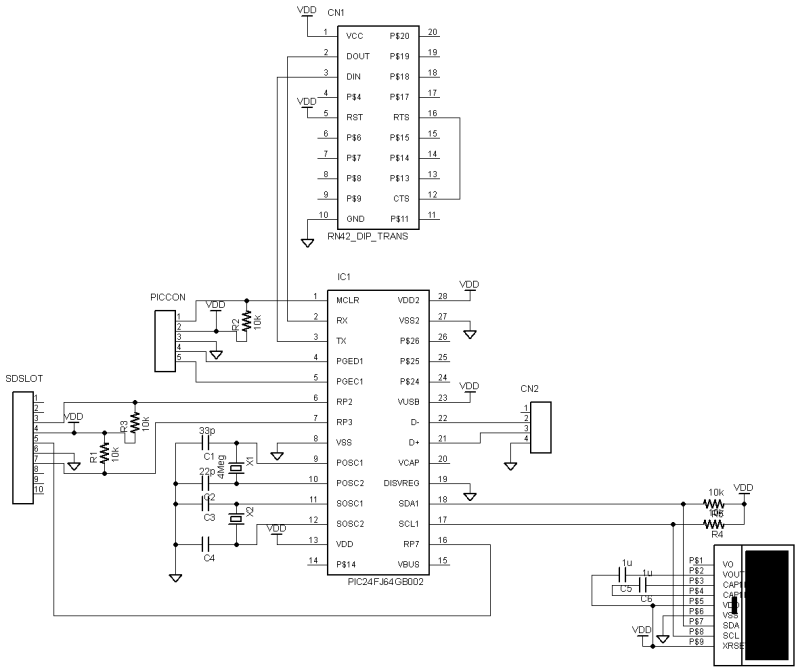


図 23: PIC テスト回路の回路図

PIC プログラムの制作に用いたテスト回路を図 22 に、回路図を図 23 に示す。UART 通信で

BlueTooth モジュールと、I^2C 通信でディスプレイと、SPI 通信で SD カードと、USB 通信でオシロスコープと通信を行っている。二つの外付けクロックは片方はシステムクロックとして入力し PLL で引き上げて USB 通信に使用している。もう片方はリアルタイムクロックを動作させるためのものである [12]。PIC24FJ64GB002 は内蔵クロックを元に USB 通信を行うことができるため、本来は外部クロックも必要ない。しかし、USB ホストのデータ読み出し時に結構な頻度で通信が停止してオシロスコープが固まってしまう現象が起きたため、外付けクロックでより安定した USB 通信を行っている。この事により USB 通信の信頼性が非常に増大したが、最初内蔵クロックを使って回路を組んでいた時はなぜ USB 通信に失敗する事があるのか長い間わからず、対応に苦労した。この回路を使って様々なテストを行い、USB ホスト機能や FAT ファイル構築機能のプログラムを個別に動作させることはできた。しかし、両機能を同時に持たせようとすると PIC24FJ64GB002 ではプログラム容量をオーバーしてしまうことが発覚した。すなわち USB ホスト機能と FAT を用いた SD カードへのファイル保存機能を両方同時に PIC に書き込むことができなかった。しかしながら、この 2 つの機能はどちらも本機の製作にどうしても必要な機能である。そこで次節に示す M5Stack を用いて両機能を実現する道を選んだ。

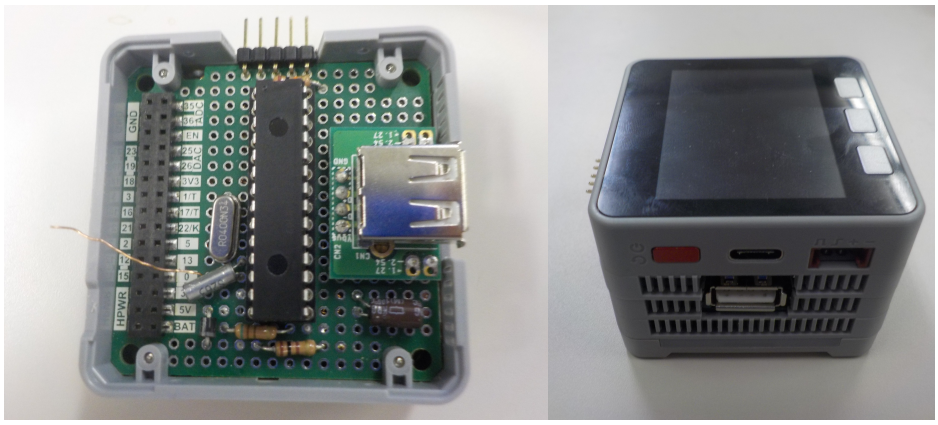
3.4 M5Stack の採用

図 24: M5StackGray の外観

M5StackGray の外観を図 24 に示す。M5Stack は ESP32 をコアにして作られたデバイスであり、ESP32 と同様 ArduinoIDE でプログラムを書き込み、動作させることができる [5] [6]。ESP32 の Wifi、BlueTooth 機能がそのまま使える他、ディスプレイ、ボタン、マイクロ SD カードスロット、バッテリーが備えつけられており、それらを扱うための高機能なライブラリも用意されている。また、バッテリーやセンサなどを増設する外部モジュールが存在し、見栄えを悪くしない形で機能を増設することができる。それらのモジュールの 1 つにプロトモジュールという自作回路を取り付けられるものが存在していたので、プロトモジュール上に今まで開発してきた PIC を中心にした回路を構築することにした。今まで UART で RN42 を使ってパソコンと BlueTooth 通信を行っていた部分を、RN42 の代わりに M5Stack 内蔵の BlueTooth ユニットを利用する。M5Stack が UART で PIC からデータを受け取ってから BlueTooth 通信でパソコンにデータを届ける形にしたのである。

これにより M5Stack にいくつかの機能を分担させられる他、ESP32 では C++ の機能を使えるために USB 版データ取得プログラムで使っていたソースコードの大部分がそのまま使用できるというメリットがあった。分担させた機能を以下に示す。

- PIC 側プログラムの機能
  - USB ホスト機能
  - オシロスコープへの命令予告
  - リアルタイムクロック
  - 状態変化割り込みによるスリープ機能
- M5Stack 側プログラムの機能
  - BlueTooth デバイス機能
  - SD カードへの記録
  - ディスプレイによる状態表示



(a) 増設回路　　(b) 増設後

図 25: 増設回路の外観

M5Stack に増設した回路を図 25 に、回路図を図 26 に示す。PIC の電源は M5Stack の電源ピン (3.3V) からではなくバッテリーピン (4.5V) を 3.3V に降圧して供給している。これは M5Stack の電源をオフにしている時でも PIC にはリアルタイムクロックを動作させ続けてもらう必要があるからである。一方で M5Stack の電源ピンは PIC の状態変化割り込みピンにつながっており、PIC は M5Stack の電源オフを検知してスリープ状態に入る事ができる。

