CHC勉強会資料

高松オリーブハムクラブ作製 Remote Keyer 改造

JG1LMK/角田 護

2026.8.23

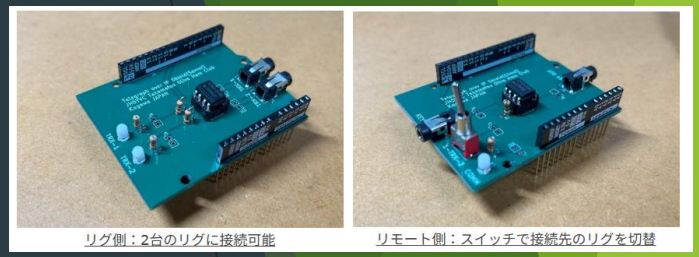
本日の発表内容

1. 高松オリーブハムクラブJH5YVC 作製 Remote Keyer について　（3分）
2. Client ボード 改造の経緯（2分）
3. ソフトウェア改造　　（5分）
4. Client PCB 基板改造　　(5分）
5. PCB 製造　(5分)
6. BUG 発覚（5分)
7. Demo　　（5分）
8. まとめ　　(2分）

1. 高松オリーブハムクラブ Remote Keyer

・**頒布情報**　https://takamatsu-olive-ham.club/distribution.php

頒布価格：キット一式 4000円（サーバ、クライアントボード分）
但し、別途 Arduino UNO R3 及び イーサネットシールドが 2個ずつ必要
　Arduino UNO R3 4710円 x 2台 = 9420円
　イーサーネットシールド2 4980円 x 2台 = 9960円



・**開発環境**
Arduino UNO プログラム開発には Arduino IDE (無料) 使用可能

・**設計データ**
・「リモートキーヤーキット for ArduinoUNO_組立手順書Ver1.0」
↑ サーバ、クライアント PCB の回路図が記載されている

・「ソースコード （toip_shield_jh5yvc.zip）」がWeb 上で公開されており、
“*アマチュア無線家による利用は改変、再配布を含め自由とします。” とコメントあり

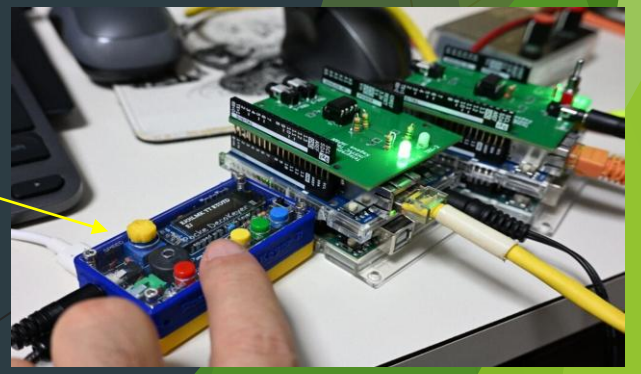
・PCB のガーバーデータ等は非公開

2. Client ボード 改造の経緯

・**エレキー機能が無かった**
キットの在庫があるうちに購入しようと思い焦って購入申し込みを行った。
→ 届いて直ぐに組上げてパドルを繋いだが、うまくいかず。
パドルを使う場合はエレキーが必要であることを、この時点で気づきました。
→ ハムフェアの AKC ブースで JA6IRK局さん作製の PockeDecoKeyer II にエレキー機能がある
ことに気付き、使用したところ無事うまく動作した。
→ その後、直ぐに Pocke Keyer II の電源に不具合が出て、電池を交換。
→ Pocke Keyer II は一度は復活したが、直ぐに壊れて今度は全く動作しなくなった。
→ 結局 Remote Keyer のセットは 暫く放置状態となっていた

⇒ 6/28 の CHC オンライン勉強会で JK1DVU/庄司さんの発表を聞いてキットの存在を再認識。
そもそも Aruduino UNO R3 上で動作させているのだから、エレキー機能も内蔵させて
しまえば良いのでは？？　と思い立ち、改造を決意。

PockeDecoKeyer II：
2023年のハムフェアで
AKCブースで 6K円 で購入



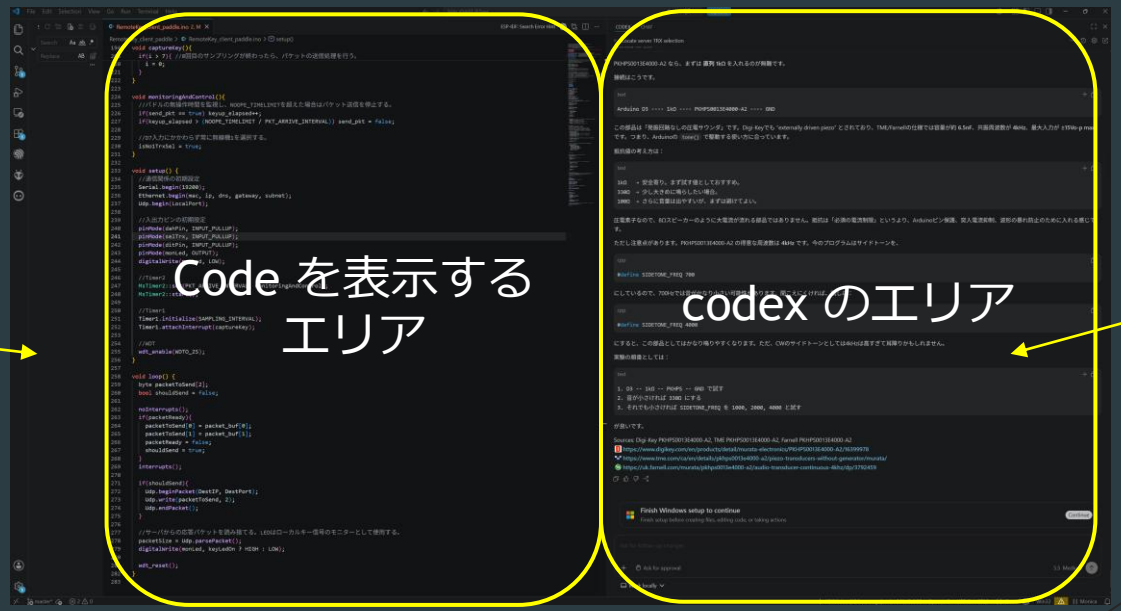
3.ソフトウェア改造 (1/2)

