知ってる？インターネット・ガジェット創作
USB、I2C、UART、Xbeeプロトコールスタック活用術

慶應義塾大学環境情報学部教授　武藤佳恭

目次

はじめに

２１世紀において、情報通信技術(ICT)の中で重要な技術の一つがインターネットです。今や、インターネットなしの生活が考えられないと思われるほど、インターネットは社会に普及し根付いています。入門書である本書の目的は、読者が自由自在にインターネット・ガジェットを設計・製作することです。“インターネット・ガジェット”とはインターネットにつなげることができる電子小道具(ガジェット)のことです。

最新オープンソースソフトウエアと安価なハードウエア技術を使うことによって、比較的簡単に、インターネットに接続できるガジェットを設計し自作することが出来るようになりました。本書では、インターネット・ガジェット（ハードウエア＋ソフトウエア）の作り方を詳細に説明します。ネットブックや携帯端末(iPod Touch/iPhone)に自作のインターネット・ガジェットを接続することによって、さまざまな楽しいことが簡単に実現できるようになります。

秋葉原で売られている部品を使って自作したガジェットをインターネットにつなげるには、色々な方法があります。例えば、無線LAN、イーサーネット（10BaseT、100BaseT、1000BaseTなどのケーブル)、パソコンや携帯端末のUSB、赤外線通信、RS232Cなどを経由する方法などです。本書では、USB、UART(RS232C)、Xbee(無線)などのインターフェイスを使って自作ガジェットをパソコンや携帯端末に接続します。ガジェットに接続されたアナログ・ディジタルセンサーの値を読んだり、アクチュエータ（モータ、LED、スピーカー、電化製品の制御および電源ON/OFF）を駆動したり、インターネット経由で別のパソコンや携帯端末からガジェットを操作できます。

本書で説明する内容は、将来ICT分野で活躍したい人が知っておくべき必要不可欠な技術であり、設計のための便利なICT道具の紹介と使い方も含まれます。

工業専門学校生、大学生、大学院生の教科書として、または、ラジオ少年やICT技術者の参考書として本書をお使いください。

USB、I2C、UARTなどのインターフェイスを利用するためには、通常は、専用のハー

ドウエアインターフェイスが必要でした。しかしながら、最近のオープンソースソフトウエアの発展によって、ソフトウエアインターフェイスからのアクセスが可能になりました。簡単に言うと、専用のハードウエアチップを使うことなく、ソフトウエアでハードウエア機能を実現することができるようになったということです。

本書では、ソフトウエアUSB、ソフトウエアI2C、ソフトウエアUART (RS232C)などのソフトウエアプロトコールの活用方法を説明します。特定のハードウエアチップを使う必要がないので、数百円の安価なインターネット・ガジェットが製作できます。

初心者の皆さんは、“自作のインターネット・ガジェットを作って、自由自在に使いこなそう！”と思われるでしょうが、そう簡単ではありません。実は、初心者にとって大きな壁が三つ存在します。その一つの壁は通信プロトコールと呼ばれ、複雑な通信手順を理解しなくてはいけない壁です。通信手順の中身が複雑過ぎて理解できないので、多くの初心者はそこで断念してしまいます。しかも、それらの複雑で難しいところの解説が英語かプログラム言語で書かれています。つまり、初心者にとって二つ目の英語の壁、三つ目のプログラミングの壁に直面します。

本書では、これらの三つの壁の克服とインターネット・ガジェットの設計・製作は別物であると考え、説明していきます。もちろん、理解できないことはそのままにして置かないで、何回も繰り返し理解しようとする努力が必要ですが…

つまり、複雑な通信プロトコールを理解しなくても、初心者がインターネット・ガジェットを設計・製作し、使いこなせるための簡単な方法を紹介します。通信プロトコールがいくら複雑であっても、それらのすべての通信手順はブラックボックスと考えてよいわけです。ただし、ブラックスボックスとの簡単なデータのやり取りだけを理解し、そのブラックボックスを使いこなす方法を習得すればよいわけです。つまり初心者にとってインターネット・ガジェットを設計・自作するには、ブラックボックスとの簡単なやり取りの仕方・手順（プログラミング）を理解すればよいわけです。

初心者にありがちな“プログラム全体を隅々まで理解”するのではなく、必要不可欠なプログラムのキーポイントのみを理解すればよいわけです。プログラムでのキ

ーポイント部分が理解できてくれば、次第に周りのわからないプログラム箇所も解かってきます。すなわち、三つの壁（通信プロトコールの理解、英語、プログラミング）を完全に克服しなくても、通信プロトコール（ブラックボックス）との簡単なデータのやり取りを理解さえすれば、本書の目的であるインターネット・ガジェット設計・製作することが出来、しかも、そのインターネット・ガジェットを使いこなせるようになります。

人間の面白い能力として、何度も何度も、“理解できない式”・“文書”・“物”・“現象”を観ていると、次第にその本質がなんとなく分かってくるようです。プログラム中で分からないところはおまじないと思って、思い切ってスキップしてもらっても大丈夫です。最初は理解できなくても、何回も理解しようと試みると、自然に（自ら然り：self-organization)、次第に解かってくるようです。中身がよく理解できないと居心地がわるい人がいるかもしれませんが、“そのうち解るさ”と気楽に思ってください。重要なことは、理解しようとする繰り返しの行動が、解らないものへの理解の手助けになります。

自然現象（重力、電磁波、音波、水、…、我々自身の存在）そのものは、最新科学を使ってもまったく説明できていません。例えば、何故、重力があるのか。自然現象の振る舞いは数式で表現できても、根本の自然現象そのものに関して我々は全く解っていません。例えば、正確な地震予知ができないのは、自然現象そのものへの理解不足です。現代社会では、科学者が自然現象を発見・発表し、その発見された自然現象を利用して技術者が製品化・商品化し、ビジネスマンがそれらを我々に売りさばいているのです。

本書で紹介する通信プロトコールは、1.5Mbps転送速度の低速USBです。USBにはホストとデバイスの２種類があります。本書で紹介するUSBガジェットはデバイスの方です。USBホストは王様で、USBデバイスは召使という関係です。つまり、USBホストがUSBインターフェイスを支配し、すべてのUSBデバイスに命令します。USBデバイスは、USBホストからの命令を粛々と実行するだけです。USBデバイスは、USBホストからの命令を識別するために固有のIDを持っています。USBデバイスは、自分がホストに呼ばれているのかどうか判断し、自分が呼ばれている場合は、その命令を実行します。

USBインターフェイスは今や大人気で、パソコン、携帯端末、ゲーム機器、ネットブックなど多くのネットワーク商品で使われてきています。つまり、読者の皆さんがこれから自作するインターネット・ガジェットは、最小限のネットワーク機能を備えた小さなコンピュータなのです。

本書で紹介するセンサーにはアナログセンサーとディジタルセンサーがあります。アナログセンサーには、電圧変化するセンサー、電流変化するセンサー、抵抗値変化するセンサー、静電容量変化するセンサーなどがあります。具体的には、照度センサー、温度センサー、湿度センサーなど、アナログセンサーを使いこなすための簡単な方法を紹介します。

本書で紹介しているI2Cは、ディジタルセンサーチップとのやり取りをするための通信プロトコールです。本書では、最新の気圧センサー(BMP085)を紹介します。また、Parallax社のBasic Stamp用のディジタルセンサーチップとして、磁気コンパスセンサー(HM55B)例も解説します。

また、今流行の携帯端末iPhoneやiPod Touchには、UART(RS232C)が備わっているので、そのUARTインターフェイスを経由してセンサー値を読み取ることが出来ます。iPhoneやiPod Touchはインターネットに接続可能なので、リモートでそのセンサー値を読み取ったり様々なアクチュエータを制御したりできます。

本書で紹介する最新の無線モジュールXbeeは、多機能型無線UARTと考えることが出来ます。通常のUARTと違うのはXbee内蔵のAES暗号を利用して暗号通信できることです。従来、無線モジュールを使うのは初心者には難しくハードルも高かったのですが、Xbeeチップを使うことでインターネット・ガジェットに無線機能を簡単に加えることが出来ます。

本書では、専門知識がなくても、できるだけ内容が理解できるように試みています。本書で一番重要な回路知識は、オームの法則です。オームの法則は、次式で表現されるような実験式で、電圧は電流と抵抗に比例するという物理現象です。

電圧（Ｖ）＝電流（Ｉ）＊抵抗（Ｒ）

例えば、抵抗Ｒ（１ＫΩ）に電池（電圧１．5V）を接続すると、流れる電流Ｉは

電流（Ｉ）＝1.5V／１ＫΩ＝1.5／10^3＝1.5ｍＡ　になります。

この抵抗での消費電力Wはどうなるでしょうか。消費電力Wは、電圧と電流の積なので、

W＝電圧（Ｖ）＊電流（Ｉ）＝1.5V＊1.5ｍＡ＝2.25ｍWになります。

先ほどの抵抗値が１ＫΩでなく５Ωの時は、抵抗の消費電力Wはどうなるのでしょうか。

電流Ｉは、
Ｉ＝1.5／5＝0.3（Ａ）
したがって、抵抗の電力Wは、
W＝1.5＊0.3＝0.45（W）

抵抗５Ωを選ぶ時に、定格電力を考慮する必要があります。定格電力とは、抵抗が耐えられる最大の消費電力のことです。定格電力を超えると、抵抗が燃えて大事故につながる可能性があります。また、定格電力を超える電力が抵抗に与えられると、想定を超えた熱を発生し、その抵抗の回りの電子部品に悪影響を与えます。実際には定格電力は消費電力の2倍くらいの余裕が必要です。抵抗の定格電力は、１／８W、１／４W、１／２W、１W、その他があります。この場合は１Wを選択したほうが良いでしょう。１W以上は、セメント抵抗やホーロー抵抗になります。最近人気のハイブリッド車では、大容量の抵抗が使われていますが、調べてみてください。

電子部品を選ぶ時に気をつけることは、定格電圧を超えないこと、定格電流を超えないこと、定格電力を超えないことの3つが重要です。

もう一つ回路設計で大事なことは、時間と周波数の関係を理解することです。

周波数（ｆ）＝１／時間（ｔ）または　時間（ｔ）＝１／周波数（ｆ）

例えば、10MHｚのクロック周波数とは１秒間に10^{7}回の（０と１の）振動をしています。M（メガ）とは10^{6}のことです。つまり、一つの周期の幅は、10^{-7}秒になります。10^{-7}秒とは、100ｎ秒（ナノ秒）のことです。ｎとは、10^{-9}です。復習として名称と単位の関係表を表１に示します。

表１　名称と単位

名称		単位
E	エクサ	10^{18}
P	ペタ	10^{15}
T	テラ	10^{12}
G	ギガ	10^{9}
M	メガ	10^{6}
K	キロ	10^{3}
m	ミリ	10^{-3}
μ	マイクロ	10^{-6}
n	ナノ	10^{-9}
p	ピコ	10^{-12}
f	フェムト	10^{-15}
a	アト	10^{-18}

本書で登場する電子部品（抵抗、キャパシタ*、ダイオード、トランジスタ、LED、LCD、各種のセンサー、AD変換器、フラッシュメモリ付きマイクロコントローラチップなど）の使い方の基礎から応用までを簡単に解説していきます。

本書は、理論よりも、実学に重点を置いて書いています。実際にそれらの部品を使いながら、理解を深めることが重要です。

*日本では、キャパシタのことをコンデンサと呼んでいますが、世界に通用しません。必ずキャパシタと呼びましょう。

本書では、電子部品の双方向性の現象を紹介します。双方向性の現象とは、例えば、

スピーカーの場合、スピーカーに電気信号を与えると音や振動が発生します。逆に、スピーカーに音や振動を与えてやると、電気信号が発生します。この現象を使って実用化されているのが、ＪＲ東日本・慶應義塾大学武藤佳恭研究室との共同研究の発電床です。乗客が床を歩く振動エネルギーを電気エネルギーに変換しています。発電床の上を歩くことによって、ピエゾ素子に振動を与え、 交流電気エネルギーを発生します。その交流電気エネルギーを直流に変換し、直流に変換された電気エネルギーをキャパシタに蓄積します。同様に、LEDにも双方向性現象があり、LEDに電流を流すと光を発生しますが、LEDに光を与えると電気信号が発生します。つまり、LEDは、発光デバイスとしても使えますがセンサーとしても利用できます。本書では、アナログ入力設定のところで、簡単なLED双方向性回路を紹介します。