3.5　電気二重層コンデンサによる RTC の動作

図 27 は M5Stack に増設していた元々の回路である。本来は M5Stack の動作中に図 28 の電気二重層コンデンサに充電を行い、電源オフ時は電気二重層コンデンサによって PIC を動作させる構想であった。実際に回路を組んだ結果、リアルタイムクロックのデータ保持は数日間持つ事が分かった。しかし、リアルタイムクロックを動作させるクロックが想定よりも短い時間で止まってしまったため、M5Stack のバッテリーから直接電源を取る形（図 26）へと変更した。また電気二重層コンデ

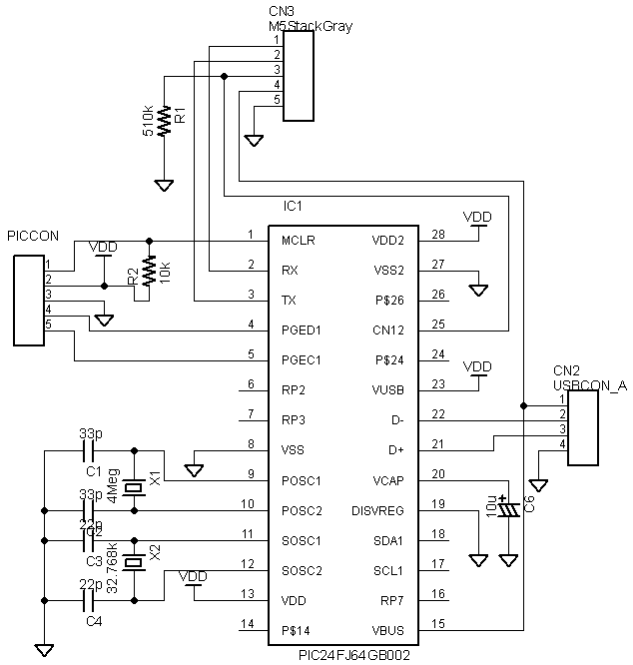


図 26: 増設回路の回路図

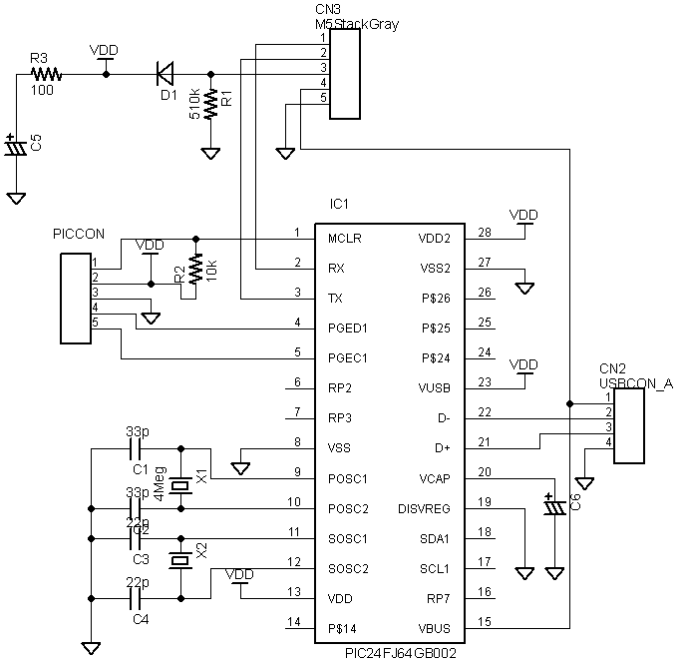


図 27: 元々の増設回路

ンサを取り外した際に、別要因から来る問題だと考えていた M5Stack の起動が不安定になってしまう問題も大きく改善された。

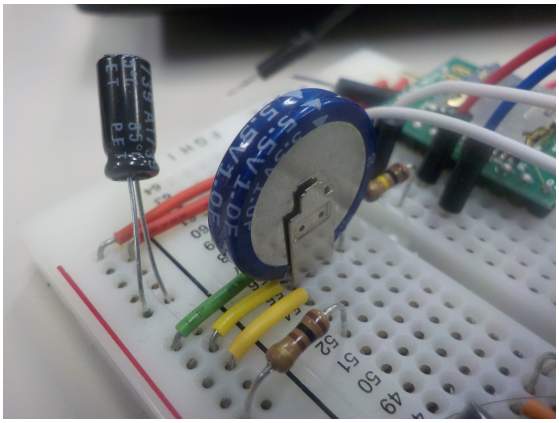


図 28: 電気二重層コンデンサ

3.6 BlueTooth 動作

M5Stack を採用するにあたって、まず最低限必要になるのは RN42 が行っていた動作を再現することである。BlueTooth モジュールの操作については M5Stack 用ライブラリの BluetoothSerial を使うと、プログラム的には UART とほぼ同じ形で BlueTooth 通信を利用できるようになっていた。UART1 はデフォルトで PC とのシリアル通信が行われておりデバッグにおいて有用なため使用しないことにした。

BluetoothSerial と Serial2(UART2) で SerialToSerial のプログラムを作り、RN42 の動作を再現した。この状態の M5Stack は内容を確認せず一文字ずつデータの横流しを行っているが、これは USB 通信と違って UART 通信にはサイズ情報が存在しないため、どこまでが一塊のデータか判別させにくいためである。一応送受信コマンドは全て末尾を 0x0a で統一してはいる。しかし、波形データはバイナリデータであり単純に特定の値を末尾扱いした場合、偶然波形データがその値を送ってきた時点で一塊のデータとして解釈されてしまう。この問題は一応、これから送るデータのサイズ情報を最初に伝えておく事や、特定データを 2 バイトデータに置き換える事などによって回避する事ができる。しかし、下手に状態遷移を持たせるとホストとの通信が中断された時や PIC が再起動した時のこと等、様々な状況をカバーしなければならない。従って、このモード中はあくまで単純な横流し処理のみを行い、M5Stack をパソコンと PIC とをつなぐ装置として使用することにした。

3.7 スタンドアローン動作

スタンドアローン動作では M5Stack 自身がホストとしてオシロスコープを操作するため、通信の流れを完全に把握することができる。従って、次に返ってくるデータの形式がバイナリなのかテキストなのかを決めつけて解釈することができる。前述した通り M5Stack のプログラムに USB 版 Windows アプリケーションの機能の一部を持たせることにした。グラフデータの取得や CSV 形式でのテキストデータ構築は全てクラスにまとめて整理していたため、Windows アプリケーションで使用した C++ ソースを丸ごと持ってくることができた。

データ取得時の画面を図 29 に示す。グラフの線描画は M5.Lcd.drawLine() 関数で、文字描画は

図 29: M5Stack によるデータ取得

M5.Lcd.println() 関数で行っている。時刻データは PIC のリアルタイムクロックのデータである。M5Stack からリアルタイムクロック関係の操作を行うため、PIC 側のプログラムには「:RTCGET」の命令を解釈する文を追加した。バッテリーの完全放電等によってリアルタイムクロックのデータが失われている場合は画面上に「XX」と表示する。

データ取得の動作を UART1 で監視した結果を図 30 に示す。「w」の後に続くメッセージが M5Stack から PIC へのメッセージであり、「r」の後に続くメッセージが PIC から M5Stack へのメッセージである。