・ソフトウェア開発環境

勿論、Aruduino IDE を使って、公開されているソースコードをゴリゴリと書き換えて自力でエレキーの機能を追加する方法も可能であるが、今日の AI 時代、人力でソフトウェア開発するのは **時代遅れ**。

当局は、最近は、論理回路設計でも、プログラム開発でも、Visual Studio Code に “codex” をインストールして、AI にお願いする形でこの手の開発を行うようにしています。
これにより、仕事の効率は 10倍以上は Up した感があります。

Visual Studio Code 画面



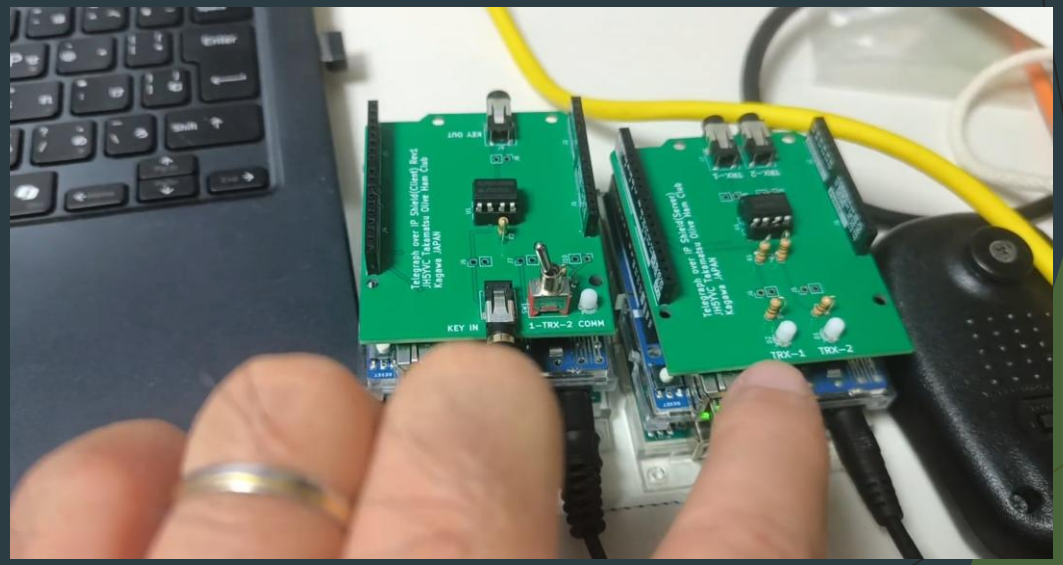
こちらのエリアで codex と Chat しながら開発を進める

3.ソフトウェア改造 (2/2)

・codex にお願いした結果

オリジナルの Remote Keyer の software を codex に喰わせ、解析してもらい、「これにエレキー機能を追加して下さい」とお願いし作業を開始。
ある程度出来たところで Arduino IDE でプログラムを実機にダウンロードしてテストを行い、デバッグすることを 小一時間 繰り返した結果、6/28 CHC の勉強会当日中に、エレキー機能が動作するようになりました。

6/28 晩に当局のブログに
エレキー機能が加えられた
ことを記事に書きました



証拠の動画をYouTube にもアップ
https://www.youtube.com/watch?v=6YDtzmUP3sE

4. Client PCB 基板改造 (1/3)

・**PCB 開発環境**

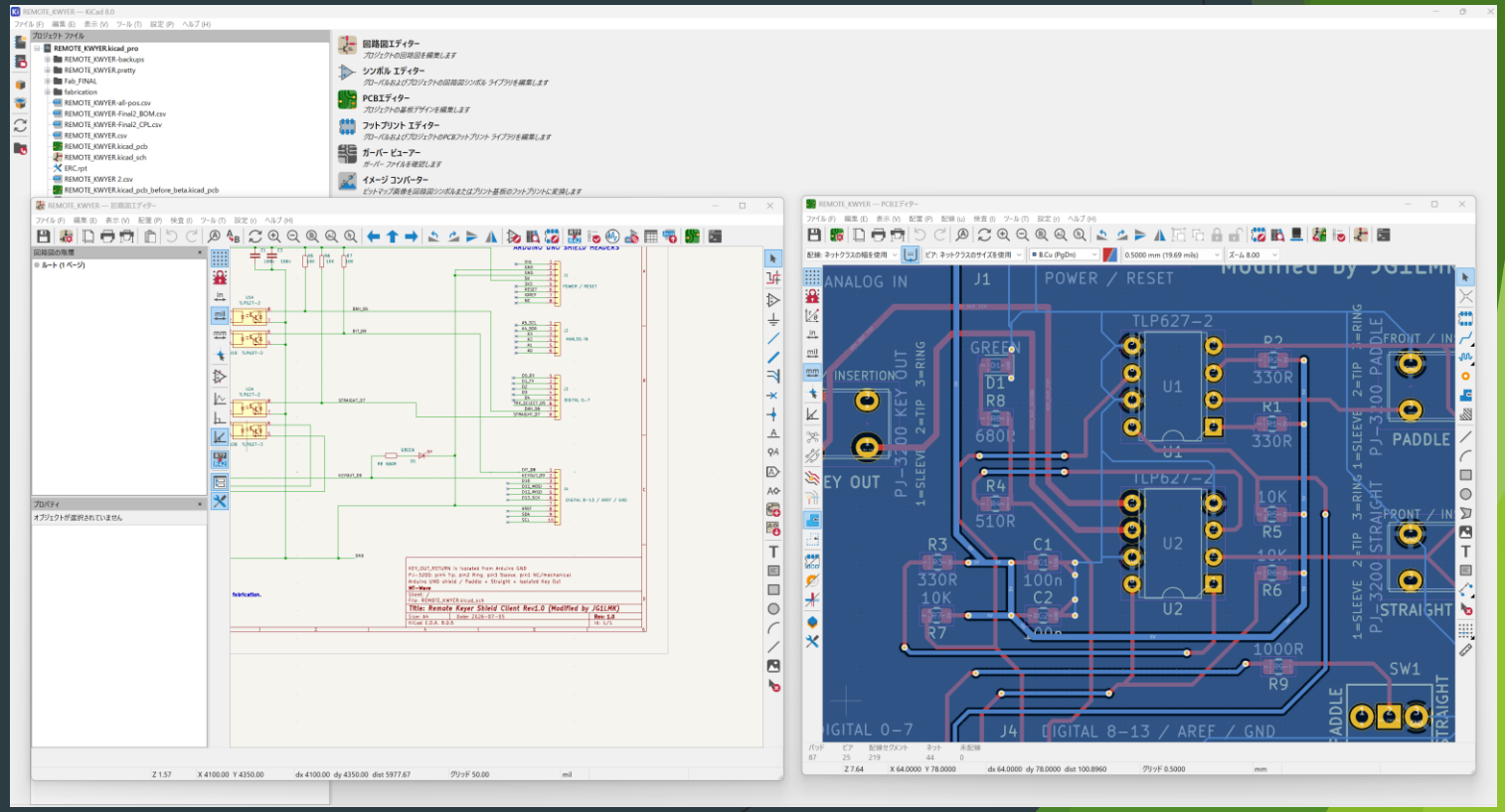
フリーソフトウェアである KiCad (Version: 8.0.6) を使用。
（ **最新版は10.0.5**（2026年7月22日リリース））

