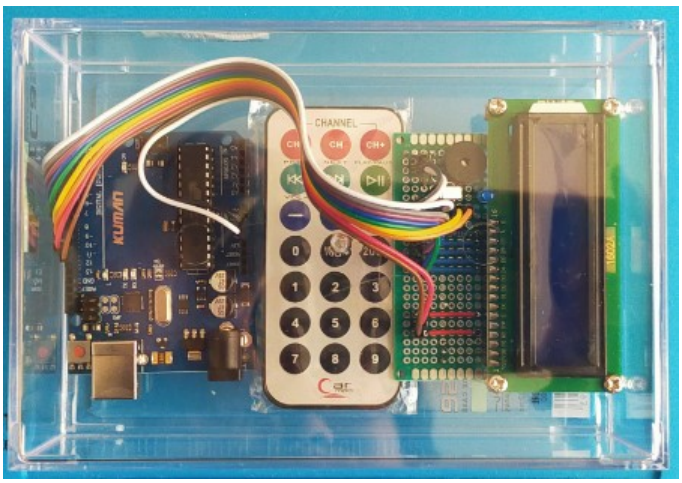
モールス符号聴解練習機

JARL 武蔵野クラブ(JA1YSW)　JJ1SLR　張弘達

目次



1　概要

アマチュア無線では主に，「電話」「電信」で交信を行います。日本の国家資格のアマチュア無線技士 3 級以上は、電信交信のために、欧文モールスの暗記やそれによる通信の技能の習得が求められます。しかし、今日本のアマチュア無線技術者国家試験は筆記試験のみ、簡易化になって、例え 1 級に合格してもモールス符号による通信の実技が身に付けるとは言えません。私はこの実技ができない者の 1 人です。モールス符号受信を自己訓練するため、適当なツールを使って練習するのは一番大切であると思います。

パソコンを使って練習ソフトを利用や、関連ウェブサイトやビデオ動画を聞くなど練習するのはいいですが、専用装置を使って練習するのは、もっとアマチュア無線のセンスを感じるのでしょうと思います。それで「モールス符号聴解練習機」を自作しました。

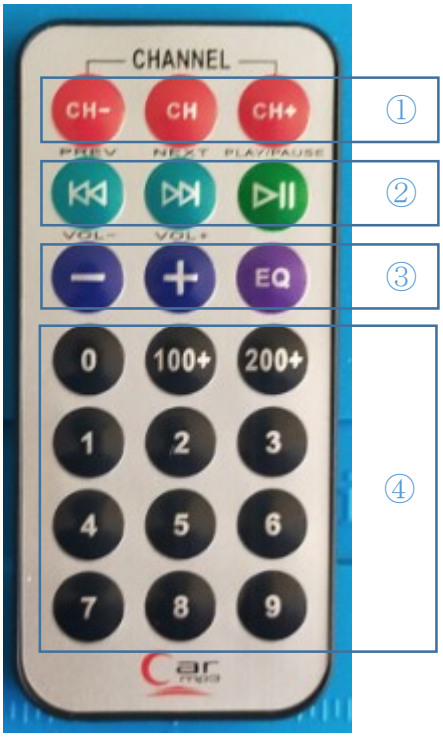
今回作成した練習機のソースコードは、下記の GitHub サイトでオープンソース（GPL3.0 ライセンス）で掲載しました。誰でも自由に入手・利用可能です。

https://github.com/JJ1SLR/MorseListening

2　機能説明

2.1　リモコン制御

練習機は、ハードウェア構成の簡素化のため、赤外線リモコンを利用して、本機を制御します。もともと音楽プレーヤーの汎用品リモコンですから、各キーの機能を再定義します。そして、以下説明するキー以外は無効となります。