インターネット・ガジェットの中心部品は、CPUであるマイクロコントローラです。ATMEL社のマイクロコントローラチップは、AVRとよばれ、世界で幅広く使われています。実は、AVRチップが世界で広く使われる理由があります。LinuxやFreeBSDなどのオープンソースソフトウエアの多くは、GNUツールを使って開発されています。8ビットCPUでは、AVRのチップがGNUツールによってサポートされています。つまり、AVRチップ用のGNU Cコンパイラ（avr-gcc）が無償でダウンロードできるわけです。また、様々なオープンソースのソースプログラムが利用可能ということになります。本書では、オープンソースのUSBプロトコールソフトウエア、I2Cプロトコールソフトウエア、ソフトウエアUARTの使い方を解説します。

開発の手順として、ハードウエア設計とソフトウエア設計(Ｃ言語)は、同時進行します。ハードウエア設計しながら、プログラムを開発していきます。開発したプログラムは、Ｃコンパイラによって機械語に翻訳されます。具体的には、Ｃコンパイラ（avr-gcc）は、Ｃ言語で書かれたプログラムを機械語に翻訳してくれます。次に、機械語に翻訳されたファイル（xxx.hex）を、フラッシュマイクロコントローラのフラッシュメモリに書き込みます。フラッシュメモリは、不揮発性メモリなので、チップに電力を供給しなくても、書き込まれたプログラム内容は消えません。マイクロコントローラのフラッシュメモリに書き込む装置のことをプログラムライタと呼びます。本書では、世界一簡単で安価なプログラムライタの作り方を紹介します。

また、この書き込むためのソフトウエアツールもオープンソースで提供されていま

す。avrdudeと呼ばれる書き込み支援ツールを使うと簡単にマイクロコントローラに書き込めます。avr-gccやavrdudeなどのツールが一つのオープンソースパッケージになっていて、WinAVRと呼ばれています。安価なネットブックなどのWindowsOS上で開発するのであれば、winAVRパッケージをパソコンにインストールさえすればよいわけです。MacOSやLinux用の開発ツールパッケージもインターネット上に公開してあります。著者がリビルドした開発ツールパッケージも紹介します。本書では、ネットブック上（WindowsOS、LinuxOS、FreeBSD、MacOS）でインターネット・ガジェットを設計・開発することを前提に解説していきます。

先ほどプロトコール（ブラックボックス）の話をしましたが、一般に、複雑なプロトコールはプロトコールスタックと呼ばれるソフトウエア郡により実装されています。スタックというのは、階層構造になっているという意味です。プロトコールスタックの多くはＣ言語で書かれているので、比較的簡単に、自作のＣプログラムに組み入れることが出来るのです。さらに複数のプロトコールスタック同士を組み込むには、かなりのノウハウが必要ですが、本書ではソフトウエアUSB、ソフトウエアI2C、ソフトウエアUARTの3つ（実は４つ*）を組み込んだ例を説明します。

*通常、マイクロコントローラを正確に駆動するために、水晶発振子を使って設計しますが、本書では、マイクロコントローラ内蔵のPLL(Phase Locked Loop)機能を利用して、水晶発振子を使わない設計方法も紹介します。この機能を使うことによって、さらに安価にガジェットが構築でき、水晶発振のためのピンも別の用途で有効に使えるので、小さな８ピンのAVRチップ能力を最大限に活用できます。

最新のＣコンパイラを使っていても、訳の分からないエラーメッセージをコンパイラが頻繁に出力してきます。どのようなプログラミングミスの時に、どのようなエラーメッセージを出すのかを理解しておけば、次第に自作プログラムのデバッグが早くなってきます。コンパイルエラーの中で比較的多いのが、“;”（セミコロン）の記入漏れ、各変数などのスペルミスです。Ｃ言語では、大文字と小文字を識別しているので注意が必要です。問題解決のためには、インターネット検索の役割はたいへん重要です。

コンパイラとはコンピュータ言語（本書ではＣ言語）で書かれたプログラムをマシン実行コードに翻訳するツールです。本書では、２つのＣコンパイラを活用します。

パソコン32ビットintel_CPU用のコンパイラ(gcc)と８ビットマイクロコントローラ用のコンパイラ(avr-gcc)の２つです。パソコン上のコンパイラはnativeコンパイラと呼ばれ、avr-gccなどのコンパイラはクロスコンパイラと呼ばれます。iPhone/iPod Touch用のガジェットを作成する場合は、arm_CPU用のnativeコンパイラかクロスコンパイラの準備が必要です。本書でも、簡単に解説します。

ハードウエア上にソフトウエアを組み込んだものがシステムです。常に、ハードウエアを考えながらプログラミングしていきます。Ｃ言語では、プログラム中に現れるものは、すべて関数です。つまり、プログラムとは関数のかたまりです。関数が計算結果を返す場合は、“return”関数で返します。return値を返す関数は、必ず関数名の前にreturn値と連動してその関数のデータ型を宣言しなくてはいけません。関数の初めと終わりは、“{”記号と“}”記号で表します。関数名の後には必ず引数宣言記号“(”と“)”記号が必要です。引数がある場合は、() 内で宣言します。Ｃ言語のコンパイラは、“;”セミコロンあるいは中カッコ“}”のところまでを一つの命令と解釈しています。

例えば、exampleという関数は、整数の型（int:16ビット）で変数を返します。この例では、引数はありません。

```
int example(){
int xxx;
…
return xxx;
}
```

このexample関数の結果を変数retに代入できます。

```
int ret;
ret=example();
```

プログラム中で使われる変数・定数には、ローカルとグローバルがあります。関数の外で定義してあればグローバルになります。関数内で定義してあればローカルになります。グローバルな変数・定数は、どの関数内でも利用できます。ローカルな

変数・定数は、ローカルな関数内でのみ利用できます。

```
int global;
int example(){
int local;
…
return local;
}
```

関数は、()内の複数の引数を取ることが出来ます。引数はローカル関数内でのみ利用できます。下のsample(x)関数は、引数を２倍にして計算結果を返します。

```
int sample(int x){
return 2*x;
}
```

sample(x)の結果は、次のようにret変数に代入できます。この場合関数への引数は８となります。変数retには、16が代入されます。

```
int ret;
ret=sample(8);
```

初心者がつまずき易いのは、プログラミング中のデータ型(data type)です。使う環境（OSやCPU）によって、同じＣ言語でもデータ型は異なります。Ｃ言語のプログラミングでは、データを表現するのに決まったデータ型のみが利用できます。本書では８ビットのAVRマイクロコントローラを用いるので、AVR-libcと呼ばれるライブラリから供給されるデータ型のみが利用可能です。次の７つのデータ型のみ利用できるということです。shortはintと等しく2 byteであり、floatはdoubleと等価なので、実際には５種類のデータ型のみが利用できます。

```
(signed/unsigned) char - 1 byte
(signed/unsigned) short - 2 bytes
(signed/unsigned) int - 2 bytes
```

(signed/unsigned) long - 4 bytes
(signed/unsigned) long long - 8 bytes
float - 4 bytes (floating point)
double - alias to float

signed とは符号付、unsigned とは符号なしのことです。また、1 byte とは、8-bit のことを意味します。1 bit では、０か１の表現が可能です。例えば、“unsigned char” の型宣言があると、プログラムでは 1 byte のデータが準備され、0 から 2^8-1 の整数値が表現できます。“signed char” では、1 byte のデータで、-2^7 から 2^7-1 の整数値が表現できます。同様に、“unsigned int” では 0 から 2^{16}-1 の整数値、“signed int” では-2^{16} から 2^{16}-1 の整数値の表現が可能です。float と double は、３２-bit 浮動小数点です。浮動小数点は、仮数部は２3bits、指数部は符号付の 8bits、符号部に 1bit で表現されます。32bit の浮動小数点は、下記により表現されます。

$$(-1)^{\text{符号部}} \times 2^{\text{指数部}-127} \times \ (1+\text{仮数部})$$

３２ビットintel_CPUパソコンの場合はGNU Cコンパイラ(gcc)ライブラリでは、データ型は、次のようになります。

char 1 byte
short 2 bytes
int   4 bytes
long 4 bytes
long long 8 bytes
float 4 bytes
double 8 bytes

データ型が不明な場合は、次のdatatype.cプログラムをコンパイルして実行してみましょう。cygwinをインストールした後であれば実行可能です。コンパイラのインストールやコンパイルの仕方に関して、本書で詳しく説明します。

cygwinというツールをWindowsOSにインストールした後(本書の開発環境のところで詳しく説明します)、次のwget命令を実行します。

```
$ wget http://web.sfc.keio.ac.jp/~takefuji/datatype.c
$ cat datatype.c
int main(){
short s;
int i;
long l;
long long ll;
float f;
double d;
printf("short=%d int=%d long=%d long long=%d float=%d double=%d",
sizeof(s),sizeof(i),sizeof(l),sizeof(ll),sizeof(f),sizeof(d));
}
$ gcc datatype.c -o datatype
$ ./datatype
short=2 int=4 long=4 long long=8 float=4 double=8
$
```

データ型に関しては、プログラミングのところでもう一度詳しく解説します。

オペレーティングシステム(OS: Windows, Linux, MacOS, FreeBSD、Unix)を搭載しているパソコンや携帯端末では、ディレクトリ (Windowsではフォルダ) やファイルは、すべて木構造になっています。例えばWindowsでは、ハードディスクがc:であるすると、その下に、“Documents and Settings”、“Program Files”、“WINDOWS”、“WinAVR”、“cygwin”、…、などのディレクトリがあります。それらのディレクトリの先には、ディレクトリやファイルが存在します。ディレクトリやファイルの場所は、パス(PATH)と呼ばれます所在経路で表現されます。Windowsでは、プログラム→アクセサリ→コマンドプロンプトを起動してください。コマンドプロンプト画面で、“tree”と入力してください。画面に木構造が表示されます。ディレクトリやファイルの所在場所は、パス(PATH)で表現されます。

本書では、半田付けを使わなくても、インターネット・ガジェットの設計・開発ができるようにしました。必要な道具は、コードストリッパー（ダイソー：420円)、ニッパー（ダイソー：100円)、万能バサミなどです。単線0.65㎜を使ってブレッドボードに配線します。USBガジェットやiPod Touch/iPhoneガジェットでは、若干の半田付けが必要です。

インターネット・ガジェットを設計・製作し、望み通りの動作をさせるためには、
1　ガジェットのハードウエア設計・実装が正しいこと、
2　ガジェットのソフトウエア（ファームウエアとよぶ）設計・実装が正しいこと、
3　ガジェットと通信するネットワーク機器（パソコンなど）のアプリケーションソフトウエア設計・実装が正しいこと、
4　ガジェットのインターフェイス設計・実装が正しく、正常に動作していること、
5　ガジェットにつながっている機器も、ネットワークも正常に動作していること、
6　ガジェットのオペレーション・使い方も正しいこと、
これら６つの条件がすべて満たされる必要があります。インターネット・ガジェットが正常に動作するための条件は、初心者にとっては、かなり大きなハードルになります。本書では、初心者が起こしやすいミスの事例を紹介しながら、できるだけ最終的に読者が6つの条件を満たせるように解説していきます。

本書では、インターネット・ガジェット構築のためのソフトウエア開発ツール環境（WinAVR）にも簡単に触れながら、オープンソース環境でプログラム構築できるように心がけています。つまり、必要不可欠なソフトウエア開発ツールはオープンソースであり、インターネットからすべて無償でダウンロード可能です。自信がついてきたら、読者の皆さんの作品をインターネットに公開したら良いかと思います。

最後に、困ったときのインターネット検索の極意をまとめてみました。
google検索のヒントのまとめ
１．+xxx -yyy　　：+演算はxxx単語を含む、－演算はyyy単語を含まない
２．”xxx yyy”　:フレーズとは、”記号で囲まれた句の検索。単語xxxと単語yyyの並びと”yyy xxx”の結果は異なります。フレーズでも+-演算を使用できます。