このソフトは、回路図エディターや、シンボルエディター、PCBエディター、
フットプリントエディター、ガーバービューア　などが融合した、超強力な開発環境です。

KiCad のイメージ →

使用するのはほぼ
初めてでした。

当然、使い方が
分からない場合は、
全部 ChatGPT へ
質問し、コマンドの
使い方などを教えてもらい
その通りに入力。



4. Client PCB 基板改造 (2/3)

・ChatGTP にお願いした結果

ソフトウェアが codex で出来たことに気を良くした当局は、完全に頭に乗って、オリジナルのボードの回路図面を ChatGTP に喰わせて、「パドル対応が可能な回路図を作成して下さい」とお願いしてみました。

すると、最初はバグだらけであったものの、数回のやり直しで動きそうな感じの回路図が完成。

そこで次は、ガーバーデータは無いので、携帯電話で撮影したクライアント基板写真を ChatGTP に喰わせて、「回路図に合ったPCBレイアウトを作成して下さい」と無理難題をお願いしてみました。

すると、暫くすると、KiCad で読込可能なレイアウトデータを作成してきました！！

4. Client PCB 基板改造 (3/3)

・ChatGPT の PCB レイアウトは使えませんでした！！

最初は ChatGPT が PCB レイアウトデータまで出力し、非常に驚いたのですが、その後 デザインルールチェッカーを通してみるとエラーの雨嵐。

パーツのフットプリントは間違っているし、ピン番号も滅茶苦茶だし、配線もショートだらけで回路図と全く一致しておらず。そもそも PCB 基板の大きさも間違っていることに気付きました。

結局、ほぼ初めから作り直すことに。

→ お蔭で、KiCad の PCB エディターの使い方については、かなりマスターできました。
　もしかすると、ChatGPT が愛をもってわざと試練を与えてくれたのかもしれません…

PCB の製造を発注できるデータが仕上がったのは 7/11日となりました。
PCB 回路図作成とレイアウトデータ作成・検証に約2週間を費やすこととなりました。

5. PCB 製造 (1/2)

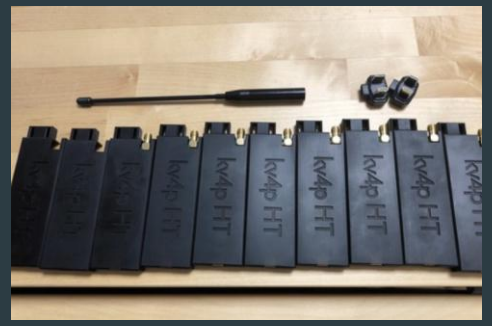
・PCB 製造メーカー選定

以前、 kv4pHT という携帯電話を 144MHz FM の無線機にしてしまう機器を作製する際、中国の JLCPCB というメーカーを使ったのですが、その際、他の会社からパーツを購入したところ、マイコンモジュールに使用されていたフラッシュが粗悪品で、その無線機を作るプロジェクトがパーッになるという苦い経験をしました。

反省し、今回は ChatGPT が最も推奨する国内メーカー（MARUTSU）に依頼しようとしたのですが…

・価格　（日本は中国の2~3倍程度？）
・TAT　（日本は中国の2~3倍程度？？？）
・データ受け入れの容易さ
　　（JLCPCB はガーバーデータを自動で読込中身を解析し、結果を表示）

の点で、圧倒的に 中国メーカーの方が優れていたため、結局 JLCPCB を使うこととしました。



←kv4pHT
この時は PCB だけでなく
ケースも作成しました。

5. PCB 製造 (2/2)

・JLCPCB

こちらの会社では、PCB基板を発注するにあたり、部品実装がある場合には、部品選択の部門である PCBA に部品のチェックを依頼しなければならない。これが終了すると、PCB 製造の発注が可能となるという2段階のステップ踏むこととなる。

1. PCBA (部品選択) (7/11~7/18 (7日間))

今回、こちらの部門の作業がなかなか進まず。何度も何度もクレームを入れてやっと進むことができた。クレーム受付はチャット越しのオペレータが対応。彼らは最後には「関連部署にリクエストしておきます」と回答するだけ。日本語でも書けるが、相手によっては英語の方が早く伝わりそうな感触あり。部品選択は別料金で発注し、チェックが終了するまでに1週間～12日程度かかった。