① 順序練習文字間隔調整（順序練習モードのみ有効）

- 「CH-」　文字間隔を増加（遅くなる）
- 「CH+」　文字間隔を減少（速くなる）
- 「CH」　文字間隔をデフォルトに戻す

② 速度（WPM）調整

- 「PREV」　速度減少
- 「NEXT」　速度増加
- 「PLAY/PAUSE」　速度をデフォルトに戻す

③ 音調（音の周波数）調整

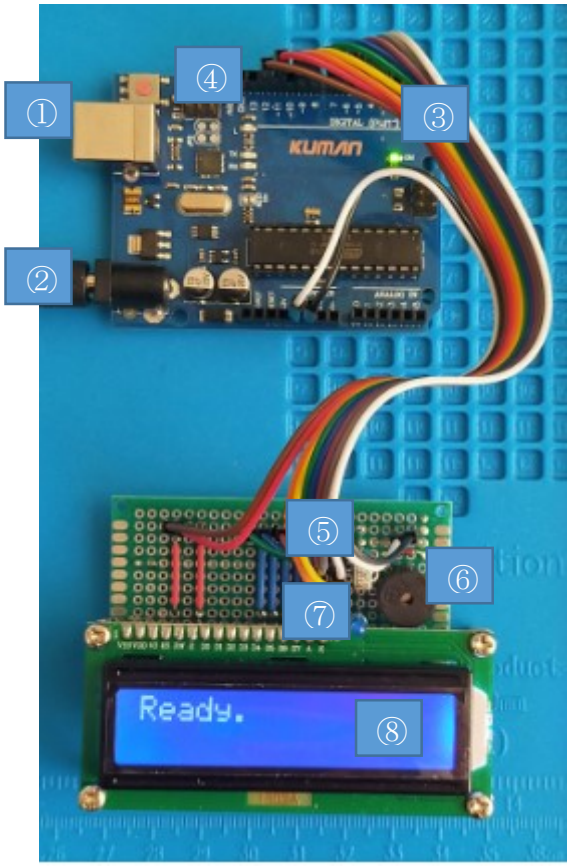
- 「VOL-」　音調を低める
- 「VOL+」　音調を高める
- 「EQ」　音調をデフォルトに戻す

④ 動作モード選択

- 「0」　初期モード
- 「1」　順序練習モード
- 「2」　ランダム練習モード１（符号全体）
- 「3」　ランダム練習モード２（英数字のみ）
- 「4」　ランダム練習モード３（英字のみ）
- 「9」　作者 CQ を出すモード

2.2　本体各部

本体は、以下の各部分で構成します。



① USB ポート（Type-B）：PC を接続してファームウェアを更新できます。また、+5V 安定化電源でこのポートから給電できます。

② 電源ポート：7V～12V DC の電源で給電します。

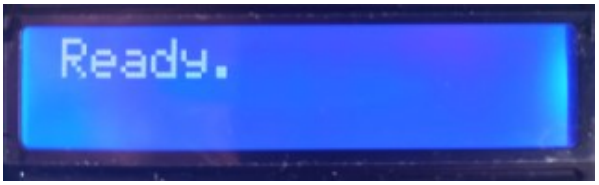
③ 電源インジケータ：給電正常の場合、点灯します。

④ リモコン受信インジケータ：リモコン受信の場合、点滅します。
⑤ リモコン受信モジュール：リモコンの制御信号を受信します。
⑥ ブザー：モールス符号の音を出力します。
⑦ モールスインジケータ：モールス符号の光を出力します。
⑧ LCD ディスプレイ：モールス符号、その他の情報を表示します。

2.3　動作モード詳細

2.3.1　初期モード

練習機を電源に接続して、初期モードになります。また、リモコンの「0」キーを押下の場合、初期モードになります。下記の「Ready.」画面を表示して、ユーザ入力を待機します。



2.3.2　順序練習モード

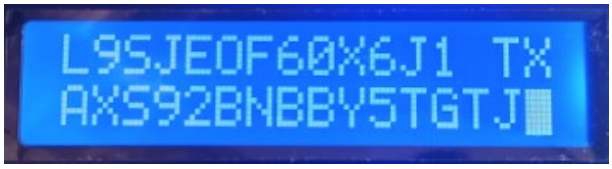
リモコンの「1」キーを押下の場合、順序練習モードになります。
「無線局運用規則　別表第一号　モールス符号（第十二条関係）」「2　欧文」
(https://www.tele.soumu.go.jp/horei/reiki_honbun/72393000001.html)
（以下、「モールス符号表」と略します）
表のモールス符号の順番通りモールス符号の音を出力し、LCD ディスプレイで現在練習中のモールス符号の文字と「.」「-」で示す符号を 2 行表示します。最後の文字出力後実行を停止します。