+”xxx yyy” -”zzz ttt”
3.filetype:pdf　:ファイルタイプ検索、その他のファイルタイプにppt、doc、zip、…、c。
４．site:go.jp　　:ドメインサイト検索
５．daterange:2455100-2455191　:julian検索、ファイルの日付検索

問題：signed long longとunsigned long longの整数表現範囲を示しなさい。

問題：USBには、２つのＩＤがありますが、それはどのようなものか。

問題：I2Cは、何のための通信プロトコールか。

問題：UARTは何の略で、どのような通信方式か。

問題：オープンソースとクローズソースとは何か。

問題：UARTの標準設定とは何か、調べてみましょう。

問題：I2Cとは何か、調べてみましょう。

問題：1GHzとは、何ヘルツのことか。また、時間に変換すると何秒か。

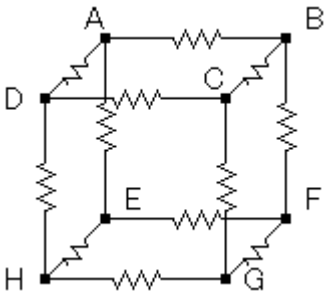
問題：Ｃ言語で、パソコン上のintとAVR上のintの大きさ（何バイト）を調べて見ましょう。

問題:iPod TouchかiPhoneを持っている場合、第何世代のものか調べてみましょう。

問題：Xbeeとは何か調べてみましょう。

問題：抵抗値Ｒが等しい１２本の抵抗が下図のように接続されています。ＣとＤ間の抵抗値、ＣとＨ間の抵抗値、ＡとＧ間の抵抗値をそれぞれ求めなさい。
ヒント：オームの法則を使います。同電位（同じ電圧）のところは、結合（接続）

してもかまいません。二点の間に電池を接続したときにどのように電流が流れるかを考えると、同電位になるところに気がつきます。



問題：プロトコールスタックとは何か？どうようなプロトコールスタックが世の中にあるのか、インターネット検索してみましょう。

問題：どのようなインターネット・ガジェットがあると便利か考えて見ましょう。

問題：“はじめに”のところで分からない単語をインターネット検索して、意味を調べておきましょう。

問題：julian検索とは何か。julian暦を計算する方法を調べましょう。

問題：C言語で型変換のためのCAST（キャスト）とは何か調べてみましょう。

重要なURLのリンク
http://cygwin.com
http://sourceforge.net/projects/winavr/

１．インターネット・ガジェット創作のための開発環境

インターネット・ガジェット構築のためには、必要最小限の開発環境が必要です。開発構築を安価にするために、ハードウエアとしてネットブックなどの3万円パソコン、AVRマイクロコントローラのフラッシュメモリに自作のプログラムを書き込むための専用プログラムライタが必要です。従来、専用プログラムライタを構築するのには、プログラムライタが必要でしたが（鶏と卵の問題と呼ばれています)、本書では、パソコンさえあればプログラムライタが簡単構築できる方法を解説します。

１．１　AVRチップ専用USBライタを組み立ててみよう

半田付けの必要がなく、もっとも工作が簡単で安上がりの方法を本書では解説していきます。USBライタとは、USBインターフェイスを介してユーザが作ったプログラムをAVRチップに書き込むための専用ライタのことです。

USBライタには、秋月電子通商で売られているFT232RLUSBシリアル変換モジュール（秋月950円）を利用します。AVRチップに書き込むためには、MISO、MOSI、SCK、RESETの４本の信号線と、電源供給のためのGND（グラウンド）とVCC（+5V）が必要です。また、AVRチップに外部からクロック供給が必要です。つまり、USBシリアル変換モジュールから６本の線がAVRチップに接続される必要があります。また、クロック供給のためにキャパシタ内蔵タイプのセラミック発振子（秋月20円）を外付けします。

USBシリアル変換モジュールとセラミック発振子とAVRチップのAtmega168（秋月230円）の3つの部品をブレッドボード（秋月250円）に図１．１のように挿し込みます。図１．１に示すように合計１3本の単線がブレッドボードに挿し込んであります。ブレッドボードに挿し込む線は、すべて**0.65ｍｍの単線**を使います。インターネットで購入するには、オヤイデ電気オンラインショップで１ｍ当たり63円です。秋葉原でショッピングするのであれば、JR秋葉原駅にあるタイガー無線で購入できます。合計予算1500円ぐらいでAVRチップ専用USBライタは完成できます。0.65mm以外の単線では、接触不良の原因になります。

USBシリアル変換モジュールはブレッドボードのｂ列とｆ列の1番から12番に差し込みます。Atmega168チップは、e列とｆ列の17番から30番に差し込みます。セラミック発振子（12MHｚセラロック）は、ｂ列の14番、15番、16番に差し込みます。

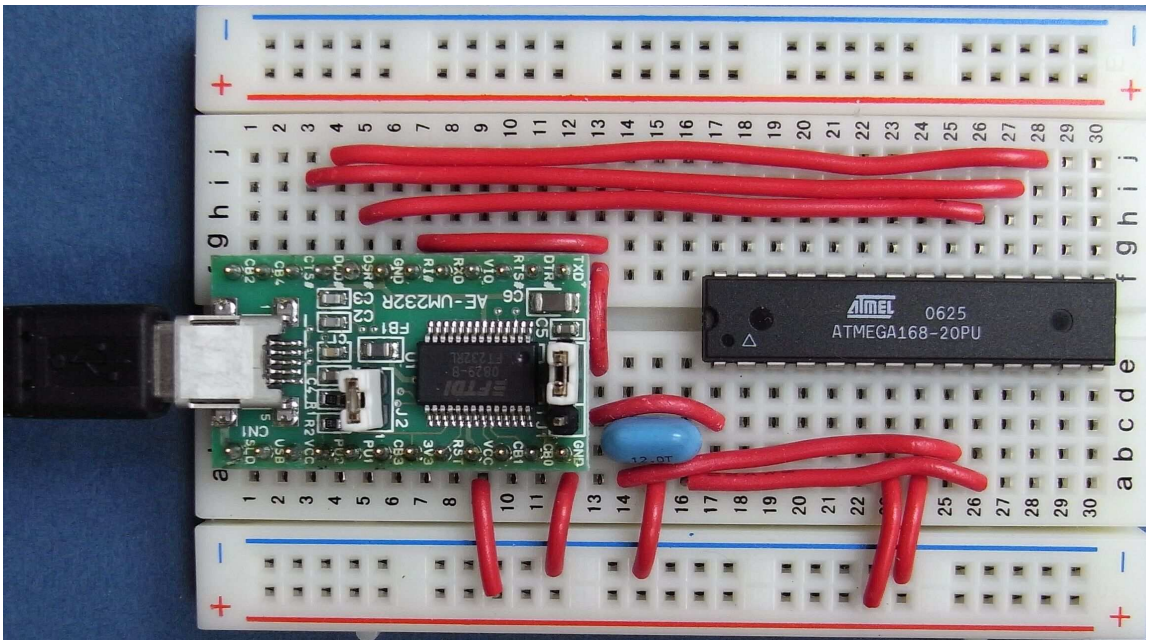


図1.1.1　AVRチップ専用USBライタ（Atmega168チップの例）

ブレッドボードでは、縦方向abcdeの５穴が電気的に結合されています。同様にfghijの５穴もつながっています。＋印は、１番から30番の30穴すべて横方向に電気的に結合されています。－印も同様です。図1.1.1に示すように13本の単線を間違えないように結線します。単線の皮をストリップする場合は、**６ｍｍから８ｍｍの銅線**が出るようにしてください。ストリップする銅線の長さが短い場合は、接触不良の原因になります。また、銅線が長すぎる場合は、ブレッドボードの下の方で、隣同士の銅線と接触する可能性があります。0.65mmの単線のストリッパーは、100円ショップでも売られています（100円以上しますが…）。

FT232RLUSBシリアル変換モジュールを少し詳しく調べてみましょう。図1.1.2に示すように、モジュールは２４ピンです。このモジュールの場合、USBミニ端子がついている方を、外側に向くようにブレッドボードのa列とf列に挿しています。**“USB－AとUSBミニのケーブル”**（秋月150円）をパソコンとモジュール間に接続します。

USBケーブルは何種類もあるので、間違えないように購入しましょう。

モジュールのGNDピンとVCCピンをそれぞれブレッドボードのー列と＋列に単線で接続します。モジュールのRI＃ピンは、Atmega168のRESET１番ピンに接続されます。モジュールのDSR#ピン、DCD#ピン、CTS#ピンは、それぞれAtmega168のSCK 19番ピン、MOSI 17番ピン、MISO 18番ピンに単線で接続します。

図1.1.3に示すように、セラミック発振子は3本足です。真ん中のピンがグランドになっているので、USBシリアル変換モジュールのGNDに接続します。残りの2本のピンは、Atmega168のXTAL1（9番ピン）とXTAL2(10番ピン)にそれぞれ接続します。セラミック発振子は発振周波数の精度が低いので、高い精度が必要な場合は水晶発振子を使います。

図1.1.4にAtmega168マイクロコントローラのピン配置を示します。Atmega168には、16Ｋbytes(1 byteは8ビット)のフラッシュメモリと、512bytesのEEPROMメモリと、1ＫbytesのSRAMメモリがあります。フラッシュメモリとEEPROMメモリは、不揮発性メモリと呼ばれ、電源を供給しなくても、メモリに書き込まれた内容は消えません。SRAMメモリは、電源を外すと、メモリ内の情報は消えます。このようなメモリを揮発性メモリと呼びます。その他、様々な機能設定のための内部レジスタが用意されています。

16Ｋbytesのフラッシュメモリにプログラムを書き込むわけですが、最近のハードディスク（数百ギガ）の大きさに比べてたいへん小さいので、直ぐにいっぱいになるのではないかと心配されるかもしれません。しかしながら、インターネット通信に使われている複雑なTCP／IPのプロトコールスタック（プログラム郡）がなんと8Ｋbytesしかないのです。USBプロトコールスタックは、1.5Ｋbytesしかありません。Atmega168の16Ｋbytesのプログラム空間は、結構大きいわけです。Ｃ言語プログラミングの中で使われる変数や定数は、Ｃコンパイラによって、すべて自動的にSRAMメモリに割り当てられます。EEPROMメモリは、データ測定値などを書き込む空間に使われたりします。

独自に作成したプログラム領域(ROM)とRAM領域の大きさを調べるには、次のchecksizeコマンドが便利です。main.binファイルは、main.hexを作成するときに

生成されます。この場合、ROM領域が1824バイト、RAM領域が69バイトです。

```
$ checksize main.bin
ROM: 1824 bytes (data=4)
RAM: 69 bytes
```

http://web.sfc.keio.ac.jp/~takefuji/checksize.zipからchecksizeをダウンロードしてください。
checksize.zipから解凍したchecksize.exeファイルを、cygwinの/binに格納します。解凍には、Lhaplusなどの解凍ソフトウエアをパソコンにインストールしてください。

```
$ mv checksize.exe /bin
```