2. PCB 製造＋部品実装＋梱包＋発送 (7/18~7/23 (5日間))

部品選択部門によるチェックが終わると、PCB製造の発注が可能となり、値段も確定する。

この発注を行ってから、PCB基板が出来実装が終わるまでが約2日。発送されて日本に入り、手元に届くまでが更に約2日。

1, 2 を合わせて 12 日間で PCB基板を入手できた！！

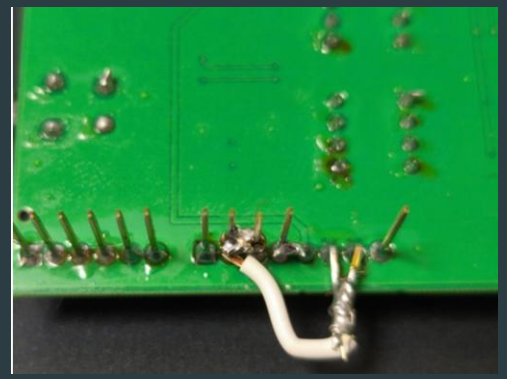
6. BUG 発覚 (その１)

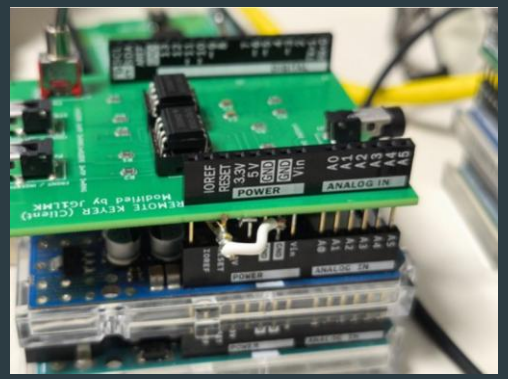
・PCB基板レイアウトBUG

7/23 に届いたボードにパーツを半田付けして組み上げ、ボードに火をいれてみると全く動作せず。デバッグを行った結果、ChatGPT が一番最初に用意した Arduino のシールド間を接続するコネクタ端子の信号の並びが逆転していることが判明。

これにより、RESET 端子が GND とショートしてしまい、Arduino が全く起動しない不具合が起きていることが判明。

この端子の信号の並びぐらいは正しいだろうと、チェックをして無かったことが原因。
とりあえず、下記のようなジャンパー信号配線で BUG を回避し、それ以外では問題がなくエレキー機能が正しく動作することを確認。





コネクタの信号並びを修正した PCB基板データを再作成し、再度 JPCPCB へ 7/25 に発注をかけ、8/11 に修正ボードが届いた。2回目は 17日間 かかった。早速火入れをして、今度は正しく動作しだした様に見えたのですが...

6. BUG 発覚 (その２)

本 Remote Keyer のセットの Client ボードには Side Tone 発生機能がないため、Key Out 端子に無線機か手持ちの Side Tone 発生機かを接続しなれば、 Clientマシーン単体では LED が光るだけなので、トーンによる確認が出来ずほぼ使えません。

今までの実験では、Server マシーン側の Key Out 出力から FT-818ND の Key 入力へ接続し、Side Tone 発生機代わりとして使っていたので、Client 側の Key Out 出力をまだ試していなかったことに気付きました。

早速繋いで実験してみたところ、音が出ず！！　こちらにも BUG があることが発覚しました。
先のバグ入りボードでもっと入念に調べておくべきでした。(トホホ…)

結局ワークアラウンドとしてプログラム変更により、空いている D3 端子にエレキー機能の出力結果に従った 0V/5V の 700Hz 方形波を発生させ出力。圧電サウンダ(他励振タイプ)を D3-GND 間に接続することで、何とか音を出すことが出来ました。

結果が違うのは何故？？

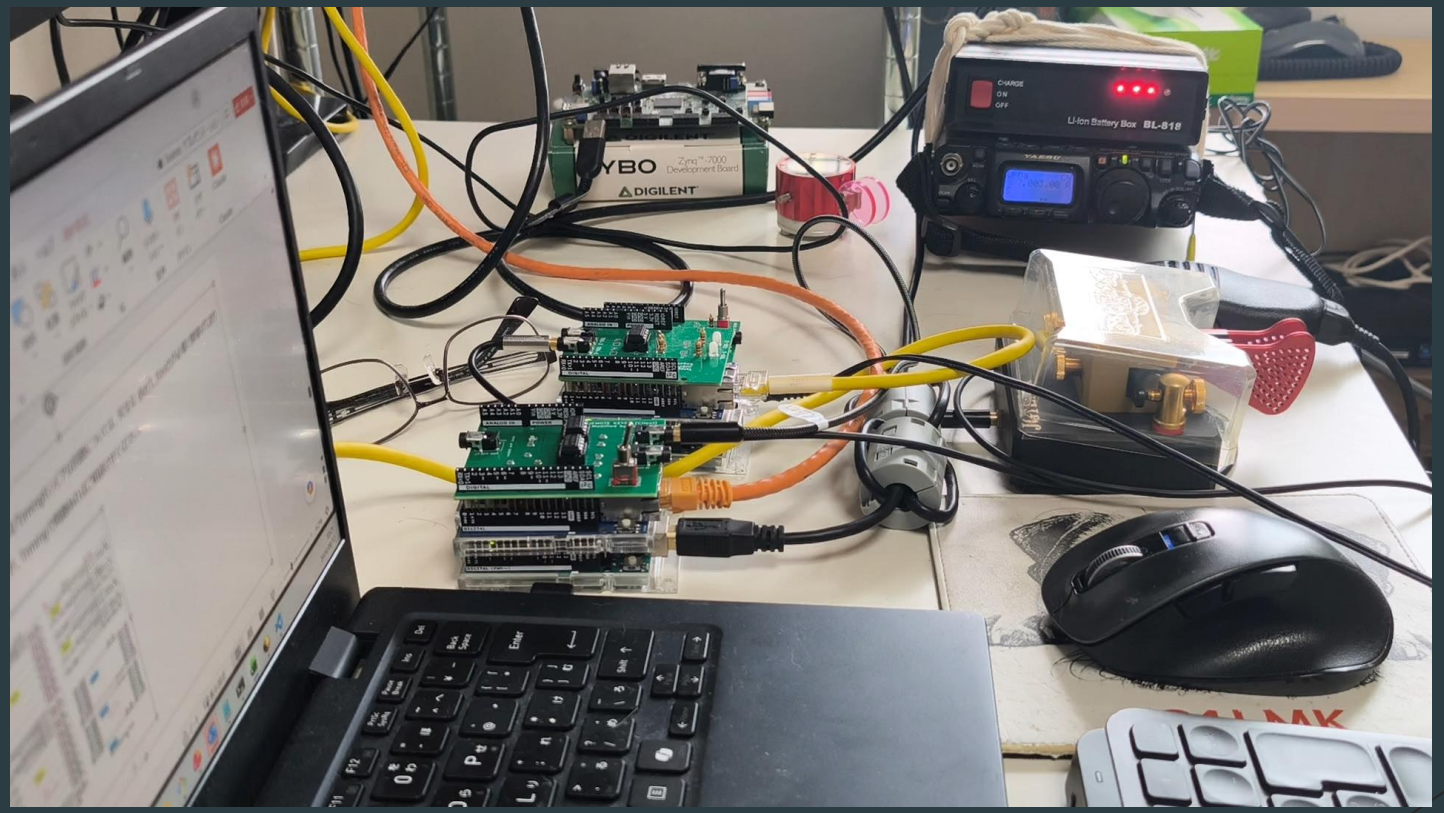
無線機vs練習機	パドル	本製品
FT-818ND	○	○
GHD GR301A	○	×