無事にデータ取得が終わると、波形データを図 31 のように CSV 形式で SD カードに保存する。ファイルの保存方法に関しては、図 32 のように時刻情報を元に YY_MM_DD の形でディレクトリを作り、その中に連番でファイルを保存するようにしている。「XX」と表示されている場合はデータは XX_XX_XX フォルダに保存される。

モードはボタンによって切り替えることができるが、一応スタンドアローンモードでも何も処理がない時はパソコンからデータが来ていないかを確認しており、もしデータが来ていれば自動的に Bluetooth モードに切り替える処理をはさんでいる。

M5Stack は USB ケーブルから充電を行うため、実際にスタンドアローンで利用する場合は図 33 のように使う事を考えている。オシロスコープの前面（USB メモリを動作させるポート）から給電を行い、この装置を利用中にバッテリーバックアップを併用して完全放電時間を遅くする。もちろん背面にも USB ケーブルを差すことでデータの読み出しを行っている。図 33 ではオシロスコープに正弦波を入力して測定させており、製作した装置のスタンドアローン動作で、ボタンを押してデータを取得した時の様子を合わせて示している。オシロスコープのデータをファイル形式で必要とするときは、製作した本機を 2 つの USB 端子に差し込み、データ取得ボタンを押せば、即座に SD カードに保存できる。ファイル名を設定したり、USB への保存メニューを探さなくても良い部分は簡単になっているといえる。日付毎にフォルダ分けして保存されているので、好きなだけデータをとって、後日まとめて呼び出せばよい。新しい形の、そしてある程度使いやすい形のオシロスコープデータ取得装置の完成である。なお、ページの都合上、この装置の使い方は付録にまとめることにした。

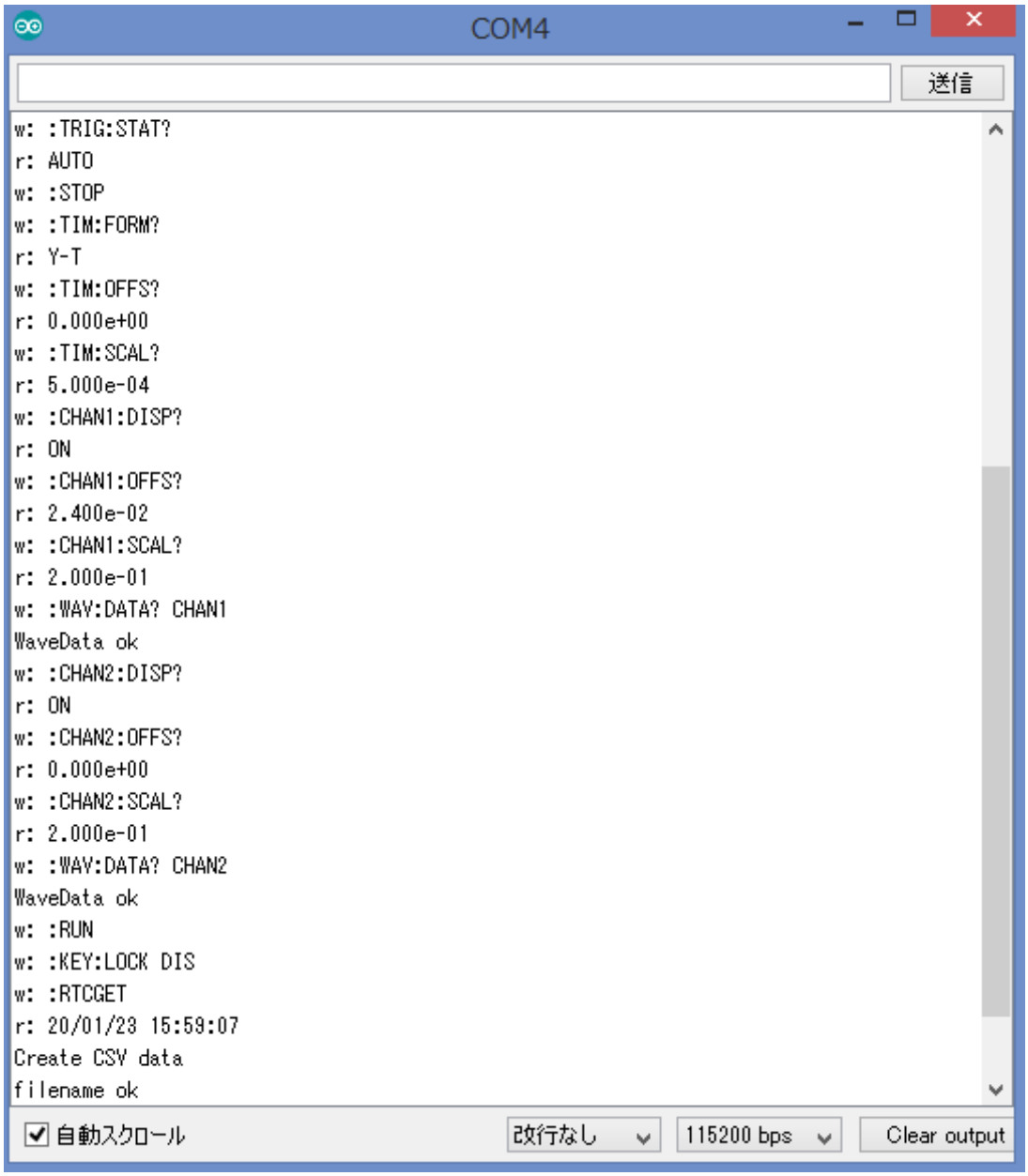
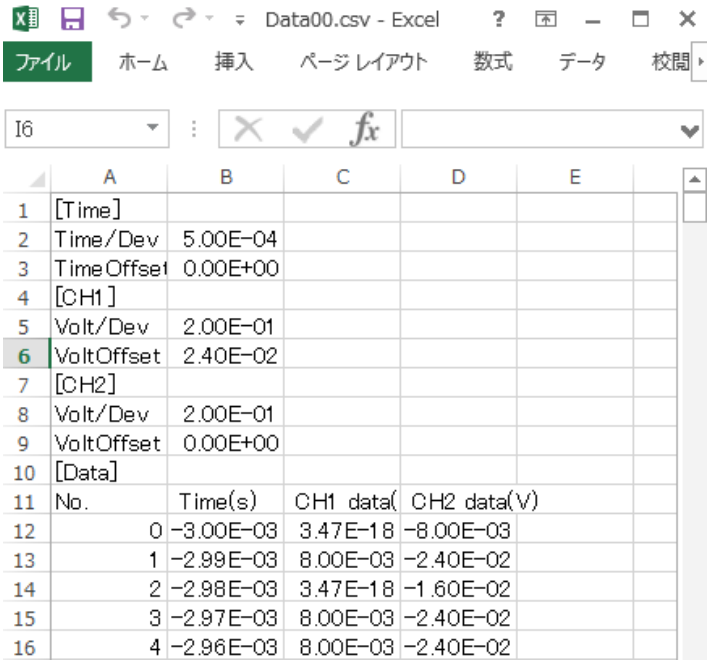