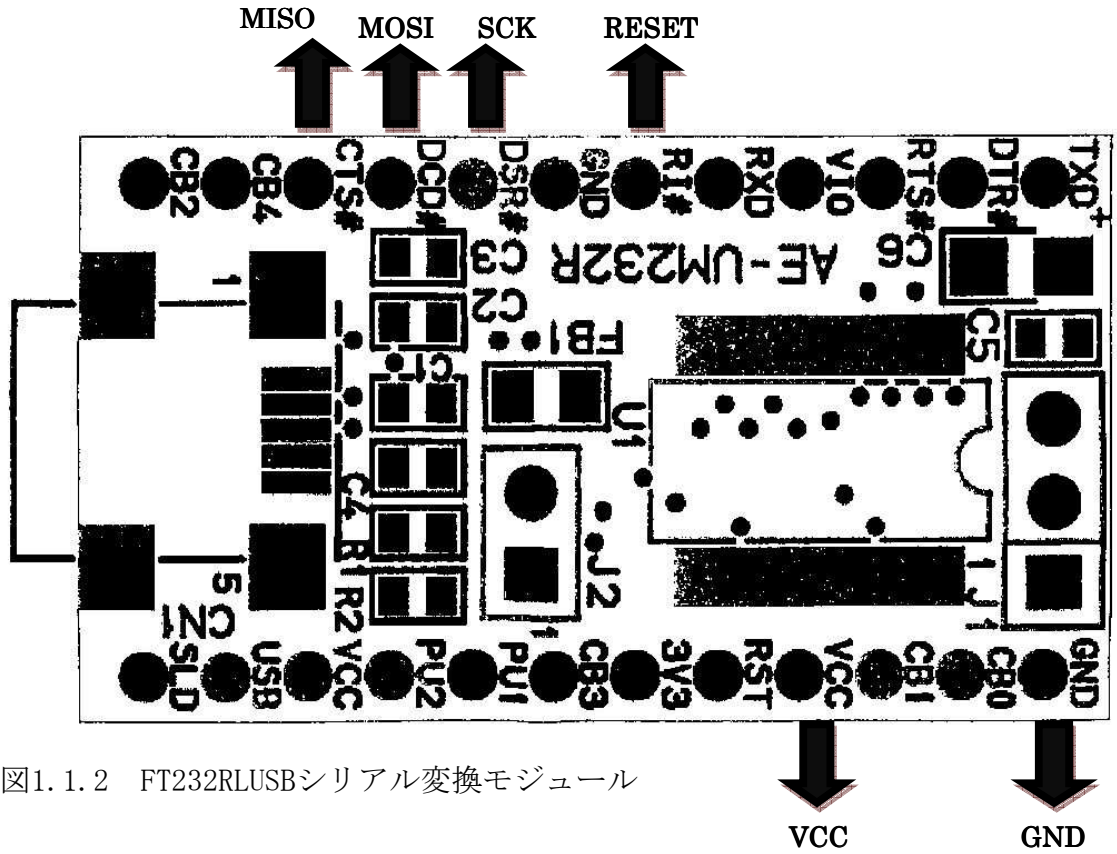


図1.1.2　FT232RLUSBシリアル変換モジュール

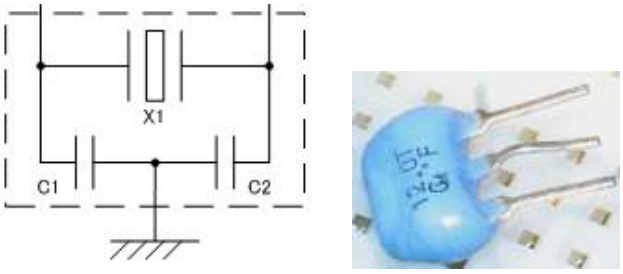


図1.1.3　3本足のセラミック発振子

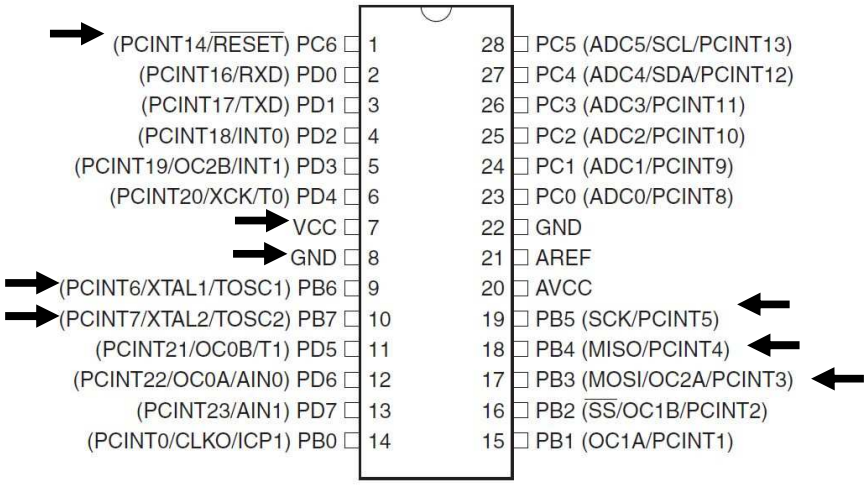


図1.1.4　Atmega168マイクロコントローラ（矢印は単線の結線が必要なピン）

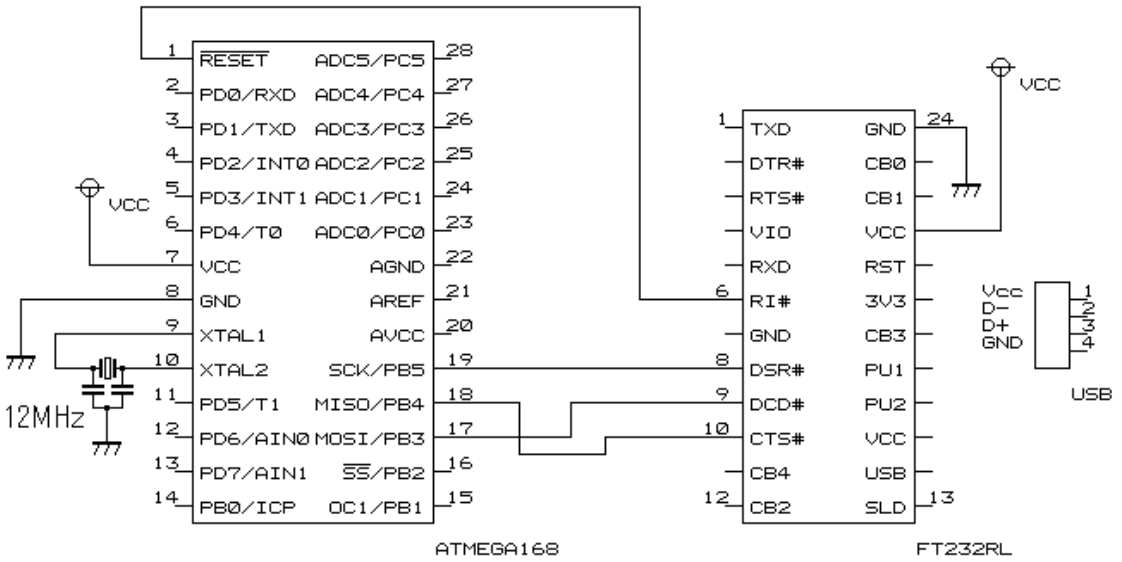


図1.1.5　AVRチップ専用USBライタ回路図

図1.1.5に示すように、回路図とはチップ間の配線情報を提供する地図です。図1.1.5に示すように、GNDは⏚印で表示され、すべてのGNDを接続します。できるだけGNDは一点接地します。その理由は、GNDの抵抗によって、各GND間に電位差が生じるのを防ぐためです。また、VCCも同様にすべてのVCCを接続します。VCCは、⊕印で表示します。ここでは、すべてのVCCは＋5Vに接続します。

FT232RLUSBシリアル変換モジュールを活用して、AVRチップ専用ライタにするには、パソコン側にソフトウエアインストールが必要です。ここでは、安価なネットブックなどのWindowsXPシステム上でのインストールの仕方を説明します。初心者にとって、インストールは面倒くさい作業なので、簡単にインストールできるようにしました。

下記のサイトからavrdudegui.exeファイルをダウンロードし、そのファイルをダブルクリックするだけで、FT232RLチップのデバイスドライバ（CDM 2.04.16.exe）を自動的に実行し、デバイスドライバをパソコンにインストールします。また、avrdudegui(yuki-lab.jp version 1.0.5)プログラムもパソコンに同時にインストールします。

http://web.sfc.keio.ac.jp/~takefuji/avrdudegui.exe

インストールした後、AVRチップ専用USBライタとパソコンの間を“USB－AとUSBミニのケーブル”で接続します。接続すると、パソコンの画面に図1.1.6の表示が現れます。真ん中を選択して、“次へ”をクリックします。図1.1.7のように、推奨を選択し、“次へ”をクリックします。

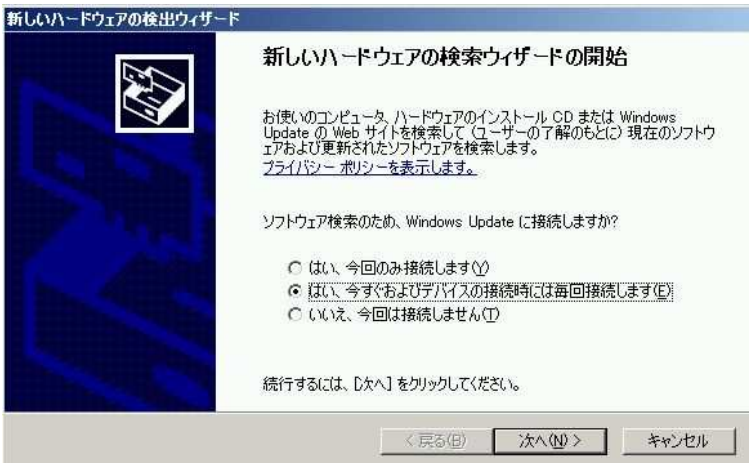


図1.1.6　AVRチップ専用USBライタをパソコン接続時の初期画面

図1.1.7　AVRチップ専用USBライタをパソコン接続時

図1.1.8の画面がでれば、デバイスドライバのインストールは完了です。

図1.1.8 AVRチップ専用USBライタをパソコン接続時

次に、パソコンのデスクトップ上に出来た、“avrdudegui”ファイルをダブルクリックしてください。もしも、図1.1.9のようなエラーが出る場合は、
http://web.sfc.keio.ac.jp/~takefuji/dotnetfx.exe
のdotnetfx.exeをダウンロードして、dotnetfx.exeファイルをダブルクリックし、.Net framework 2.0をパソコンにインストールします。

図1.1.9 .Net framework 2.0インストールが必要なエラーメッセージ

avrdudeguiが起動すると図1.1.20が表示され、ProgrammerにはFT232R Synchronous BitBang(diecimila)を選択してください。また、DeviceにはAtmega168を選択し、Command line Optionには、“-P ft0 -B 4800”の文字列を入力します。ソフトウエアとハードウエアの検証のために、FuseのReadボタンをクリックして、hFuse、lFuse、eFuse、が読み込めれば、すべてうまくいっている可能性があります。hFuse=DF、