共立電子産業
圧電サウンダ

7. DEMO

・動画を YouTube に Upload しました。

https://youtu.be/mrWdAQO0ZXg



8. まとめ

今回の 高松オリーブハムクラブ様の頒布品の Remote Keyer の改造を通して、

・オリジナルの Remote Keyer の仕組みを理解出来た
・エレキー機能の実装の一手法を理解出来た
・KiCad での PCB 基板の開発フローと手順について習得出来た
・AI を使ったプログラミングのコツを掴むことが出来た
・全項目を検証をすることの大切さが分かりました

といった貴重な体験をすることが出来ました。

今回の経験が出来たのも、こちらのキットが Arduino を使った汎用性が高い開発教材となっており、ソースコードがオープンで、尚且つ回路図も入手できたこと。そして、ボードの回路もソフトウェアも比較的シンプルで分かり易かったため、当局のようなド素人でも AI の助けをかりて、こんな実験が出来たのだと思います。大変素晴らしい教材をご提供頂いた高松オリーブハムクラブ様には心から感謝申し上げます。

今後、益々 AI は急速に進化し、最新の技術を使いこなせる人だけが仕事を得られ、それ以外の人は仕事にも就けない時代が到来すると思います。

生き残りをかけて今後も精進していかねばと強く思いました。

ご清聴頂き、
ありがとうございました

Appendix

高松オリーブハムクラブ作製 Remote Keyer Original vs Modified Source Code (Client) (1/7)

```
/*=============================================================================*
*リモートキーヤー クライアント用ファームウェア Ver2.1
*Copyright (C) 2019-2025 NAGAO, Toshitsugu(JF5SIM)
*アマチュア無線家による利用は改変、再配布を含め自由とします。
*本プログラムを運用して生じたいかなる結果に対しても、著作権者は一切の責任を負いません。
*=============================================================================*/


#include <avr/wdt.h>
#include <MsTimer2.h>
#include <TimerOne.h>
#include <SPI.h>
#include <Ethernet.h>

#define SAMPLING_INTERVAL 5000    //キー接点のサンプリング周期　5000ナノ秒 = 5ミリ秒ごとにサンプリング
#define PKT_ARRIVE_INTERVAL  200  //サーバからの応答パケットが返ってくる間隔（ミリ秒）
#define NOOPE_TIMELIMIT 2000      //無操作時間（ミリ秒） この時間を超過してキーイングが行われないとパケットの送信を停止する。




/*=============================================================*
*ネットワーク関連設定　ここから
*お持ちのイーサネットシールドやネットワーク環境に合わせて設定してください。
*=============================================================*/
//イーサネットシールドのMACアドレスの登録
byte mac[] = {0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX};

//自局のNW設定
IPAddress ip(192, 168, 0, XX);
IPAddress dns(192, 168, 0, XXX);
IPAddress gateway(192, 168, 0, XXX);
IPAddress subnet(255, 255, 255, 0);
unsigned int LocalPort = XXXX;

//通信先サーバの指定
IPAddress DestIP(192, 168, 0, X);
unsigned int DestPort =XXXX; //必要がない限り変更しないでください。
/*=============================================================*
*ネットワーク関連設定　ここまで
*=============================================================*/
```

```
/*=============================================================================*
*リモートキーヤー クライアント用ファームウェア パドル直結版 Ver2.1p
*Copyright (C) 2019-2025 NAGAO, Toshitsugu(JF5SIM)
*Paddle keyer extension added for direct paddle operation.
*アマチュア無線家による利用は改変、再配布を含め自由とします。
*本プログラムを運用して生じたいかなる結果に対しても、著作権者は一切の責任を負いません。
** 2026.06.28 Modified by JG1LMK/M.Tsunoda powered by CODEX using OpenAI
**          Keyer Function Added
*=============================================================================*/

#include <avr/wdt.h>
#include <MsTimer2.h>
#include <TimerOne.h>
#include <SPI.h>
#include <Ethernet.h>

#define SAMPLING_INTERVAL 5000    //キー出力波形のサンプリング周期 5000マイクロ秒 = 5ミリ秒
#define PKT_ARRIVE_INTERVAL  200  //サーバからの応答パケットが返ってくる間隔（ミリ秒）
#define NOOPE_TIMELIMIT 2000      //無操作時間（ミリ秒） この時間を超過してキーイングが行われないとパケットの送信を停止する。

#define KEYER_WPM 15              //内蔵エレキーの速度。短点長は 1200 / WPM ミリ秒。
#define SIDETONE_SELF_TEST true
#define SIDETONE_FREQ 700         //ローカルサイドトーンの周波数(Hz)。
#define SWAP_PADDLE_INPUTS false  //trueにすると短点/長点入力を入れ替えます。
#define KEYER_IAMBIC_B true       //true:Iambic B / false:Iambic A
#define PADDLE_DEBOUNCE_TICKS 2   //5ミリ秒サンプルがこの回数連続したらパドル状態を確定します。

#define SEL_NO1_TRX 0x01
#define SEL_NO2_TRX 0x00


/*=============================================================*
*ネットワーク関連設定　ここから
*お持ちのイーサネットシールドやネットワーク環境に合わせて設定してください。
*=============================================================*/
//イーサネットシールドのMACアドレスの登録
byte mac[] = {0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX};

//自局のNW設定
IPAddress ip(192, 168, 0, XX);
IPAddress dns(192, 168, 0, XXX);
IPAddress gateway(192, 168, 0, XXX);
IPAddress subnet(255, 255, 255, 0);
unsigned int LocalPort = XXXX;

//通信先サーバの指定
IPAddress DestIP(192, 168, 0, X);
unsigned int DestPort = XXXX; //必要がない限り変更しないでください。
/*=============================================================*
*ネットワーク関連設定　ここまで
*=============================================================*/
```