図 30: M5Stack と PIC 間の通信



	A	B	C	D	E
1	[Time]				
2	Time/Dev	5.00E-04			
3	TimeOffset	0.00E+00			
4	[CH1]				
5	Volt/Dev	2.00E-01			
6	VoltOffset	2.40E-02			
7	[CH2]				
8	Volt/Dev	2.00E-01			
9	VoltOffset	0.00E+00			
10	[Data]				
11	No.	Time(s)	CH1 data(	CH2 data(V)	
12	0	-3.00E-03	3.47E-18	-8.00E-03	
13	1	-2.99E-03	8.00E-03	-2.40E-02	
14	2	-2.98E-03	3.47E-18	-1.60E-02	
15	3	-2.97E-03	8.00E-03	-2.40E-02	
16	4	-2.96E-03	8.00E-03	-2.40E-02	

図 31: CSV 形式でのファイル構築

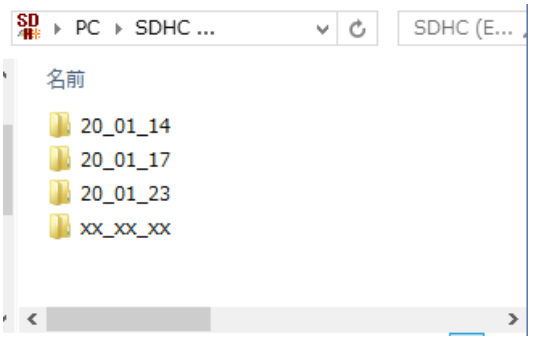


図 32: 日付データを利用したディレクトリ作成

図 33: オシロスコープからの USB 給電

第 4 章　BlueTooth 版オシロデータ取得プログラムの開発

4.1　USB 版との違い

USB 版オシロデータ取得プログラムから BlueTooth 版を作るにあたって、変更するべき点がいくつか存在する。まず接続処理において、USB 版では強制的に WinUSB ドライバをインストールさせておき、WinUSB クラスの GUID を SetupAPI 関数群を使って検索し、レジストリから通信先のデバイスパスを取得して CreateFile() 関数に使用した。一方で BlueTooth 通信に用いている BlueTooth ドングルは COM ポートを用意してくれるため、パス名の検索処理などは必要なく、BlueTooth デバイスとペアリングを行ったポート名で CreateFile() 関数を実行すれば通信用ファイルハンドルが得られる。USB 版よりも非常に簡単な接続処理ではあるが、逆に USB 版のように自動識別で接続しようと思うと単純にはいかない。というのも用意されている COM ポートのハードウェア ID はデバイス特有のものではないため、SetupAPI による検索処理を用いてもドングルの用意した複数の COM ポートの内、どれが目的のデバイスとペアリングしたポートなのか判断できないのである。

次に書き込みや読み取り処理に関してだが、既存ドライバを利用した USB 版では送信先のパイプ番号 (0x01) や受信先パイプ番号 (0x82) をアプリケーションが指定する必要がある。そのため WinUSB ドライバの専用関数を利用して送受信を行った。一方で BlueTooth 版では PIC が適切なパイプ番号を利用してホスト処理を行ってくれるため、アプリケーションは何も気にすることなく標準的な関数の WriteFile() 関数や ReadFile() 関数で通信を行っている。

4.2　COM ポートの検索

前述した通り、COM ポート用のファイルハンドル取得は簡単である。更に、利用している BlueTooth ドングルのドライバは BlueTooth 機器の接続状態に関わらず常に COM ポートを用意してくれているため、勝手に COM 番号がずれることもなく、同じパソコンを使い続けている限りはポート番号を決め打ちしていても問題は起こらない。とはいえ、その場で接続対象を選べる方がプログラムとして綺麗であることには違いないので、コンボボックスでポート名を選択できるようにした。この時、存在しないポートも選べてしまうのは見栄えが悪いため、TeraTerm の接続処理のように存在している COM ポートのみを表示するようにしたいと考えた。

有効なポート番号を取得する処理の王道は.NET フレームワークの GetPortName() 関数を使うことである。しかし、本研究は Win32 の GUI プログラミングを行いながらも VisualStudio は使わない方向性で進めている。この関数の再現として RegOpenKeyEx() 関数で”HARDWARE/DEVICEMAP/SERIALCOMM” を開き、RegQuaryValueEx() 関数でキー名”/Device/porteX”(X はポート番号) から値を読み出すとことで、レジストリから有効なポート名を取得してリスト化する関数を作った [15]。

GetPortName() 関数でも言えることだが、ポート名しか分からないとそのポートが何に使われているものかが一目で分からず不便であった。そこで、SetupAPI を用いてデバイスのフレンドリーネームを取得することにした。フレンドリーネームはデバイスマネージャーでデバイスを表示したと

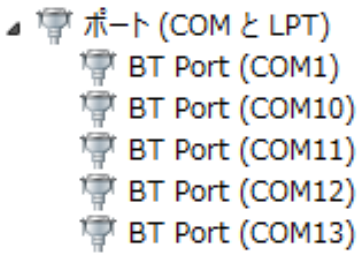


図 34: フレンドリーネーム（デバイスマネージャ）

きに表示される名前である（図 34 参照）。情報取得の手順は以下に示す通りであり、USB 版の自動接続に近い。

1. SetupDiGetClassDev() 関数の第一引数に CDC（ポートグラス）の GUID を指定し、該当するデバイスのテーブルを作成する。
2. SetupDiEnumDeviceInfo() 関数を使い、指定したインデックスのデバイス情報を DEVINFO_DATA 構造体として受け取る。
3. SetupDiGetDeviceRegistryProperty() 関数に SPDRP_FRIENDLYNAME を指定し、フレンドリネームのサイズを受け取る。
4. 適切なサイズのバッファを用意してもう一度 SPDRP_FRIENDLYNAME を指定し、フレンドリネームを受け取る。
5. フレンドリネームの (COMx) 部分を読んでポート名を割り出す。
6. インデックスを増やしながらテーブルの最後まで処理を繰り返す。

この手順を用いてポート情報を読み出した結果、図 35 のように有効なポート番号のみをグループボックス上に示すことができた。コンボボックスの処理は WM_COMMAND メッセージ内での WPARAM の上位バイトを確認して行う。ドロップダウンリストの表示タイミング（CBN_DROPDOWN）でポート名情報を取得して、ポート名情報をリスト化しコンボボックス上に表示する。リストが選択されたとき（CBN_SELENDOK）は、ComboBox_GetCurSel() 関数を使って選択されたインデックス番号を取得して、ポート名情報のリストから対象のポート名を取り出して接続の前準備を行う。