2.3.3　ランダム練習モード１（符号全体）

リモコンの「2」キーを押下の場合、ランダム練習モード１（符号全体）になります。
モールス符号表をランダムで出力し、32 文字（空白文字も出力する場合があります）のモールス符号を出力します。最後の文字出力後実行を停止します。停止後、LCD ディスプレイの表示はクリアされないから、筆記したの場合、筆記文字と LCD ディスプレイで表示する文字と照合ができます。
順序練習モードと違い、「.」「-」で示す符号は出力しません。そして、強化練習のため、音は先に出力してから LCD ディスプレイで文字を表示します。

2.3.4　ランダム練習モード２（英数字のみ）

リモコンの「3」キーを押下の場合、ランダム練習モード２（英数字のみ）になります。「ランダム練習モード１（符号全体）」と同じ動作で、出力文字の範囲は英数字のみです。



2.3.5　ランダム練習モード３（英字のみ）

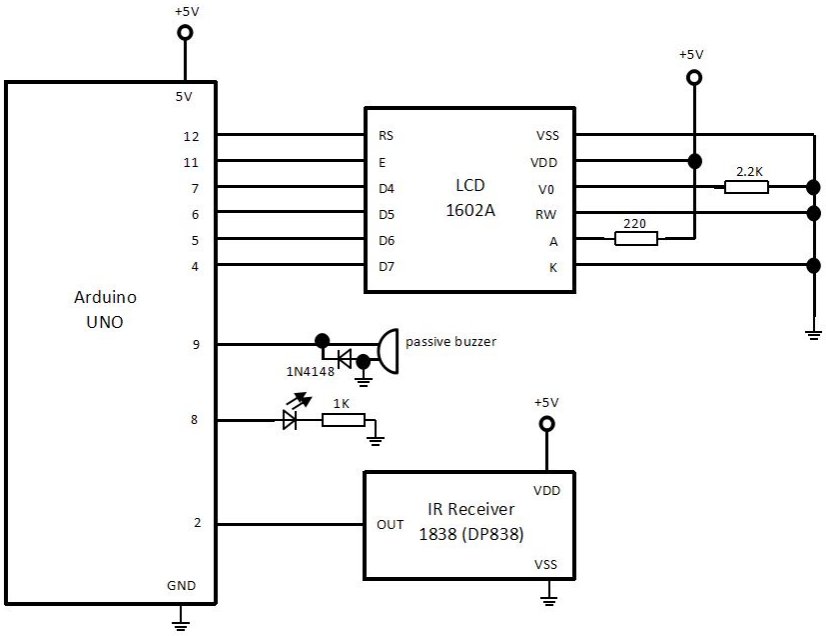
リモコンの「4」キーを押下の場合、ランダム練習モード３（英字のみ）になります。「ランダム練習モード１（符号全体）」と同じ動作で、出力文字の範囲は英字のみです。

2.3.6　作者 CQ を出すモード

リモコンの「9」キーを押下の場合、作者 CQ を出すモードになります。
作者 JJ1SLR の CQ を出す時のモールス符号「CQ CQ CQ DE JJ1SLR JJ1SLR K」を出力します。最後の文字出力後実行を停止します。ランダム練習モード１～３と違い、このモードでは音の出力と LCD ディスプレイ文字の表示は同時に行います。