高松オリーブハムクラブ作製 Remote Keyer
Original vs Modified Source Code (Client) (2/7)

```
/*=====================================================================*
*使用するピンの設定　ここから
*本プログラムではArduino UnoのデジタルI/Oピンの7番から9番を使用します。
*必要に応じて変更してください。
*=====================================================================*/
const int selTrx = 7; //無線機選択スイッチの入力ピン。LOWの場合無線機1、HIGHの場合無線機2が選択されているとして扱います。
const int keyPin = 8; //キーの入力ピン HIGH:キーダウン LOW:キーアップ として扱います。
const int monLed = 9; //サーバとの通信状態を表示するLEDの出力ピン。サーバからの応答パケットを受信するとHIGHになります。
/*=====================================================================*
*使用するピンの設定　ここまで
*=====================================================================*/

//無線機の選択定義
const int selNo1Trx = LOW;

//キーのUP/DOWNの定義
const int key_down = LOW;
const int key_up = HIGH;

EthernetUDP Udp;




bool send_pkt = true;
bool isNo1TrxSel =true;




int i = 0, j = 0;
int keyup_elapsed = 0;
int packetSize = 0;
byte key_buf1[2] = {0, 0};
byte key_buf = 0;
```

```
/*=====================================================================*
*使用するピンの設定　ここから
*本プログラムではArduino UnoのデジタルI/Oピンの6番から9番を使用します。
*パドルはGNDに落とす接点として、各入力はINPUT_PULLUPで使用します。
*=====================================================================*/
const int dahPin = 6;  //長点パドル入力 LOW:接点ON HIGH:接点OFF
const int selTrx = 7;  //無線機選択スイッチ入力 LOW:無線機1 HIGH:無線機2
const int ditPin = 8;  //短点パドル入力 LOW:接点ON HIGH:接点OFF
const int monLed = 9;  //ローカルキー信号のモニターLED。エレキー出力がキーDOWN中にHIGH。
/*=====================================================================*
*使用するピンの設定　ここまで
*=====================================================================*/

//無線機の選択定義
const int selNo1Trx = LOW;

///パドル接点の定義
const int paddle_on = LOW;

EthernetUDP Udp;

enum KeyerState {
  KEYER_IDLE,
  KEYER_DIT_DOWN,
  KEYER_DIT_SPACE,
  KEYER_DAH_DOWN,
  KEYER_DAH_SPACE
};

volatile bool send_pkt = true;
volatile bool isNo1TrxSel = true;
volatile bool keyLedOn = false;
bool ditMemory = false;
bool dahMemory = false;
bool lastElementWasDit = false;
bool ditPressedStable = false;
bool dahPressedStable = false;
byte ditOnSamples = 0;
byte ditOffSamples = 0;
byte dahOnSamples = 0;
byte dahOffSamples = 0;
int i = 0;
volatile int keyup_elapsed = 0;
int packetSize = 0;
byte key_buf1[2] = {0, 0};
volatile bool packetReady = false;
volatile byte packet_buf[2] = {0, 0};
KeyerState keyerState = KEYER_IDLE;
int keyerTicksRemaining = 0;

const int keyerTickMs = SAMPLING_INTERVAL / 1000;
const int ditTicks = (1200 / KEYER_WPM + keyerTickMs - 1) / keyerTickMs;
const int dahTicks = ditTicks * 3;
```

高松オリーブハムクラブ作製 Remote Keyer Original vs Modified Source Code (Client) (3/7)

Original

```
void startDit(){
  keyerState = KEYER_DIT_DOWN;
  keyerTicksRemaining = ditTicks;
  ditMemory = false;
  lastElementWasDit = true;
}

void startDah(){
  keyerState = KEYER_DAH_DOWN;
  keyerTicksRemaining = dahTicks;
  dahMemory = false;
  lastElementWasDit = false;
}

void startNextElement(){
  if(ditMemory && dahMemory){
    if(lastElementWasDit){
      startDah();
    }else{
      startDit();
    }
  }else if(ditMemory){
    startDit();
  }else if(dahMemory){
    startDah();
  }
}

void debouncePaddle(bool rawPressed, bool &stablePressed, byte &onSamples, byte &offSamples){
  if(rawPressed){
    if(onSamples < PADDLE_DEBOUNCE_TICKS) onSamples++;
    offSamples = 0;
    if(onSamples >= PADDLE_DEBOUNCE_TICKS) stablePressed = true;
  }else{
    if(offSamples < PADDLE_DEBOUNCE_TICKS) offSamples++;
    onSamples = 0;
    if(offSamples >= PADDLE_DEBOUNCE_TICKS) stablePressed = false;
  }
}
```

高松オリーブハムクラブ作製 Remote Keyer Original vs Modified Source Code (Client) (4/7)