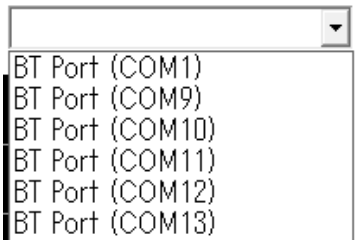


図 35: フレンドリーネーム（自作ソフト）

4.3　時間設定

パソコン側から PIC に対して時間設定を行う操作を作った。PIC には新しく「:RTCSET」を解釈してリアルタイムクロックを設定するプログラムを追加し、M5Stack はこれまで通り BlueTooth モードで横流しを行ってもらう。時間取得は time.h の time() 関数と localtime() 関数で行い、テキストフォーマットは strftime() 関数で":RTCSET %y/%m/%d/%H/%M/%S" を指定して整形している。

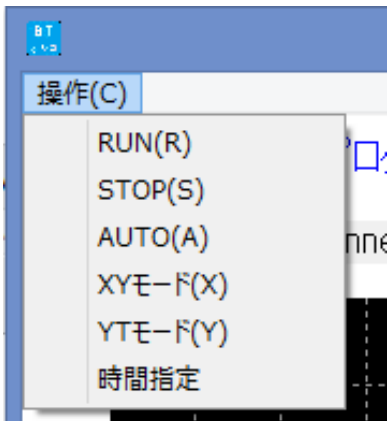


図 36: 時間設定ボタン

最初は「:RTCGET」を送ってリアルタイムクロックの値を読み出し、もしも時刻がずれていれば自動的に設定するような処理にしていた。しかし、パソコンの時刻自体がずれている可能性もあるため、図 36 のようにメニューに「時刻設定」ボタンを追加して手動で行うことにした。

第 5 章　考察

この二年間の研究活動において、技術的な学びは非常に多いものだったと考えている。この論文に載せた活動内容はある程度形になったものばかりであるが、制作物として残らなかった学習活動も含め、様々な寄り道を経て身の回りのモノの理解を深めることができた。Linux ドライバの構造や作り方、字句解析、DLL の作り方といった知識は勿論、実際に味わって実体験として学べたことも多かったように思う。本研究の様に形になり完成したものだけが論文の形になる場合、考察という形態では書きにくい。そもそも何か問題があれば修正をされているか、ある程度不便でありながらも完成形態として組み上げられている。こうすれば良かったと思うことはいろいろあるが、実際にそのようにして良くなったとしても、それが最終形態になるだけである。そこで、考察というよりも、今の様な形になるまでの変遷を思いつく程度に書いていこうと思う。

今回のオシロスコープデータ取得プログラムの開発においては、バージョン違いのほぼ同一なアプリケーション制作を初めて経験した。更に PIC、M5Stack、Windows というバラバラな環境で似通った目的のプログラムを制作してきた。その過程で普段よりも特に感じたのが機能分割、そしてオブジェクト指向の重要性であった。完全に自分が仕様を把握できる個人のプログラミングにおいては、そこまでオブジェクト指向の強みが活かされることはない。勿論複数の似たオブジェクトを生成する必要があれば話は別だが、シングルトンのオブジェクトばかりならばスコープ制限が邪魔になる事の方が多い。そのため最初に USB 版のプログラムを作る際は、どうせ自分しか読まないソースだからと、一つのソースファイルに多数の機能を持たせ、ほとんどの情報をグローバル変数で保存していた。実際、グローバル変数によってどこでも情報が共有でき、開発を高速化できた面はあった。そして、BlueTooth 版をつくる際には接続処理を変えるだけだと思い、安易な気持ちでソースをコピーして部分的に内容を書き換えた。その後は必要に合わせてポート名選択用コンボボックス等の BlueTooth 版独自の処理を書き加えていった。問題になったのはカーソル機能や計算軸機能を両方のバージョンに追加しようとした時である。機能を実現するためにグラフ用関数の仕様を変更し、グローバル変数名なども変えていく内に、もう一方のバージョンではどこに処理を追加するべきなのか、この変更は既に適用したのかなどを考え込むことが多くなっていった。結果、BlueTooth 版ではカーソル機能と計算軸機能の追加を断念することになった。一方で、Windows 版のプログラムを元に M5Stack 版のプログラムを開発する際には、Windows 版の処理を整理しており、オシロスコープのデータ取得とファイル出力を一つのクラスにまとめてあった。そのため、どの範囲が一塊の機能なのかが分かりやすく少しの作業で開発することができた。バグを修正した際も USB 版のプログラムからクラスを丸ごとコピーするだけで済ませる事ができた。この件からは、機能分割の有用性とそれを実現する上でオブジェクト指向が有効な手段であることを実体験をもって学ぶことができたと思う。このようなバージョン違いのプログラム制作は、趣味のゲームプログラミング、特に一人での開発作業ではあまり味わう機会のない体験だったと感じている。

また中間発表の質疑応答において、実際の利用という観点から見て、製作予定の無線化デバイスに何の意味があるのかを答える事ができなかった。しかし、今このデバイスを作る意義を一つ見出すことができている。無線通信を行うために本機を作り、USB ホストとしての機能を持たせて、通信状態が悪いときの備えとしてスタンドアローン動作を追加した結果、本機はパソコンが無くてもボタン

を一回押すだけで波形データを保存できるデバイスとなった。パソコンと通信しなくて良いことは十分なメリットだと考えている。考察として振り返ってみると、クラス化等は今後のプログラム制作に役立つプロセスの 1 つであると思う。

第 6 章　結論

本研究の結論として、USB 端子とパソコンとを USB ケーブルで結び、パソコンからオシロスコープを制御したり、取得したデータを USB 経由でパソコンに転送し、表示するとともに CSV 形式で保存できるようなソフトを開発することができた。また、自作 USB ホストを制作してオシロスコープを操作し、無線によって通信を行う機器の開発も行うことができた。プログラムや機器の開発は、実際にそれを使う人がいて、使われる事によって価値を与えられる。USB ケーブルによる読み出しソフトに関しては、新バージョンのソフトにおいては試用試験を行っていないが、バージョンアップ前のソフトは実際に利用されており、来年以降も使われるだろうという話である。この点において意味のあるものを構築したと考えることはそれ程間違っていないと思う。オシロスコープの測定データを SD カードにボタン 1 つで保存することができる機器の開発は量産を考えておらず、1 個しかない状態でおわるであろう。この点においては後々使われていくとは思えず、また 2 個目、3 個目の同じ機器が他者によってできるとも思えない。しかしながら、これ自体に利便性があることは明らかであり、考え方自体も含めて、開発に大きな意味があったと思う。以上を通して本研究の目的は達成できた。

謝辞

今回の修士論文の作成にあたり、終始に丁寧で熱心なご指導とご教示を賜りました高知工科大学大学院基盤工学専攻電子・光システム工学コース 綿森 道夫准教授に心からの感謝を申し上げます。本研究では綿森道夫准教授のお力添えなしでは完成に至ることはなかったと思います。副指導教員の山本 真行教授、星野 孝総准教授にも感謝を申し上げます。

また、高知工科大学大学院基盤工学専攻電子・光システム工学コース在学中 本研究実験遂行や学生生活、その他様々な面で終始ご厚意頂きました、山本 利水教育講師や皆様には重ねて感謝の意を述べさせて頂きます。