3　ハードウェア構成

練習機のハードウェア構成はシンプルで、回路図は以下です。

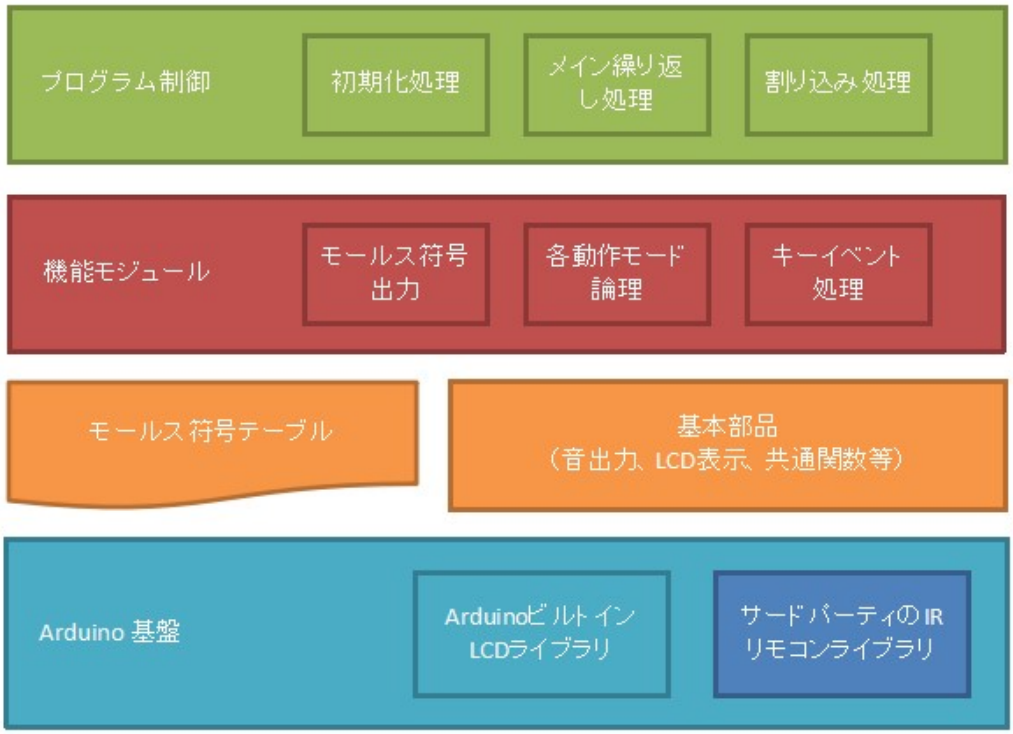


- 各抵抗器は概略計算の上、回路をテストしながら値を選定しました。
- Arduino のピン接続はソースコード上の定義と一致しなければなりません。
- 音調を調整可能にするため、パッシブブザーを利用し、ソフトウェアで矩形波を発生して作動します。矩形波の周波数をソフトウェアで制御します。
- 今回のパッシブブザーはコイル式の部品なので、逆起電力から IC を保護するため、ダイオード 1N4148 を追加しました。

4　ソフトウェア構成

4.1　ソフトウェア構成概要

ソフトウェア構成は以下の図を参照してください。



ソースコードの所在は「1 概要」を参照してください。紙幅の限りでここでは全部の説明を割愛します。重点的に一部を以下説明します。

4.2　モールス符号テーブル

モールス符号のデータ構造は今回のプログラムに対して一番重要なポイントであると思います。なぜなら、今回開発のプログラムは Arduino Uno の ATmega328P マイコンに組み込む予定で、利用可能な ROM（フラッシュメモリー）は 32KB 未満で、RAM は 2KB のみあります。これを計画的に使わないと、特に RAM がオバーして、プログラム全体は正常に動作ができなくなります。限られた資源の有効活用は組み込み系プログラミングの特徴です。

モールス符号は一見 2 進数で表せるが、「・」「—」以外、空白も存在するので、実質 3 進数であれば適当です。しかし、コンピューターは 2 進数でデータを処理するので、この矛盾をどう解決するのは重要です。

4.2.1　一般的なプログラムの考え方

一般ウェブや PC プログラミングであれば、Key-Value ペアでモールス符号テーブルを格納するのは簡単でできます。

例：

文字	Key	Value
A	‘A’	“.-”
B	‘B’	“-...”
C	‘C’	“-.-.”
...	...	...