lFuse=62、eFuse=01であれば、正常です。

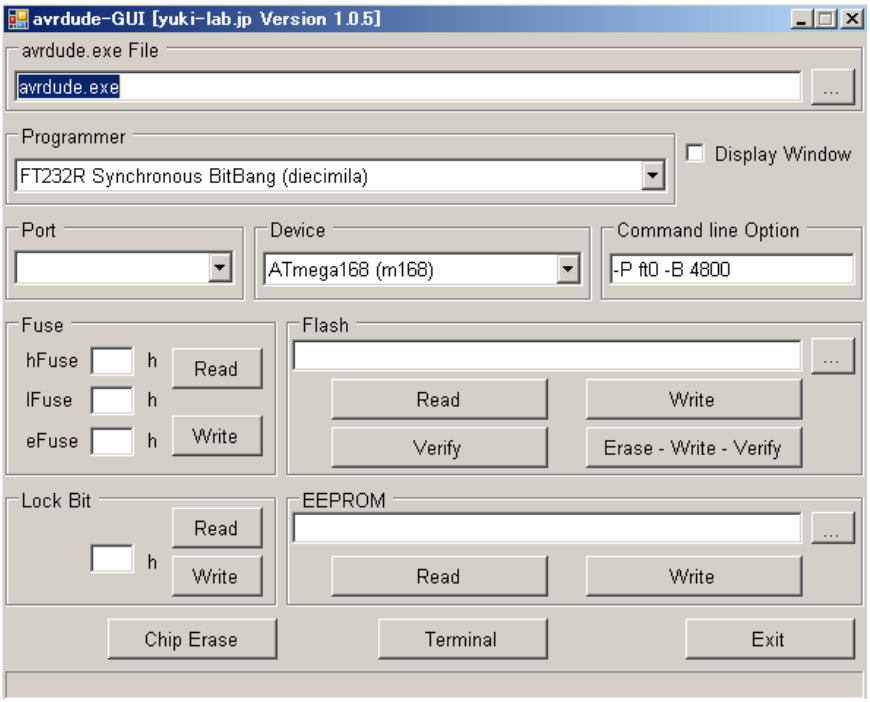


図1.1.20　avrdudeguiの設定画面

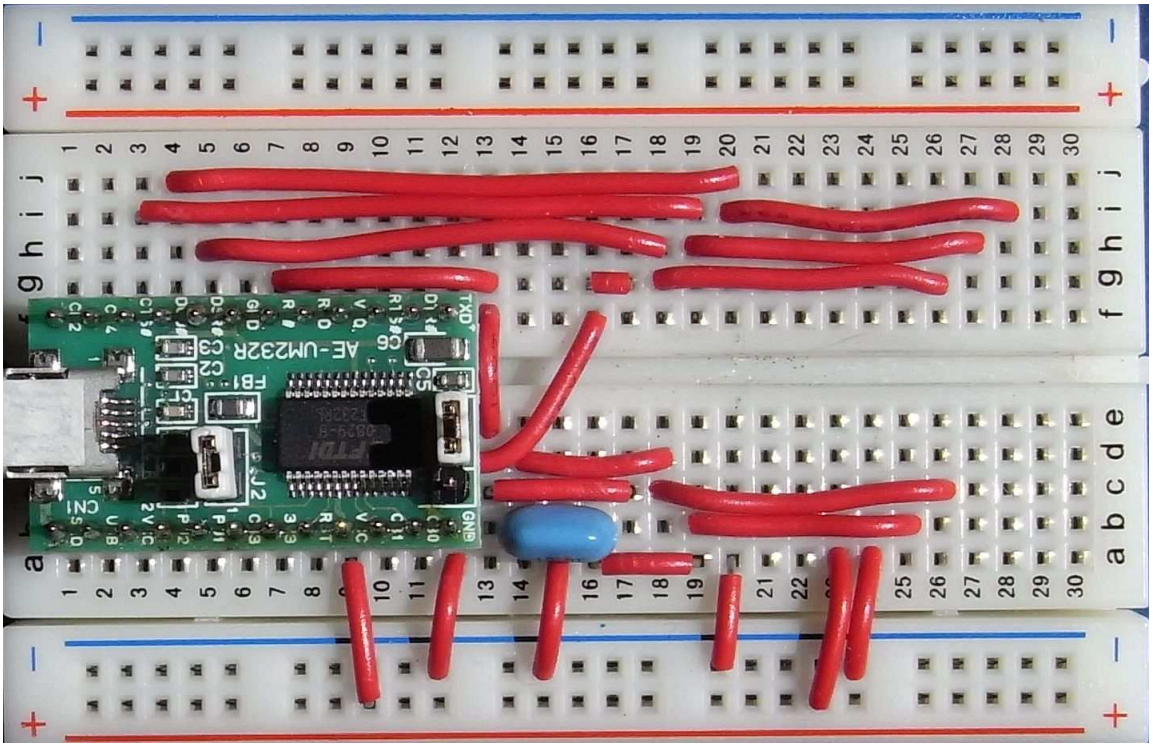


図1.1.21 Atmega168とTiny85も可能にしたAVRライタ配線回路

図1.1.21のようにAVRライタ回路を配線すると、Atmega168だけでなく、８ピンのTiny85なども読み書き可能になるのでたいへん便利です。AVRの専用ライタは、必要な配線とブレッドボードにFT232RLUSBシリアル変換モジュールを挿すだけで完成します。

問題：さまざまなUSB型AVRライタが世の中にありますが、もっと簡単なUSB型AVRライタがあるか調べてみましょう。

１．２　CygwinとWinAVRツールのインストールとLED表示回路

Cygwinのインストール

LinuxやFreeBSD、MacOSであれば、cygwinをインストールする必要はありませんが、WindowsOS(XP、VISTA、Windows7)であれば、下記サイトからsetup.exeファイルをダウンロードして、setup.exeファイルをダブルクリックします。

http://cygwin.com/setup.exe

セットアップは次の手順で実行してください。

1. “Install from Internet”を選択します。
2. Root Directoryに“C:¥cygwin”を入力します。
3. Local Package Directoryに“C:¥cygwin”を入力します。
4. Direct Connectionを選択します。
5. ftp://ftp.jaist.ac.jpなどの日本サイトを選択します。
6. 必要なパッケージをインストールします。
   ここで最小限必要なパッケージとは、gcc-core、gcc-g++、 wget、 tar、vim、unzipとすべてのLibs(ライブラリ)をインストールします。後からでも、他のパッケージをインストールできるので、必要に応じてパッケージをインストールしてください。

WinAVRのインストール

下記サイトから、WinAVRツールインストールファイルをダウンロードします。

http://sourceforge.net/projects/winavr/

ダウンロードしたWinAVR-xxx-install.exeファイルをダブルクリックして、パソコンにインストールします。図1.2.1の表示がパソコンの画面に現れます。ＯＫをクリックすると、図1.2.2が表示されるので、“次へ”をクリックします。図1.2.3に示すように、“同意する”をクリックします。図1.2.4に示すように、WinAVRのインストール先を設定します。ここでは、“c:¥WinAVR”を設定します。図1.2.5のように、インストールをスタートします。完了画面がでれば、WinAVRのインストールは終了です。

図1.2.1　WinAVRインストール初期画面

図1.2.2　WinAVRインストール画面２

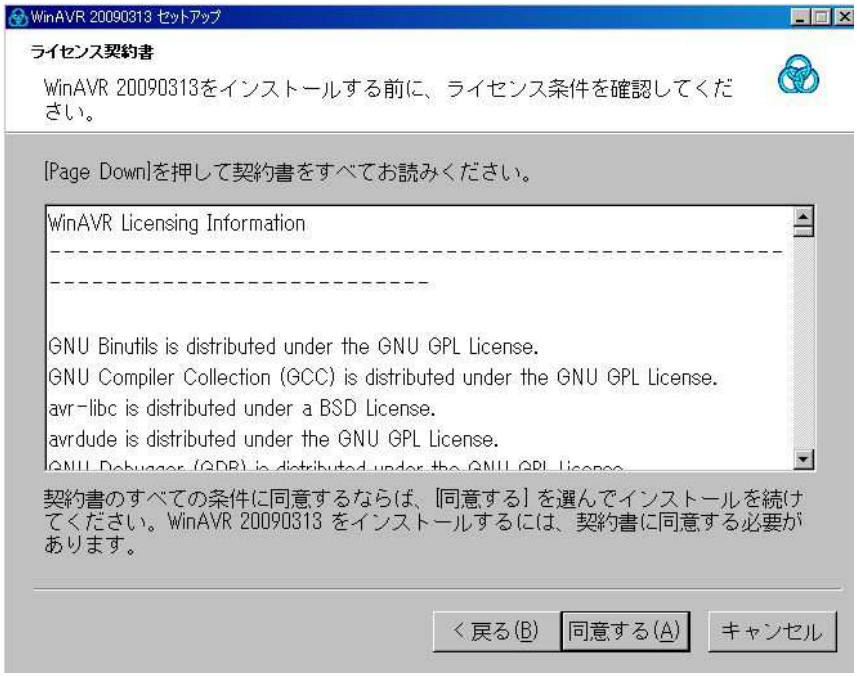


図1.2.3 WinAVRライセンス画面

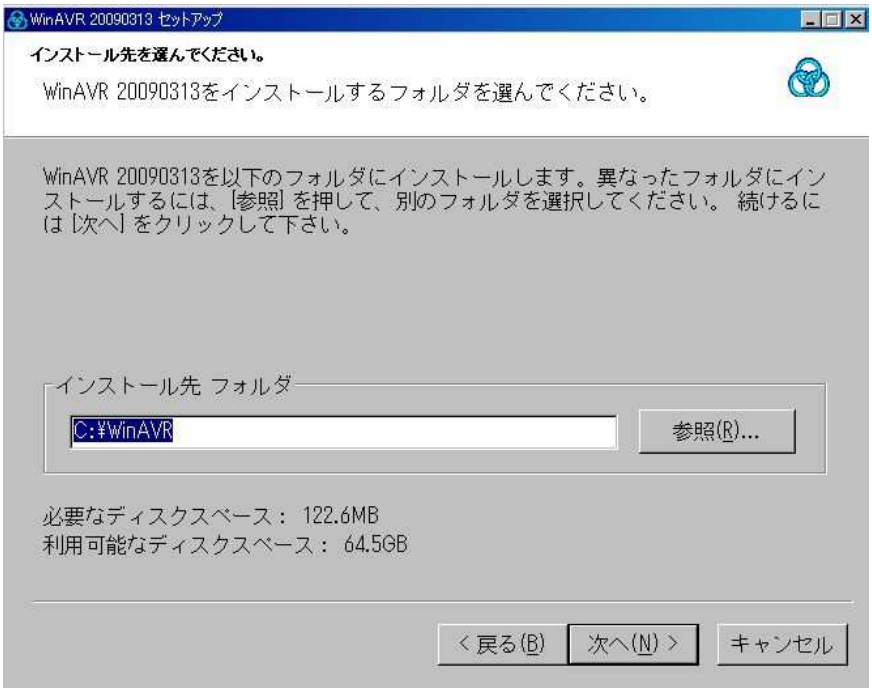


図1.2.4　WinAVRインストール先の設定

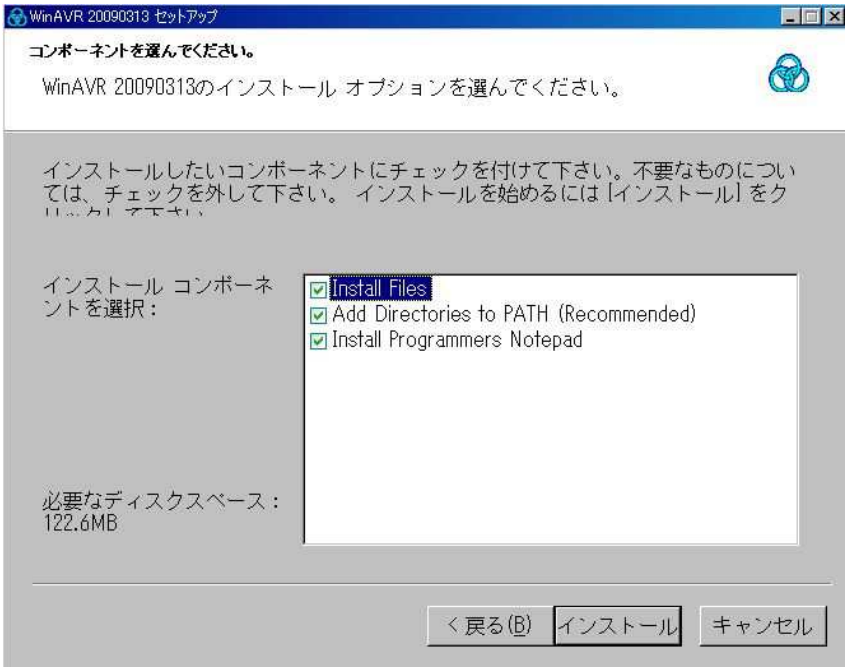


図1.2.5　インストール開始画面

WinAVRツールがパソコンに正しくインストールされているかどうか確かめるために、下記サイトからled0-168.zipファイルをダウンロードして、解凍します。

http://web.sfc.keio.ac.jp/~takefuji/led0-168.zip

led0-168.zipファイルを解凍すると、図1.2.6のように4つのファイルが生成されます。

図1.2.6　テスト用ファイル

led-168.pnprojファイルをダブルクリックすると、図1.2.7に示すように Programmer’s Notepadが立ち上がります。ここで、Toolsメニューの“Make All”を実行します。実行後に図1.2.8のように、main.hexファイルが生成されます。