参考文献

[1] ジャン・アクセルソン ,”USB コンプリート第三版” インサイトインターナショナル株式会社訳 (エスアイビー・アクセス 2006 年)

[2] 後閑哲也,” 改定新版 PIC で楽しむ USB 機器 自作のすすめ” (技術評論社 2011 年)

[3] 後閑哲也,”C 言語ではじめる PIC24F 活用ガイドブック” (技術評論社 2007 年)

[4] 後閑哲也,”PIC で楽しむ Android アクセサリの自作” (技術評論社 2012 年)

[5] 大澤文孝,”「M5Stack」ではじめる電子工作” (株式会社工学社 2019 年)

[6] 下島健彦,” みんなの M5Stack 入門” (株式会社リックテレコム 2019 年)

[7] インターフェース編集部,”Interface 増刊 改定新版 USB ハード＆ソフト開発のすべて” (CQ 出版株式会社 2005 年)

[8] 土井滋賀, 那須靖弘, 上田悦子,”Win32API 完璧マスタ” (CQ 出版株式会社 2005 年)

[9] 赤坂玲音,”Windows ゲームプログラミング” (ソフトバンク クリエイティブ株式会社 2006 年)

[10] 鈴木哲哉,”PIC と C 言語の電子工作” (株式会社ラトルズ 2009 年)

[11] 鈴木哲哉,”PIC で動かす USB MCHPFSUSB framework ＋付属プリント基板で即日完成” (株式会社ラトルズ 2010 年)

[12] 鈴木哲哉,” プリント基板で作る PIC 応用装置” (株式会社ラトルズ 2009 年)

[13] 安部恵一,”USB2.0 インターフェース設計術” (株式会社電波新聞社 2008 年)

[14] 粂井康孝,” 猫でもわかる Windows プログラミング第 4 版” (SB クリエイティブ株式会社 2014 年)

[15] 石川竜也,” プログラミングのスパイス Windows プログラムの隠し味” (ソフトバンク パブリッシング株式会社 2005 年)

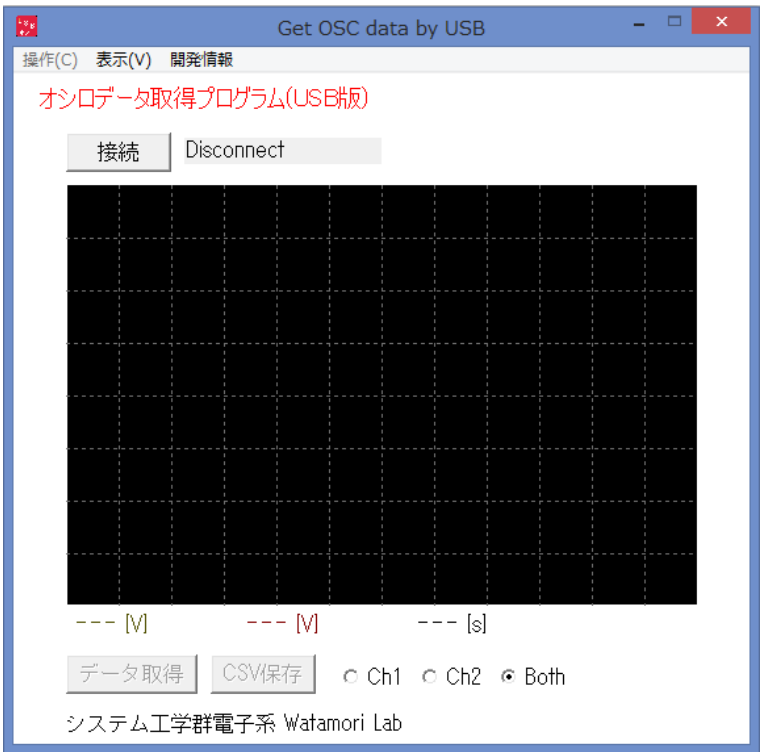
[16] トランジスタ技術編集部,”Bluetooth 無線でワイヤレス I/O” (CQ 出版株式会社 20015 年)