これは一番考えやすいデータ構造で、リスト又はハッシュテーブルに保存します。プログラムではKey文字でこのテーブルを検索し、取得するValueの「.」「-」文字列を1文字ずつ処理しモールス符号情報出力等処理が行えます。
とてもシンプルな構造ですが、たとえ文字「B」の場合、Keyは1Byte、Valueは5Byte（最後尾‘¥0’文字含む）で、少なくとも6Btyeが必要です。プログラム実行の時、このテーブル全体はRAMにロードされますから、メモリが足りない恐れがあります。

4.2.2　今回開発の考え方

コンピュータにて文字（英数字、記号）はASCIIコードの形式で処理されます。例えば文字‘A’は、メモリ上は0x41を格納し、入出力の時はデータ型によって文字‘A’と認識されます。この特徴を利用してASCII順で各文字のモールス符号情報を配列に格納すれば、指定文字のインデックスが算出できますから、Key情報の保存は必要なくなりました。
また、今回練習機の処理対象モールス符号は単独文字のみなので、一番長いものは一部の記号で音6つまでです。1Byteは8ビットで、「いくつのビットは有効であるか」の問題が解決できたら文字1つのモールス符号は1Byteで収めることは可能であると思います。
上記の観点で、以下のモールス符号とそのテーブルのデータ構造を設計しました。

■モールス符号

1Byteの「・」は0で、「—」は1で、ビット単位で左側（上位桁）から右側（下位桁）へ順序格納します。そして無効ビットの始めの桁は1を設定し、残りの全桁は0を設定します。
例：
文字‘C’の格納イメージは以下に示します。

ビット	7	6	5	4	3	2	1	0
値	1	0	1	0	1	0	0	0
意味	—	・	—	・	無効ビット開始	無効ビット	無効ビット	無効ビット

このようなデータを処理時に、まず右側ビットから左側へデータを読んで、初めて読んだ1を含めて無効ビットと認識し、無効ビット数を取得できます。そして有効ビット数も算出できます。上記の例の場合、無効ビット数は4bitで、有効ビット数は(8-4)=4Bitであることがわかりました。そして左側ビットから右側へも

う一度 4Bit の情報を読み込んで、「1010」つまり「―・―・」の正しい情報が取得できます。

■モールス符号テーブル

ASCII コード表の 0x20（空白文字）～0x5A（‘Z’）の範囲ではモールス符号表の全文字を含みます。長さは 59 のバイト配列を定義し、各アイテムは上記の「モールス符号データ」を格納すれば、モールス符号テーブルが作成できます。但し、ASCII コード表に含み、モールス符号表に含まない文字に対して下記特別な処理が必要です。

- 空白文字：0x00 で表します。
- その他モールス符号表に含まない文字（例：‘#’ ‘$’ ‘*’等々）：0x80 で表します。

文字を指定して、モールス符号情報を取得の場合、モールス符号テーブルのインデックスを算出します。計算は単に文字の ASCII コード値引く固定のオフセット値（0x20）で済みます。

例：

文字‘C’のインデックス　＝‘C’の ASCII コード(0x43)―オフセット値(0x20) ＝　0x23　＝　35

モールス符号テーブルのインデックス 35 の値を取得すれば、文字‘C’のモールス情報が取得できます。

4.3　割り込み処理

初期化時の定義によって、IR リモコンのキーを押下する時、ハードウェア割り込みが発生し、割り込みハンドラーで処理します。そして IR リモコンの受信情報をデコード結果によって、各キーの押下イベントとして処理し、予め用意の各キーイベント処理関数に振り分け、キー制御機能を実現します。

順序練習文字間隔、速度（WPM）、音調（音の周波数）の設定情報は、常にグローバル変数に退避しているから、これらのキー処理関数では単純に該当グローバルの値を変更して完了します。メイン処理ではこれらのグローバル変数を参照して、適当な動作を行います。

現在実行モード情報もグローバル変数に退避しているから､キー処理関数でもこの変数を設定します。但し、メイン処理でこのモード遷移の事を検出させるため、もう 1 つのモード遷移フラググローバル変数を用意します。モード遷移用キーを押下時、処理関数でこのフラグ変数も合わせて true に設定する必要です。

設計要点として､割り込みハンドラで実行時間が長い処理を避けて､できるだけ早めに割り込み処理を終わらせ、メイン処理に戻すのはポイントです。この観点によって上記の設計にしました。