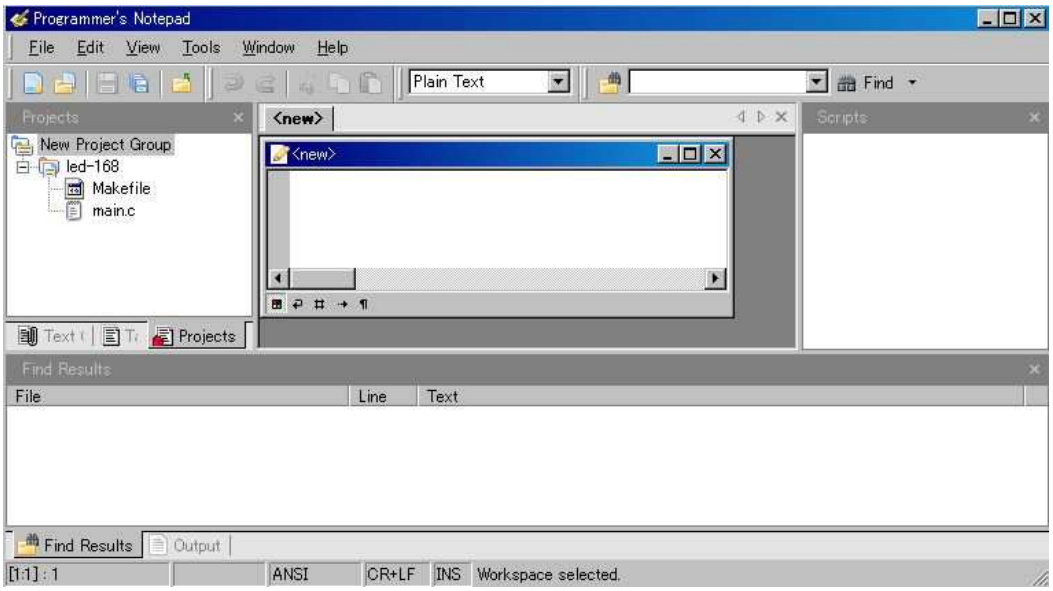


図1.2.7　Programmer’s Notepadの起動

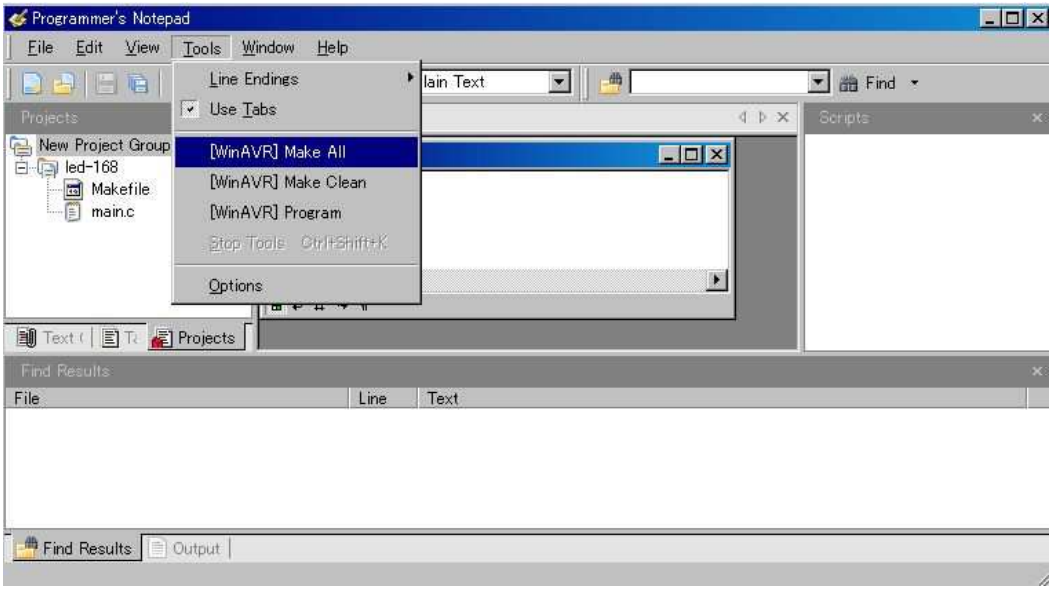


図1.2.8　Make Allを実行する

ここで図1.2.9に示すように、avrdudeguiのFlashメニューの矢印ボタンをクリックし、main.hexファイル開きます。パソコンとUSBライタが接続していることを確認し、Atmega168がライタに挿してあることもチェックしてください。次にErase-Write-Verifyボタンをクリックすれば、main.hexプログラムをAtmega168チップに書き込みます。

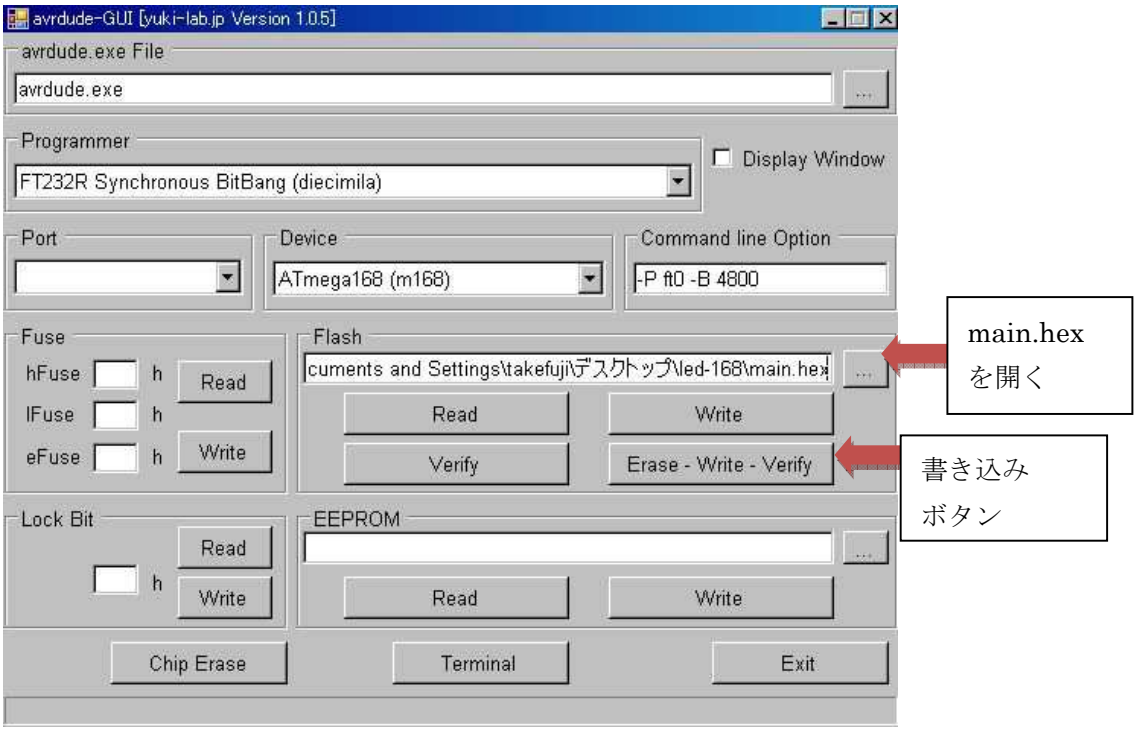


図1.2.9 main.hexファイルを開き、書き込みボタンのクリック

ハードウエアとしては、図1.2.10の小型のブレッドボード（秋月150円）に10本の単線を接続します。

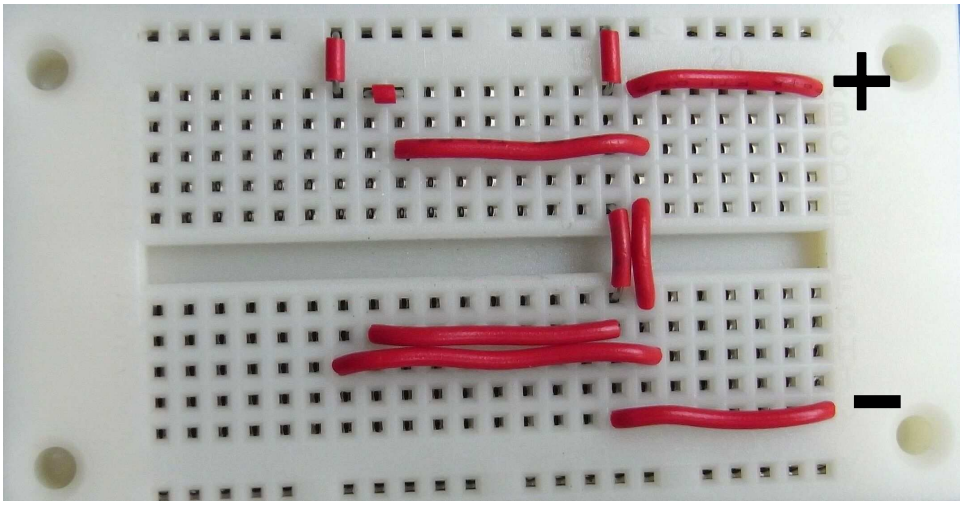


図1.2.10　テスト用ブレッドボード

先ほど、書き込んだAtmega168と超高輝度LED（赤色）を図1.2.11のように接続します。LEDはAtmega168の28番ピンとGNDとの間に接続します。CR2032ボタン電池（3V）をボタン電池基板取付用ホルダー に入れて、図のようにブレッドボードの端に差し込みます。

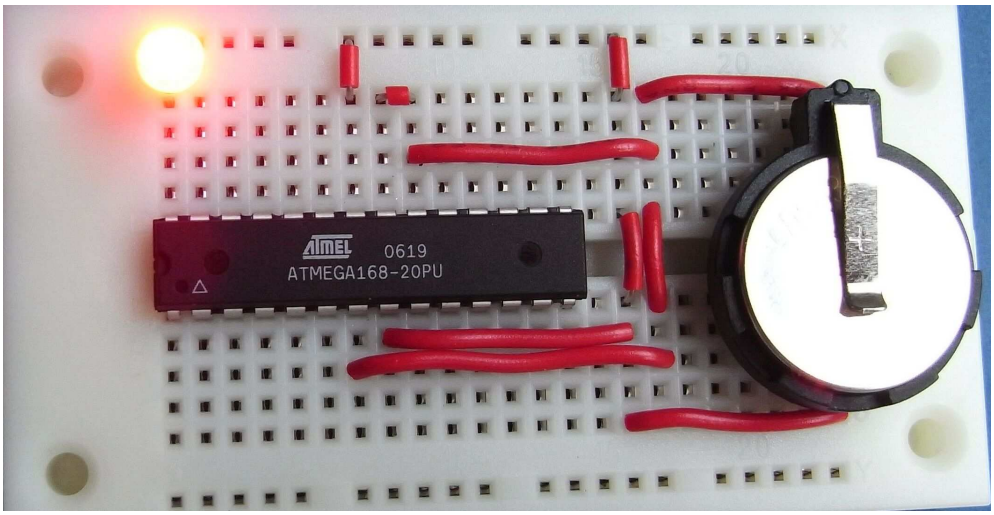


図1.2.11　LEDテストプログラム

超高輝度赤色LED（５ｍｍ　18cd）は、10個で200円です（秋月）。18cdは18カンデラのことです。図1.2.12に示すように、LEDの2本足のうち根元の太い足の方がカソードでGNDに接続します。細い足の方がアノードです。LEDのアノードは、Atmega168の２８番ピンにつながっています。

LEDはダイオードの一種です。両端に順方向（アノードの方が高い電位でカソードの方が低い電位）にかける電圧を上げていくとあるところで急に電流が流れるようになります。この電圧のことを、Vf（Forward Voltage）と呼んでいます。簡単に言うと、Vf以上の電圧を与えないとLEDは光りませんし、電流も流れません。この超高輝度LEDは、Vfが2.2Ｖです。メーカーによって、このVfは±20%ほど変動します。

図1.2.12　超高輝度赤色LED（Vf：2.2V、５ｍｍサイズ）

AVRライタがパソコンで認識できなければ、下記サイトからFT232用のドライバをパソコンにインストールしてください。
http://www.ftdichip.com/Drivers/D2XX.htm
http://www.ftdichip.com/Drivers/VCP.htm

問題：cygwinのコマンドは、UnixOS(LinuxやFreeBSDやMacOS)で使われている便利なコマンドです。次のコマンドを実行して、その結果を調べてみましょう。

```
$ set
$ set|grep PATH
$ pwd
$ which avr-gcc
$ cd ..; ls
$ cd c:; ls
```

問題：cygwinでは、bashシェルが実行されています。頻繁に使うコマンドに、which、grep、ls、ps、ipconfig、pwd、cd、vim、cat、date、ssh、scp、wget、tar、unzipがあります。それぞれのコマンドの意味と使い方を練習しましょう。