Original

```
bool updateKeyer(){
  bool ditInput = (digitalRead(ditPin) == paddle_on);
  bool dahInput = (digitalRead(dahPin) == paddle_on);

  debouncePaddle(ditInput, ditPressedStable, ditOnSamples, ditOffSamples);
  debouncePaddle(dahInput, dahPressedStable, dahOnSamples, dahOffSamples);

  bool ditPressed = SWAP_PADDLE_INPUTS ? dahPressedStable : ditPressedStable;
  bool dahPressed = SWAP_PADDLE_INPUTS ? ditPressedStable : dahPressedStable;

  if(keyerState == KEYER_IDLE){
    if(ditPressed) ditMemory = true;
    if(dahPressed) dahMemory = true;
  }else if(keyerState == KEYER_DIT_DOWN || keyerState == KEYER_DIT_SPACE){
    if(dahPressed) dahMemory = true;
    if(!ditPressed) ditMemory = false;
  }else if(keyerState == KEYER_DAH_DOWN || keyerState == KEYER_DAH_SPACE){
    if(ditPressed) ditMemory = true;
    if(!dahPressed) dahMemory = false;
  }

  if(!KEYER_IAMBIC_B && (keyerState == KEYER_DIT_SPACE || keyerState == KEYER_DAH_SPACE)){
    if(!ditPressed) ditMemory = false;
    if(!dahPressed) dahMemory = false;
  }

  if(keyerState == KEYER_IDLE){
    startNextElement();
  }

  bool keyDown = (keyerState == KEYER_DIT_DOWN || keyerState == KEYER_DAH_DOWN);

  if(keyerState != KEYER_IDLE){
    keyerTicksRemaining--;
    if(keyerTicksRemaining <= 0){
      if(keyerState == KEYER_DIT_DOWN){
        keyerState = KEYER_DIT_SPACE;
        keyerTicksRemaining = ditTicks;
      }else if(keyerState == KEYER_DAH_DOWN){
        keyerState = KEYER_DAH_SPACE;
        keyerTicksRemaining = ditTicks;
      }else{
        keyerState = KEYER_IDLE;
      }
    }
  }

  return keyDown;
}
```

```
// Workaround for BUG2
void selfTestSidetone(){
  if(SIDETONE_SELF_TEST){
    tone(sidetonePin, SIDETONE_FREQ);
    delay(200);
    noTone(sidetonePin);
    digitalWrite(sidetonePin, LOW);
  }
}
```

高松オリーブハムクラブ作製 Remote Keyer Original vs Modified Source Code (Client) (5/7)

```
//キーのUP/DOWNをサンプリングしてUDPでサーバに送信する関数
 void captureKey(){
  //if(digitalRead(keyPin) == key_down){
  if((PINB & 0b00000001) == key_down){ //処理を高速化するためにレジスタを直接操作している。意味はコメントアウトした上の行と同じ。
      delay(1);
      if((PINB & 0b00000001) == key_down){
        key_buf1[0] = key_buf1[0] | (1 << i); //キーダウン状態であれば、バッファのLSBを1にする。
        keyup_elapsed = 0; //無操作時間のカウンタをクリア
        send_pkt = true;///パケット送信許可フラグをtrueにする。
       }
  }else delay(1);
  i++;
  if(i > 7){ //8回目のサンプリングが終わったら、パケットの送信処理を行う。
    if(isNo1TrxSel){ //選択している無線機をサーバに通知するデータを設定
      key_buf1[1] = 0x00;
    }else{
      key_buf1[1] = 0x01;
    }
    if(send_pkt == true){ //パケットの送信処理
      Udp.beginPacket(DestIP, DestPort);
      Udp.write(key_buf1, 2);
      Udp.endPacket();
      //Serial.println(key_buf1[0], BIN);
    }
    //パケットの送信が終わったらバッファをクリアする。
    key_buf1[0] = 0;
    key_buf1[1] = 0;
    i = 0;
  }
}
```

```
//内蔵エレキーで生成したキー波形をUDPでサーバに送信する関数
void captureKey(){
  bool keyDown = updateKeyer();
  keyLedOn = keyDown;

  if(keyDown){
    key_buf1[0] = key_buf1[0] | (1 << i);
    keyup_elapsed = 0;
    send_pkt = true;
  }

  i++;
  if(i > 7){ //8回目のサンプリングが終わったら、パケットの送信処理を行う。
    if(isNo1TrxSel){
      key_buf1[1] = SEL_NO1_TRX;
    }else{
      key_buf1[1] = SEL_NO2_TRX;
    }

    if(send_pkt == true){
      packet_buf[0] = key_buf1[0];
      packet_buf[1] = key_buf1[1];
      packetReady = true;
    }

    key_buf1[0] = 0;
    key_buf1[1] = 0;
    i = 0;
  }
}
```

高松オリーブハムクラブ作製 Remote Keyer Original vs Modified Source Code (Client) (6/7)

```
void monitoringAndControl(){
  //サーバから応答パケットが返ってきているか監視し、LEDに状態表示する。
  packetSize = Udp.parsePacket();
  if(packetSize){
    digitalWrite(monLed, HIGH);
  }else{
    digitalWrite(monLed, LOW);
  }

  //電鍵の無操作時間を監視し、NOOPE_TIMELIMITに設定した時間を超えて操作がない場合は、パケットの送信フラグをfalseにしてパケットの送信を停止させる。
  if(send_pkt == true) keyup_elapsed++;
  if(keyup_elapsed > (NOOPE_TIMELIMIT / PKT_ARRIVE_INTERVAL))send_pkt = false;

  //無線機選択スイッチの状態を読み取り、結果をisNo1TrxSelに格納する。
  //if(digitalRead(selTrx) == selNo1Trx){
  if(((PIND>>7) & 0b00000001) == selNo1Trx){
    isNo1TrxSel = true;
  }else{
    isNo1TrxSel = false;
  }
}
```

```
void monitoringAndControl(){

  //パドルの無操作時間を監視し、NOOPE_TIMELIMITを超えた場合はパケット送信を停止する。
  if(send_pkt == true) keyup_elapsed++;

  if(keyup_elapsed > (NOOPE_TIMELIMIT / PKT_ARRIVE_INTERVAL)) send_pkt = false;

  //D7入力にかかわらず常に無線機1を選択する。
  isNo1TrxSel = true;
}
```