付録

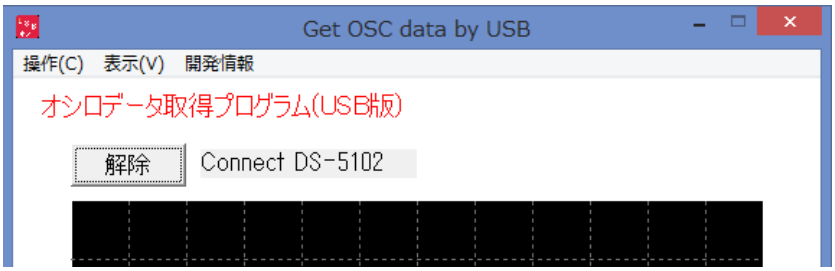
付録 A　USB 版オシロスデータ取得プログラムの使い方

- データ取得
  1. オシロスコープとパソコンとを USB ケーブルでつなぐ。

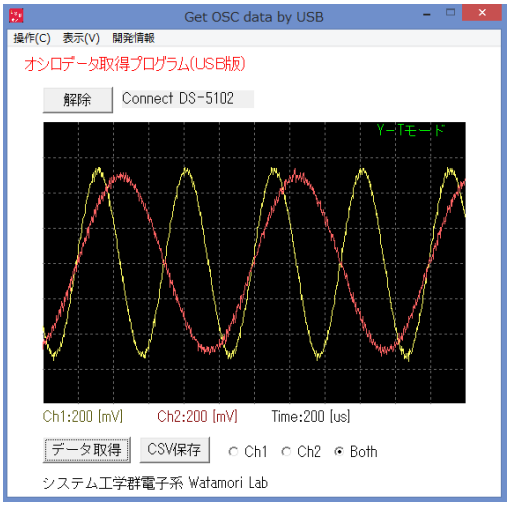
  2.「オシロデータ USB」をダブルクリックしてソフトを起動する。



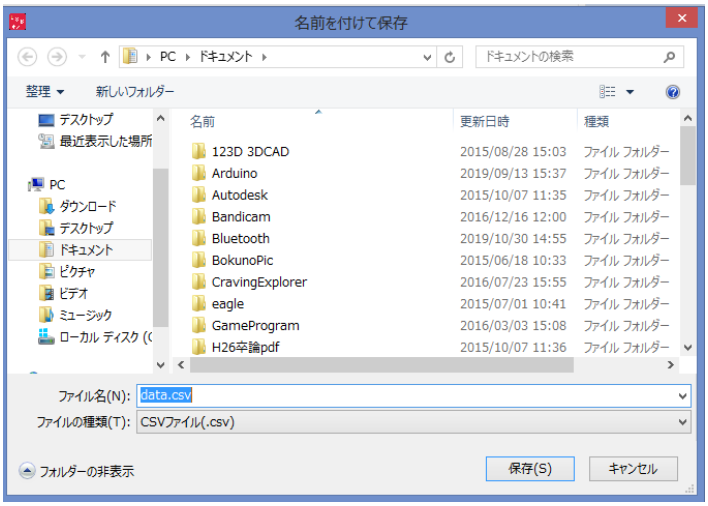
3.「接続」ボタンを押す。



4.「データ取得」ボタンを押す。



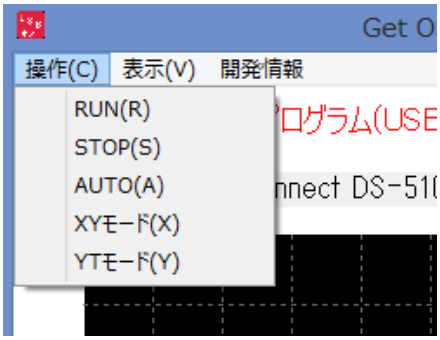
5.「CSV 保存」ボタンを押し、保存場所を決めてファイルを保存する。



- 操作機能

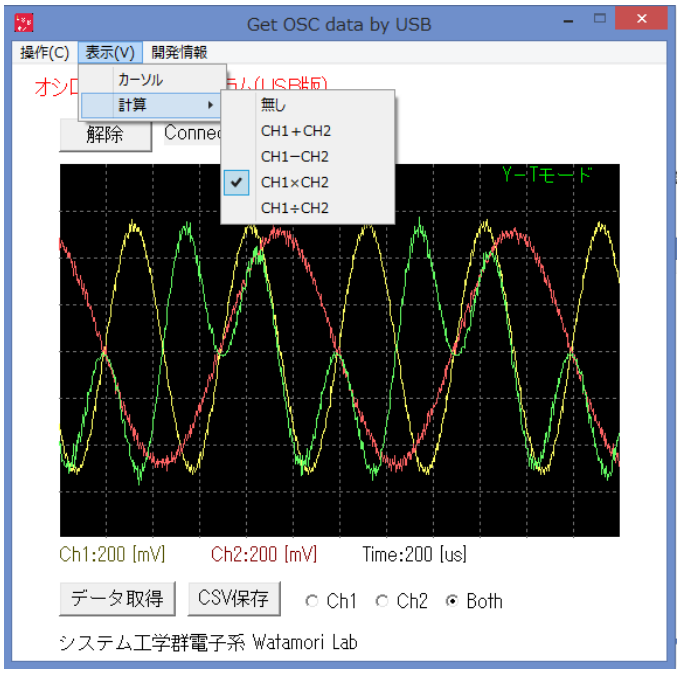
  接続中はメニューから以下の操作を行える。

  – RUN　　：測定を行う
  – STOP　：測定を止める
  – AUTO　：自動調整
  – XY モード：リサージュ表示へ変更
  – YT モード：時間-電圧表示へ変更



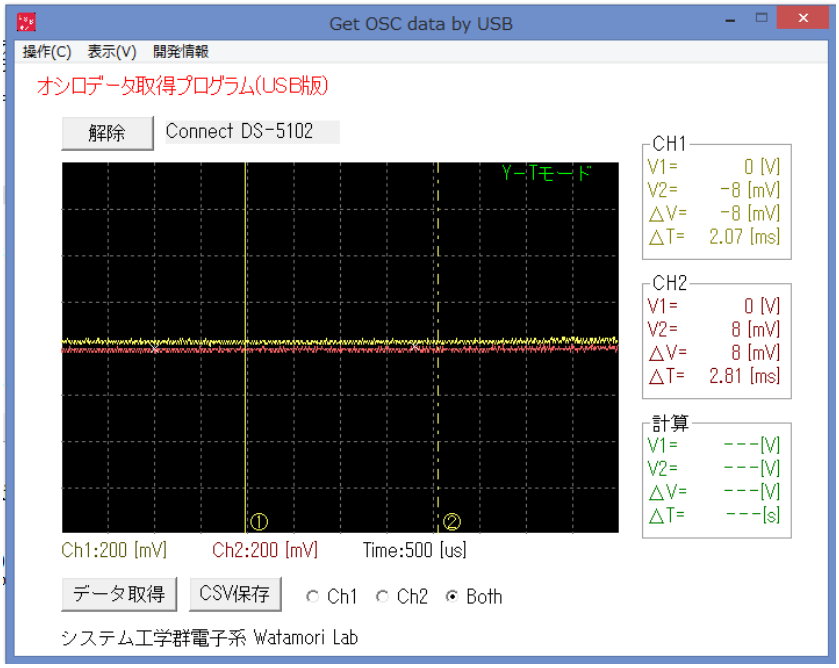
- 計算軸機能

  メニュー中の計算軸を選択するとチャンネル１、２を元に四則演算した値のグラフが表示される。



- カーソル機能

  メニュー中のカーソルを選択するとカーソル表示モードになる。グラフエリアをクリックする事で近くのカーソルを移動させられる。カーソルはチャンネル毎に用意されており、カーソル情報の表示範囲をクリックすると操作チャンネルを変更できる。



- 開発情報の表示

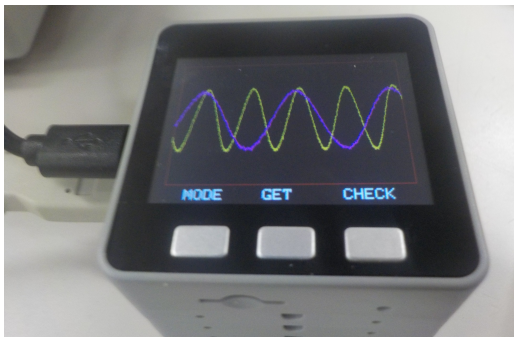
  メニュー中の開発情報を選択すると、開発に使用した API を記したダイアログが立ち上がる。