１．３　LED点滅表示のためのＣ言語プログラミング

１．２章で実験した小さなＣプログラムを見てみましょう。led0-168.zipのファイル内のmain.cはプログラムソースと呼ばれます。main.cのプログラムソースを図1.3.1に示します。

```
/* LED flash using atmega168 designed by takefuji on sept. 22, 2009*/
#define F_CPU 1.0E6       //1MHzのクロック周波数を宣言
#include <avr/io.h>       //入出力ライブラリ
#include <util/delay.h>  //遅延時間ライブラリ

int main(void)
{
for(;;){DDRC=0x20;
        PORTC=0x20;_delay_ms(500);
        PORTC=0x00;_delay_ms(500);}
}
```

図1.3.1　led0-168.zipのmain.cプログラム(1秒毎の点滅)

Ｃ言語のプログラミングでは、すべての表現は関数として考えます。プログラムは関数の集合体であり、その中には必ず一つのmain関数が必要です。常に、main関数からプログラム実行を開始します。また、main関数の中の上から下の行に向かって逐次実行していきます。もちろん、割り込みの場合は、割り込み機能が起動されると現在のプログラムを中断して、割り込み処理をしてから、元の状態に戻り継続します。

図1.3.1では、2文字記号“/*”と2文字記号“*/”の間に挟まれた文字列は、すべてコメント行になります。複数行でもコメント可能です。また、“//”記号の後ろもコメントとなります。これは、1行だけです。

“#define”、“#include”、“int main(void)”などが理解できれば、このプログラムが何をしようとしているのか理解できるかもしれません。

まず、“#define”はユーザが定義できる、定義文です。ここでは、F_CPU（グロー

バル変数）は1.0E6、つまりAtmega168チップのクロック周波数は、10^6Ｈｚ＝１ＭＨｚであることを宣言します。すべてのプログラム実行は、クロックによって制御されます。AVRチップは、RISC（reduced instruction set computer）タイプのコンピュータなので、複雑な命令を除いて、多くの命令セットは、１クロックで1命令を実行できます。

“#include”のincludeは含みなさいとコンパイラに命令しています。具体的には、“#include”は、avr-gccのＣ言語コンパイラのライブラリ（AVR-libc）を利用しなさいと宣言しています。xxx.hはライブラリのヘッダーファイルと呼ばれ、includeディレクトリ置いてあるファイルで、利用者が理解できるＣ言語で記述されています。そのxxx.hの中では、いろいろな関数などが定義してあります。実際のライブラリは、libディレクトリにある実行ファイルです。コンパイラはそれらのライブラリ実行ファイルをリンクします。

<avr/io.h>は、Atmega168チップの入出力機能を使うために入出力ライブラリを使うことを宣言しています。同様に、<util/delay.h>は遅延関数を使うことを宣言しています。<avr/io.h>ライブラリ情報は、パソコンのC:¥WinAVR¥AVR¥include¥AVRにあり、<util/delay.h>ライブラリ情報は、C:¥WinAVR¥AVR¥include¥utilにあります。この遅延関数は先ほどのグローバル変数F_CPUと連動していて、CPUスピードを定義することによって、自動的に正確な遅延時間を作ってくれます。
_delay_ms(500);は500ms（ミリ秒）の遅延時間を作ってくれます。

Ｃ言語では、必ずプログラム中にmain関数がなくてはいけません。int main(void)とは、main関数は整数関数であり、戻り値は整数を返すと言う意味です。 main関数の“void”とは、この関数が引数を取らないことを意味します。“{”と“}”は重要なシンボルで、関数の初めと終わりを定義しています。つまり、{”と“}”で囲まれた部分は、一塊の関数ということです。“void xxx(void)”の関数の場合は、戻り値なしで引数なしのxxx関数になります。Ｃ言語では、main関数内で上から下の行に向かって逐次実行していきます。

この例では、main関数の中にfor文が一つあります。for(;;)は無限ループを意味します。つまり、forループの中を無限回繰り返せという実行です。つまり、強制終了しない限り、forループを無限回実行します。関数内で定義されていない変数は

グローバル変数です。ここでは、DDRC、PORTCなどがグローバル変数です。これらのグローバル変数は、先ほど説明した<avr/io.h>ライブラリで定義されています。

```
for(;;){DDRC=0x20;PORTC=0x20;_delay_ms(500);
        PORTC=0x00;_delay_ms(500);}
```

for文の中には、５つの関数があります。一つの関数は、セミコロン“;”で終了します。“DDRC=0x20;”は、代入文です。0x20とは、16進法（“0x”）で20のことを意味します。2進法で説明すると、DDRCに8bitのデータで0010 0000を代入すること表現しています。DDRCとは、Atmega168に用意してある内部レジスタで、Ｃポートの入出力設定を行います。DDRCが理解できなければ、Atmega168のPDFマニュアルを検索しましょう。すべてのAVRのレジスタは基本的に８ビット長です。

ポートとは、入出力ビットの集まりのことで、図1.1.4で示したようにAtmage168には、８ビットのＢポート、８ビットのＤポート、７ビットのＣポートの3つのポートがあります。Atmega168には、合計２3ビットの入出力ビットがあるということです。このプログラムでは、７ビットのＣポートの入出力を設定しています。入出力は、各ビット独立して設定できます。DDRCのいずれかのビットに１が立つと、そのビットは出力設定になります。また、DDRCのいずれかのビットが0であれば、そのビットは入力設定となります。DDRC=0x20;”とは、PC5（ＣポートPC5）を出力ビットに設定せよということです。Ｂポートであれば、DDRBが入出力設定レジスタになります。

LEDをＣポートの別ビットに接続しても、LEDは光らないはずです。なぜならば、Ｃポートの他のビットは、入力に設定されているからです。LEDが点滅するためには、０のデータと１のデータを交互にPC5に出力します。出力に設定したbitに対して、データを出力するには、PORTC=0x20;のようにPORTCに代入すればよいわけです。PORTC=0x00;はPORTCのPC5に０を出力します。PORTC=0x20;は１をPC5に出力します。図1.3.2のように3V（ボタン電池の電圧）と0Vのデータが PC5に交互に出力されるので、LEDが点滅するわけです。AVRマイクロコントローラの出力は、電源電圧の値が論理値1で、論理値0が0Vになります。

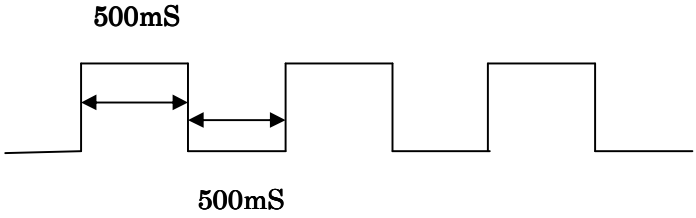


図1.3.2 PORTCのPC5に１と０を繰り返し出力する

次に、led1-168.zipを下記のサイトからダウンロードして、解凍します。

http://web.sfc.keio.ac.jp/~takefuji/led1-168.zip

led1-168.zipを解凍すると、図1.3.3に示すmain.cがあらわれます。

```
/* LED flash using atmega168 designed by takefuji on sept. 22, 2009*/
#define F_CPU 1.0E6     //1MHz clock
#include <avr/io.h>
#include <util/delay.h>

int main(void)
{
 unsigned char i;

for(;;){DDRC=0x20;
        for(i=0;i<10;i++){PORTC=0x20;_delay_ms(1); PORTC=0x00;_delay_ms(9);}
        _delay_ms(100);
                }
}
```

図1.3.3 led1-168.zipのmain.c（200m秒点滅）

led0-168のmain.cとの大きな違いは、“unsigned char i;”です。これは、iという８bitの符号なし変数を定義しています。先ほどは、５００m秒毎に１になったり０

になったりしましたが、この場合は、1m秒の１と9m秒の０を10回繰り返し、100m秒の間０を出力し、200m秒の周期を繰り返します。200m秒の周期で点滅しているように見えていても、光っている時の周期は10m秒です。このようにパルス幅をコントロールする方法を、PWM(Pulse Width Modulation) と呼びます。この点灯している時間幅を変化させることによって、電圧を変えることなく、PWM手法でLEDを暗くしたり明るく光らせたりコントロールできます。

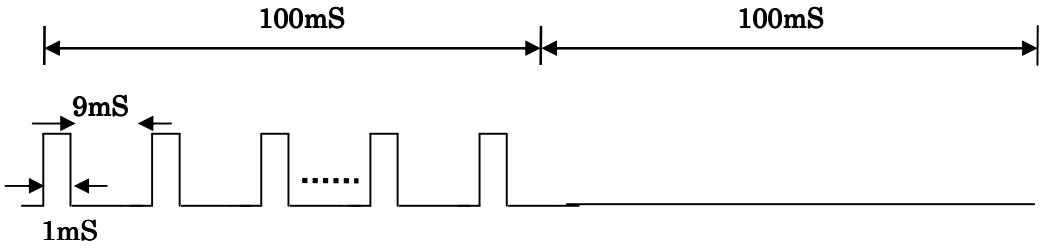


図1.3.4　PWM信号(光っている時のデューティ比１／10、周期200m秒)

超高輝度LEDの最大電流が100mAだとしても、PWMでは150mA流すことが可能になります。ただし、デューティ比をコントロールする必要があります。PWMでは、明るく光らせて、しかも消費電力を小さく出来ます。

デューティ比とは、１周期に占める１のパルス幅の割合のことです。この場合、周期は10m秒で１のパルス幅は１m秒なので、デューティ比は１／10になります。デューティ比を小さくすれば暗くなるわけです。

for(i=0;i<10;i++)は、10回のループを作っています。i=0から始まり、i++でiが１ずつインクリメントされ、i=９で終了します。forループでは、for(初期条件;終了条件;最初期化)になります。i++は、i=i+1と同じ意味になります。

for(i=0;i<10;i++){PORTC=0x20;_delay_ms(1); PORTC=0x00;_delay_ms(9);}を実行すると、図1.3.4に示すように、forループ内では、PORTC=0x20;によってPC5に１を出力し、１m秒の遅延、PORTC=0x00;によってPC5を０にして、９m秒の遅延を生み出します。for(i=0;i<10;i++)は、10m秒の周期（１m秒の１出力、９m秒の０の出力）を10回繰り返します。図1.3.4に示す200m秒の周期をfor(;;)によって、無限回繰り返します。

遅延関数には、_delay_ms(ｘ)のほかに_delay_us(ｘ)があります。_delay_us(ｘ)はμＳ（マイクロ秒）の遅延を作ります。変数ｘは浮動小数点になります。精度の高い遅延時間を作るには、_delay_loop_1(ｘ)と、_delay_loop_2(ｘ)の２種類が用意してあります。変数xは、_delay_loop_1では符号なし８bit、_delay_loop_2では符号なし１６bitを与えます。_delay_loop_1では、１ループあたり、3クロック分の遅延時間になります。_delay_loop_2では、１ループあたり、４クロックの分の遅延時間になります。例えば、CPUクロックが１ＭＨｚの場合、_delay_loop_1(5)は１μＳ＊3＊５＝１５μ秒の遅延時間を作ります。_delay_loop_2(８)は、
１μＳ＊４＊８＝３２μ秒の遅延時間になります。