なお､Arduino ではデフォルトで多重割り込みができませんから、割り込みハンドラーでの多重割り込みの考えは不要です。

4.4　音出力と遅延関数

パッシブブザーを使うため､直流電圧を印加すれば音が出ません。一定の周波数の

矩形波をソフトウェアで発生する必要があります。Arduino 基盤が tone()関数を提供し、この関数を利用してパッシブブザー作動させます。tone()の中身はハードウェアタイマー2を利用して音信号生成するので、この関数を利用の場合、タイマー2は他用途に転用できません。しかし IR リモコンのライブラリでもディフォルトでこのタイマー2を利用していますので、IR リモコンのライブラリを改修し、タイマー1利用のように変更しました。

tone()関数は呼び出した後すぐに戻します。つまり、これはノンブロッキング関数ですから、その後音が鳴らす期間に delay()関数を追加しなければなりません。しかし、この期間中にもリモコンの操作、特に実行モード変更操作の受け入れが求められるので、Arduino 基盤の delay()関数の代わりに自分の delayWithChk()関数を作りました。

```
// Delay milliseconds, non-blocking function.
// It calls eventChecker() function in the delay loop.
void delayWithChk(unsigned int mils) {
  unsigned long startMillis = millis();
  while (millis() - startMillis < mils) {
    eventChecker();
  }
}
```

この関数で開始時間を記録し、繰り返し処理で時間経過をチェックし、タイムアウト後に戻します。そして、繰り返し処理中に常に eventChecker()関数を呼び出して、実行モード遷移フラグを監視することができます。ユーザのキー操作による実行モード遷移の時は素早く対応できるようになりました。

4.5　ロングジャンプ制御

ユーザのキー操作による実行モード遷移の時、現在実行モードの処理を中止し、速やかに指定実行モードの処理の始まりが望ましいです。

eventChecker()関数処理中にこれを検出し、そして戻り値等による判断で、関数コールスタックを順次戻すのはいいですが、色々なところで判断処理が必要で、煩雑になります。だから今回は C 言語の標準ライブラリ関数 setjmp()、longjmp()を利用して一発で戻します。Arduino ではこれらが問題なく動作します。

Arduino のメイン繰り返し処理の入口で setjmp()を呼び出します。

```
// The loop() function of the Arduino framework.
void loop() {
  setjmp(g_jmpBuf);
  switch (g_runStatus) {
  …
  }
  // wait for remote control input
  for (;;) {
    delayWithChk(1);
  }
}
```

ここで、g_jmpBuf はコンテキスト退避グローバル変数です。

そして、eventChecker()関数は下記のように実装します。

```
// This function is called by delayWithChk() function.
// When run status changed, it will do a long jump to the
// start of loop() function.
void eventChecker() {
  if (g_bRunStatusChanged) {
    DBG_MSG("eventChecker: status changed");
    g_bRunStatusChanged = false;
    longjmp(g_jmpBuf, 1);
  }
}
```

ここで longjmp()関数を呼び出したら、setjmp 関数のところにジャンプして、代わりに setjmp()関数から戻します。これはかなり変わった制御フローですがかなり便利です。

ちなみに今回の開発は動的なメモリ割り当てを使わないので、このロングジャンプ処理によってメモリ解放漏れ、いわゆるメモリリークは特に起こしません。

5　後書き

Arduino は元々プログラミング教育用の開発ボードですが、使い勝手のいいから産業用としても色々な場面で活用されています。多様なセンサーと組合わせて利用できますので、創造力を発揮し、個人的なちょっとした電子工作が最適です。そして、プログラムの書き込みも USB ケーブル 1 本で、特別な設備が要りません。個人的な感想なら、一番メリットはやはり安価ですね。今回はアマゾンから Arduino Uno 互換品開発セットを 3000 円頃で調達しました。Arduino Uno 互換品単品なら 1000 円頃でも入手できます。ちなみに Arduino はオープンソースなものなので、非正規品（汎用品、互換品）でも Arduino と表記しない限り違法ではありません。だから今回は Arduino Uno を選択してモールス符号聴解練習機を作りました。

今後時間があればこのモールス符号聴解練習機の機能をもっと充実し、例えば Q 略語・その他略語の強化練習機能や、和文モールス符号聴解練習機能や、パソコンで EEPROM に文章を書き込んでそれをモールス符号出力練習機能等のバージョンアップを考えています。皆様引き続きのご応援よろしくお願いいたします。

以上