高松オリーブハムクラブ作製 Remote Keyer Original vs Modified Source Code (Client) (7/7)

```
void setup() {
  //通信関係の初期設定
  Serial.begin(19200);
  Ethernet.begin(mac, ip, dns, gateway, subnet);
  Udp.begin(LocalPort);

  //入出力ピンの初期設定
  pinMode(selTrx, INPUT_PULLUP);
  pinMode(keyPin, INPUT_PULLUP);
  pinMode(monLed, OUTPUT);



  //Timer2
  MsTimer2::set(PKT_ARRIVE_INTERVAL, monitoringAndControl);
  MsTimer2::start();

  //Timer1
  Timer1.initialize(SAMPLING_INTERVAL);
  Timer1.attachInterrupt(captureKey);

  //WDT
  wdt_enable(WDTO_2S);
}


void loop() {
  wdt_reset();
}
```

```
void setup() {
  //通信関係の初期設定
  Serial.begin(19200);
  Ethernet.begin(mac, ip, dns, gateway, subnet);
  Udp.begin(LocalPort);

  //入出力ピンの初期設定
  pinMode(dahPin, INPUT_PULLUP);
  pinMode(selTrx, INPUT_PULLUP);
  pinMode(ditPin, INPUT_PULLUP);
  pinMode(monLed, OUTPUT);
  digitalWrite(monLed, LOW);
  pinMode(sidetonePin, OUTPUT);
  selfTestSidetone();

  //Timer2
  MsTimer2::set(PKT_ARRIVE_INTERVAL, monitoringAndControl);
  MsTimer2::start();

  //Timer1
  Timer1.initialize(SAMPLING_INTERVAL);
  Timer1.attachInterrupt(captureKey);

  //WDT
  wdt_enable(WDTO_2S);
}

void loop() {
  byte packetToSend[2];
  bool shouldSend = false;

  noInterrupts();
  if(packetReady){
    packetToSend[0] = packet_buf[0];
    packetToSend[1] = packet_buf[1];
    packetReady = false;
    shouldSend = true;
  }
  interrupts();

  if(shouldSend){
    Udp.beginPacket(DestIP, DestPort);
    Udp.write(packetToSend, 2);
    Udp.endPacket();
  }

  //サーバからの応答パケットを読み捨てる。LEDはローカルキー信号のモニターとして使用する。
  packetSize = Udp.parsePacket();
  digitalWrite(monLed, keyLedOn ? HIGH : LOW);

  wdt_reset();
}
```

Original vs Modified Source Code (1/3)

主な変更点

1. スケッチ名と用途の変更
   変更後ファイル冒頭で、`パドル直結版 Ver2.1p` になっています。
   また `Modified by JG1LMK/M.Tsunoda`、`Keyer Function Added` のコメントが追加されています。

←Ver2.1 ベースに改造

2. 入力ピンの役割変更
   オリジナルでは D8 が単純なキー入力でした。

```cpp
const int keyPin = 8;
```

変更後は、D6/D8 がパドル入力になっています。

←パドル用に D8 を追加

```cpp
const int dahPin = 6;  // 長点パドル
const int selTrx = 7;  // 無線機選択
const int ditPin = 8;  // 短点パドル
const int monLed = 9;
```

Original vs Modified Source Code (2/3)

3. 内蔵エレキー機能の追加
   変更後には以下の設定が追加されています。

←内蔵エレキー機能は丸々追加した

```cpp
#define KEYER_WPM 15
#define SWAP_PADDLE_INPUTS false
#define KEYER_IAMBIC_B true
#define PADDLE_DEBOUNCE_TICKS 2
```

←現状 15 WPM に固定

← IAMBIC はスクイーズの機能のこと　B モード選択
Iambic A と Iambic B は、その「両方握って離したときに最後の1要素を出すかどうか」の違い
←パドル入力の チャタリング防止の強さを決める値
#define SAMPLING_INTERVAL 5000 // 5ミリ秒 の間隔でパドルの状態を読んで この回数同じ状態であれば、押された・離されたと認識しチャタリングを防止している

   つまり、パドル入力から短点・長点を生成します。速度は現在 `15 WPM` です。

4. エレキー状態管理の追加
   変更後には `KeyerState` が追加されています。

```cpp
KEYER_IDLE
KEYER_DIT_DOWN
KEYER_DIT_SPACE
KEYER_DAH_DOWN
KEYER_DAH_SPACE
```

←パドルに必要となるのは、これらの 5つのステート（状態）
- アイドル
- 短点 ON
- 短点 OFF
- 長点 ON
- 長点 OFF

   これで、短点送出中、長点送出中、スペース中、待機中を管理しています。

Original vs Modified Source Code (3/3)

5. チャタリング対策の追加
   `debouncePaddle()` が追加され、パドル入力が数サンプル連続して同じ状態になったら確定するようになっています。 ←前述した通り

6. キー波形生成方法の変更
   オリジナルでは D8 のキー入力を直接読んでいました。
   変更後は `updateKeyer()` でパドル状態からキーON/OFF波形を作り、それを `captureKey()` がUDP送信用バッファに詰めます。 ←ステートから判断

7. UDP送信形式は維持
   サーバーに送る2バイト構造は基本的に同じです。 ←UDPパケットは基本踏襲

```cpp
packetToSend[0] = キー波形
packetToSend[1] = TRX選択
```

   なのでサーバー側との通信仕様は大きく変えていません。

8. TRX選択の最新変更
   先ほど最後に加えた変更で、D7入力によるTRX選択を無効化し、常にTRX-1固定にしました。 ←サーバー側の無線機は＃１に固定

   [RemoteKey_client_paddle.ino](G:\Remote CW Keyer\toip_shield_jh5yvc\RemoteKey_client_paddle\RemoteKey_client_paddle.ino:229)

```cpp
//D7入力にかかわらず常に無線機1を選択する。
isNo1TrxSel = true;
```