問題：図1.3.5に示しすような、マンチェスターコードを生成するプログラムを構築せよ。ただし、Clockは１ＭＨｚで動作しています。

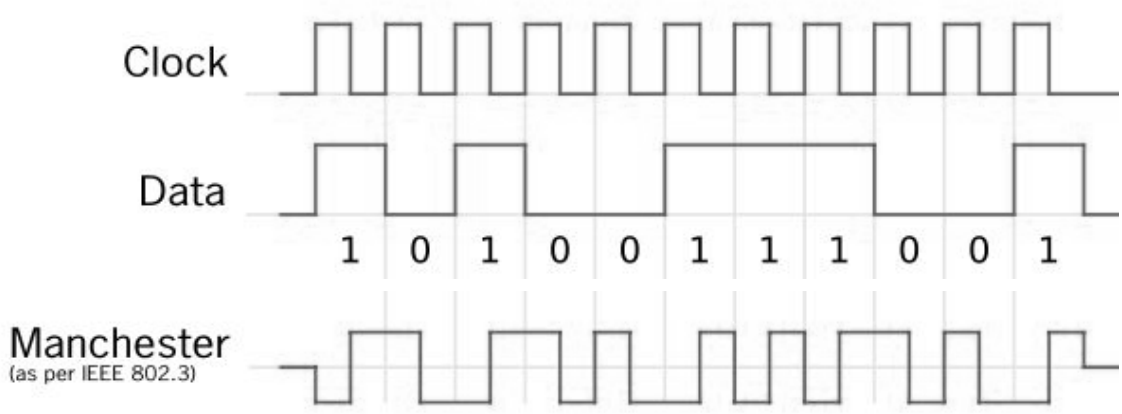


図 1.3.5 Manchester コード

5 Linux・MacOS・FreeBSDユーザのための開発ツール（avr-gcc, avrdude）

GNU開発ツールを使って、avr-gccとavrdudeクロスツールを別のOSで再構築する方法を紹介します。Linux、MacOS、FreeBSDなどさまざまなOSでも便利なツールを利用できるようになります。ここでは、Vine Linuxでの実行例を説明します。ここでは、binutilsツール、gcc-coreツール、avr-libcツール、avrdudeツールの４つを紹介します。それぞれのツールのバージョンには相性があります。相性を合わせることは大変重要です。相性を合わせないと、ライブラリの依存関係によって問題を引き起こします。

先ず、binutilsツールを作成します。本家のbinutilsファイルをダウンロードします。

```
$ wget ftp://ftp.gnu.org/gnu/binutils/binutils-2.19.1.tar.bz2
```

次に、ダウンロードしたファイルを解凍します。

```
$ tar xvf binutils-2.19.1.tar.bz2
```

binutils-2.19.1に移動します。

```
$ cd binutils-2.19.1
```

次に、configureを実行し、AVRツールのbinutilsを作成します。

```
$./configure --target=avr --prefix=/usr/local --disable-nls --enable-install-libbfd
```

次にmakeします。

```
$ make
```

makeして、インストールします。

```
$ sudo make install          管理者になって、インストールします。
```

パスワードを入力するとインストールが完了します。

次は、gcc-coreを作成します。下記サイトからgcc-coreをダウンロードします。

```
$ wget ftp://ftp.gnu.org/gnu/gcc/gcc-4.3.2/gcc-core-4.3.2.tar.bz2
```

workディレクトリを作成します。

```
$ mkdir work
$ cd work
$ ../configure --target=avr --prefix=/usr/local --disable-nls --disable-libssp
$ make
$ sudo make install
```

次は、avr-libcライブラリを作成します。

```
$ wget http://ftp.twaren.net/unix/NonGNU/avr-libc/avr-libc-1.6.7.tar.bz2

$ tar xvf avr-libc-1.6.7.tar.bz2
$ cd avr-libc-1.6.7
$ ./configure --build=`config.guess` --host=avr --prefix=/usr/local
$ make
$ sudo make install
```

最後に、avrdudeを作成します。３つのファイルをダウンロードし解凍します。

```
$ wget http://download.savannah.gnu.org/releases-noredirect/avrdude/avrdude-5.3.1.tar.gz
$ wget http://www.nmj.sakura.ne.jp/suz-AVR/serjtag/serjtag-0.3.tar.gz
$ wget http://www.ftdichip.com/Drivers/D2XX/Linux/libftd2xx0.4.16.tar.gz
$ tar xvf avrdude-5.3.1.tar.gz
$ tar xvf serjtag-0.3.tar.gz
$ tar xvf libftd2xx0.4.16.tar.gz
```

次に、avrdude-5.3.1ディレクトリに移動し、patchを当てます。

```
$ cd avrdude-5.3.1
$ patch <../serjtag-0.3/avrdude-serjtag/src/avrdude-5.3.1-USB910.patch
$ patch <../serjtag-0.3/avrdude-serjtag/src/avrdude-5.3.1-AVR910d.patch
$ patch <../serjtag-0.3/avrdude-serjtag/src/avrdude-5.3.1-serjtag.patch
$ patch <../serjtag-0.3/avrdude-serjtag/src/avrdude-5.3.1-ft245r.patch
$ patch <../serjtag-0.3/avrdude-serjtag/src/avrdude-5.3.1-baud.patch
$ cp ../libftd2xx0.4.16/ftd2xx.h ./
$ cp ../libftd2xx0.4.16/WinTypes.h ./
$ cp ../libftd2xx0.4.16/static_lib/libftd2xx.a.0.4.16 ./
```

次に、configure を実行します。

```
$ ./configure CFLAGS =” -g -O2 -DHAVE_LIBUSB -DSUPPORT_FT245R” LIBS=” -lncurses
-ltermcap -lftd2xx -lrt”
```

次に、Makefile の次の２箇所を変更します。

```
CFLAGS = -g -O2 -DHAVE_LIBUSB -DSUPPORT_FT245R
```

```
LIBS= -lncurses -ltermcap -lftd2xx -lrt
```
ここで、make します。
```
$ make
```

ここで、avrdude.conf ファイルに下記の９行を加えます。

```
Programmer
  id    = "chicken";
  desc  = "FT232R SynchronoUSBitBang";
  type  = ft245r;
  miso  = 3;  # CTS X3(1)
  sck   = 5;  # DSR X3(2)
  mosi  = 6;  # DCD X3(3)
  reset  = 7;  # RI  X3(4)
;
```

最後に、avrdude をインストールします。
```
$ sudo make install
```
AVR ライタを Linux マシンに接続し、ライタに Atmega168 を挿し、次の命令を実行してください。
```
$ avrdude -c chicken -p m168 -t -B 38400
```
次のメッセージが表示されれば、avrdude も問題なく稼動できています。
```
avrdude.exe: BitBang OK
avrdude.exe: pin assign miso 3 sck 5 mosi 6 reset 7
avrdude.exe: drain OK

 ft245r:  bitclk 28800 -> ft baud 14400
avrdude.exe: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.00s

avrdude.exe: Device signature = 0x1e9406
avrdude>
```

問題：ツールを作成する際に、何故、ライブラリの依存関係（相性）が重要なのか考えて見ましょう。ライブラリーのバージョンによって、ツール作成がうまくいく場合といかない場合があります。試してみましょう。

問題：ライブラリー同士の相性をあらかじめ知るには、どうすれば良いのか、調べてみましょう。

5.1 役立つシステムコマンド(expect, cron)

ネットワークプログラミングで便利なコマンドがexpectとcronです。expectは、ネットワークで“会話”するプログラムです。cronは定期的に実行する場合に便利な機能です。インターネットにつながったパソコンのセンサーガジェットを定期的にアクセスしたり、収集したセンサーデータをデータベースに格納したりするためには、複雑なプログラミングが必要です。しかしながら、expect機能とcron機能を使いこなすことで、目的のシステムが比較的簡単に構築できます。

iPod Touchでは、Cydiaやapt-getコマンドがパッケージインストーラですが、expect機能とcron機能が含まれていないので、携帯端末（iPod Touch/iPhone）やパソコンのためのexpect機能とcron機能のインストール手法をここでは紹介します。

iPod Touchでは、pythonのAPIであるpexpectを使って、データ収集してみます。cron機能は、Vixie Cronを構築してみます。

次のサイトからcron3.0pl1.tar.gzファイルをiPodにダウンロードし解凍します。スーパーユーザーになってから、次の命令を実行します。

```
root# wget ftp://metalab.unc.edu/pub/Linux/system/daemons/cron/cron3.0pl1.tar.gz
root# tar xvf cron3.0pl1.tar.gz
root# cd cron3.0pl1
```

Makefileファイルの１行を次のように変更します。

```
CC = arm-apple-darwin-gcc
```

root# make
クロスコンパイルの場合は、生成されたcronを/usr/sbinに移動させます。また、crontabを/usb/binに移動させます。nativeコンパイルの場合は、次の命令を携帯端末上で実行します。
root# make install
root# ldid -S /usr/bin/crontab
root# ldid -S /usr/sbin/cron
root# cron　　cronを実行します。
“crontab -e”を実行すると、vim(vi)のブランク画面が立ち上がります。vim(vi)の簡単な使い方は、iが入力でEscキーが入力の終了です。上下左右の矢印→カーソルでプロンプトを移動させます。プロンプト上の文字を消すのは、xキーです。
root# crontab -e　　crontabを実行します。
crontabの画面には、例えば、次のような情報(分，時，日，月，曜日)を記述します。

分　時　日　月　曜日　　実行コマンド
0-59 * * * * /var/root/test　　１分毎に/var/root/test命令を実行
0-59/5 * * * * /var/root/test　　５分毎に実行
59 23 * * 0 /var/root/test　　毎週日曜日の23時59分に実行

パソコンでは、cygwinからexpect(expect: A program that ‘talks’ to other programs)をインストールします。iPodでは、pyexpectをインストールするためには、次の命令を実行します。
root# wget http://pexpect.sourceforge.net/pexpect-2.3.tar.gz
root# tar xzf pexpect-2.3.tar.gz
root# cd pexpect-2.3
root# python ./setup.py install
問題がなければ、pyexpectを利用できます。

パソコンから、iPodにssh認証接続し、UARTに接続された気圧ガジェットから気圧を読み取るプログラムです。

下準備として、ssh認証接続するには、パソコンのcygwinで次の命令を実行します。

ipod2gをcygwinに認識させるためには、/etc/hostsに次の設定をしています。この場合は、iPodのIPアドレスが192.168.0.18の場合は、hostsファイルに次の行を挿入します。
192.168.0.18 ipod2g

cygwinを起動します。
$ cd　　homeへ移動します。
$ ssh root@ipod2g　　パスワードを入力し、iPodに移動します。
root# mkdir .ssh　　.sshディレクトリを作成します。
root# exit　　cygwinに戻ります。
cygwinに戻ってから次の命令を実行し、.sshディレクトリにid_rsa.pubファイルを生成します。
$ ssh-keygen　　ssh-keygenを実行し、すべての質問に対しリターンキーを入力します。次に、.sshディレクトリに移動します。
$ cd .ssh
作成したid_rsa.pubファイルを、iPodの.sshディレクトリにファイルコピーし、そのファイル名をauthorized_keysにします。
$ scp id_rsa.pub root@ipod2g:~/.ssh/authorized_keys
パスワードを入力し、実行を完了させます。

次に、ssh認証接続を試してみます。
$ ssh root@ipod2g　　ssh命令で、パスワードを要求されなければ、インストールは成功です。

iPodにUART接続された気圧センサーを読んで見ます。cygwinを起動して、次の命令を実行します。
$ wget http://web.sfc.keio.ac.jp/~takefuji/getdata.exp
$ chmod 755 getdata.exp
次に、getdata.expを実行し、パソコンからiPodと会話しながら気圧を読み出す様子が表示されます。
$ ./getdata.exp
spawn ssh root@ipod2g
ipod2g:~ root# serial

```
enter command: r
updated
enter command: p
p=101652
enter command: end
ipod2g:~ root# exit
logout
Connection to ipod2g closed
```

$ crontab -e　　crontabを実行し、次の設定をすると、ホームディレクトリに最新の気圧データが1分毎にpdataファイルに書き込まれます。

```
0-59 * * * * /home/Administrator/getdata.exp
```

iPod上で、気圧センサーデータ収集するpget.pyを紹介します。

```
# cat pget.py
#!/usr/bin/env python
import serial
com=serial.Serial('/dev/tty.iap',baudrate=9600,bytesize=8,parity='N',stop
bits=1,timeout=1)
com.write('r¥n')
a=com.readline()
com.write('p¥n')
a=com.readline()
fp=open('/var/root/pdata','a')
fp.write(a)
com.close()
fp.close()
```

crontabを次のように設定すると、1分毎に気圧データをpdataファイルに書き込みます。

```
0-59 * * * * /var/root/pget.py
```

パソコンから、iPod上のpdataファイルを定期的に転送するプログラムgetipodfile
を紹介します。

```
$ cat getipodfile
#!/bin/expect
set timeout 30
spawn sftp root@ipod2g
expect ″ftp″
send ″get pdata¥n″
expect ″ftp″
send ″exit¥n″
interact
```

このように、cron機能やexpect機能を使うことによって、ネットワークを経由して、
自由自在にデータ採取が簡単にできます。