付録 B　BlueTooth 版データ取得デバイス関連の操作説明

- データ取得
  1. 充電用 USB 端子をオシロスコープ前面とつなぐ。

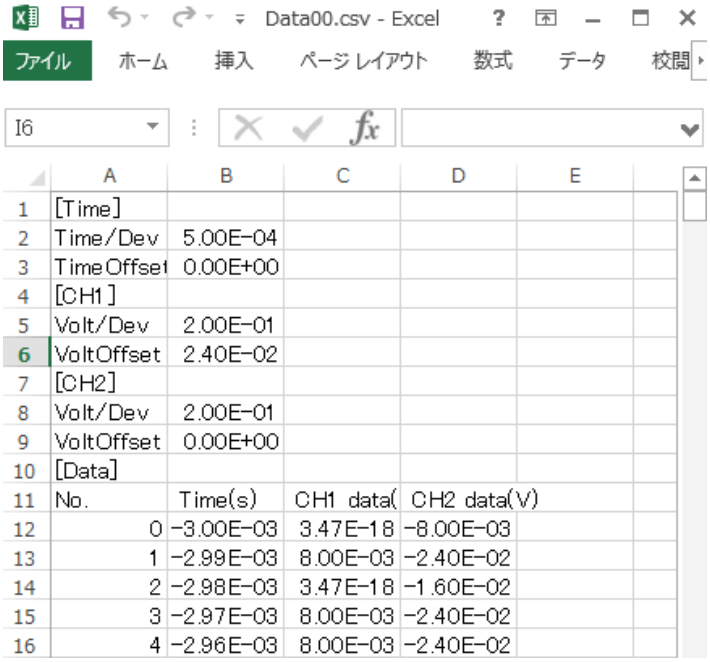
  2. データ用 USB 端子をオシロスコープ背面とつなぐ。

  3. GET ボタンを押すと SD カードにデータが記録される。



4. SD カードの内容は以下のようになる。

5. データファイルを開くとこのようになっている。



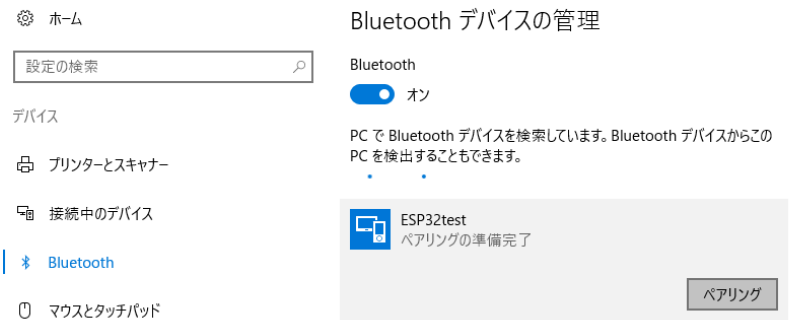
	A	B	C	D	E
1	[Time]				
2	Time/Dev	5.00E-04			
3	TimeOffset	0.00E+00			
4	[CH1]				
5	Volt/Dev	2.00E-01			
6	VoltOffset	2.40E-02			
7	[CH2]				
8	Volt/Dev	2.00E-01			
9	VoltOffset	0.00E+00			
10	[Data]				
11	No.	Time(s)	CH1 data(	CH2 data(V)	
12	0	-3.00E-03	3.47E-18	-8.00E-03	
13	1	-2.99E-03	8.00E-03	-2.40E-02	
14	2	-2.98E-03	3.47E-18	-1.60E-02	
15	3	-2.97E-03	8.00E-03	-2.40E-02	
16	4	-2.96E-03	8.00E-03	-2.40E-02	

- 他ボタンの効果
  - MODE ：BlueTooth モードとスタンドアローンモードの切り替え
  - CHECK：オシロスコープの接続確認
    (接続中なら Y, 非接続なら N が表示される)

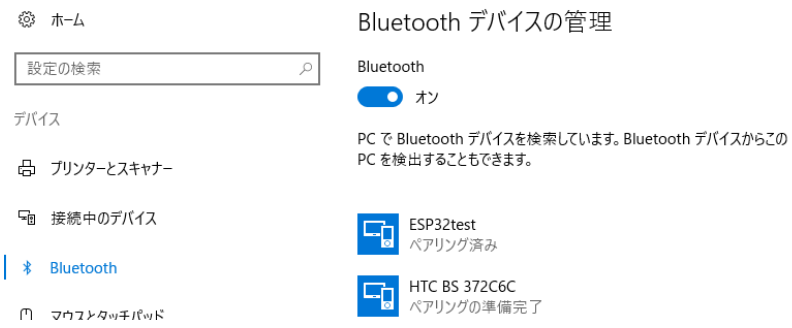
- ペアリング
  1. タスクバーから BlueTooth アイコンを右クリックする。

  2.「デバイスの追加」を選ぶ。
  3.「ESP32test」が認識されているかを確認する。

  4. 選択してペアリングボタンを押す。



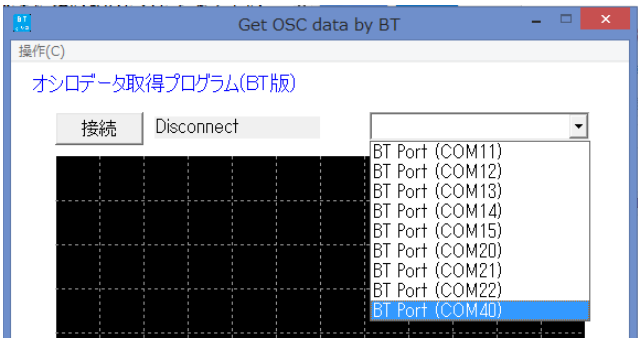
  5.「ペアリング済み」と表示されれば成功。



- 時刻設定

1. BT 版アプリケーションを起動する。

2. ペアリングしたポート番号を選択して接続する。



3. 時刻設定を押す。

4. モードを切り替えると反映された時間が確認できる。

付録 C　今回利用した Latex2 のプリアンブル部

```
\documentclass[a4paper,11pt,dvipdfmx]{jsarticle}
\usepackage[hiresbb]{graphicx}
\usepackage{subcaption}
\usepackage{fancyhdr}
\usepackage{cite}

%% 用紙左上を原点とする
\setlength{\voffset}{-1in}
\setlength{\hoffset}{-1in}

%% 左マージン 25mm
\setlength{\oddsidemargin}{27.7mm}
\setlength{\evensidemargin}{17.7mm}

%% 上マージン 25mm
\setlength{\headheight}{0pt}
\setlength{\headsep}{0pt}
\setlength{\topsep}{0pt}
\setlength{\topmargin}{22.7mm}

%% 傍注を削除
\setlength{\marginparsep}{0pt}
\setlength{\marginparwidth}{0pt}
\setlength{\marginparpush}{0pt}

%% 脚注領域を削除
\setlength{\footnotesep}{20pt}
\setlength{\footskip}{28pt}

%% テキスト領域
\setlength{\textwidth}{\paperwidth}
\addtolength{\textwidth}{-\oddsidemargin}
\addtolength{\textwidth}{-\evensidemargin}
\setlength{\textheight}{\paperheight}
\addtolength{\textheight}{-2\topmargin}
\addtolength{\textheight}{-20pt}
```

```
\cfoot{--\ \thepage \ --}
\renewcommand{\thesection}{第 \arabic{section} 章}
\renewcommand{\headrulewidth}{0pt}
\fancyhead{}

\begin{document}
\thispagestyle{empty}
% 左上{\hfill}右上
\vspace*{15mm}
\begin{center}
\Huge{\textbf{修士論文}} \\
\vspace{13mm}
\LARGE{USB システムの構築と応用に関する研究} \\
\vspace{0.5\baselineskip}
\LARGE{The investingation of construction and application of USB systems} \\
\hrule
\vspace{29mm}
\Large{\textbf{報　告　者}} \\
\vspace{\baselineskip}
\underline{\parbox[b]{6.4cm}{学籍番号 : 1225069\\ 氏名 : 谷脇宇京 }} \\
\vspace{15mm}
\Large{\textbf{指　導　教　員}} \\
\vspace{0.5\baselineskip}
\underline{\makebox[6.4cm][c]{\Large{綿森　道夫　准教授}}} \\
\vspace{21mm}
\Large{\textbf{令和 2 年 2 月 10 日}} \\
\vspace{19mm}
\Large{高知工科大学　システム工学群　電子工学コース}
\end{center}
%{\vfill}
% 左下{\hfill}右下
```