Technical Info

Guy Marchal - ON5FM on5fm@uba.be
Rik Strobbe - ON7YD on7yd@uba.be

IC-7300 Paddle of Handpomp? Eenvoudig opgelost!

door ON7DQ Luc

Ik beschik sinds een viertal maanden over een Icom IC-7300 transceiver. Een prachtig toestel opgebouwd als een direct sampling SDR transceiver, met allerlei toeters en bellen, een gebruiksvriendelijk touchscreen en nog zoveel meer… maar daar zal ik het hier verder niet over hebben.

Eén ding ergerde mij echter al vanaf het begin: de ontwerpers hebben duidelijk niet aan de veelzijdige CW operator gedacht. Wie zoals ik al eens wisselt tussen paddle en de gewone handpomp komt met dit toestel bedrogen uit. Er is aan de IC-7300 slechts 1 KEY connector en deze zit dan ook nog eens op de achterkant. Om deze ingang voor paddle of handpomp te gebruiken, kan je weliswaar een paddle en key parallel aansluiten. Maar dan moet je toch steeds via een menu een instelling veranderen, en hiervoor maar liefst **ACHT** keer op een knop drukken!
(voor wie zelfs niet weet hoe je dit doet… druk MENU> KEYER > EDIT/SET > CW-KEY SET > KEY TYPE > Kies KEY of PADDLE > EXIT > HOLD M.SCOPE… dat is alles, hi)

Dat moest dus sneller en gemakkelijker kunnen… iedere transceiver van Icom kan namelijk bestuurd worden via de zogenaamde CI-V bus. Bij de IC-7300 heb je zelfs twee zo'n CI-V bussen, de ene zit in de USB aansluiting van de transceiver, die kan je dus gebruiken voor CAT-sturing vanuit een logprogramma of een programma voor digimodes. Daarnaast is er ook een klassieke Icom CI-V connector, en deze gebruiken we in dit project.

De CI-V bus

Om te begrijpen hoe ons Arduino project werkt, eerst even een korte inleiding over de CI-V bus.

Ik haalde veel informatie bij Jean-Jacques, ON7EQ (zie http://www.qsl.net/on7eq/en/ onder Projects > Arduino), maar vooral ook uit de "Bijbel van de CI-V", geschreven door Ekki, DF4OR (zie http://www.plicht.de/ekki/civ/index.html).

CI-V staat voor Computer Interface 5, en is al in gebruik bij Icom transceivers sedert de jaren 80. Het is een ééndraads bus op TTL niveau, de aansluiting is een 3,5 mm stereo stekker waarvan enkel de GND en de TIP gebruikt worden. Voor sturing vanuit een RS-232 poort (PC) is een interface nodig (bvb. op basis van een MAX232), maar met de Arduino kan deze vervallen, gezien we reeds op TTL niveau werken.

In rusttoestand is de bus hoog of "zwevend" (+5V). Als de computer iets wil sturen, wordt vanuit de computer de bus laag getrokken, als de transceiver iets stuurt, zal die de bus laag trekken. Het formaat van de data is NRZ (Non Return to Zero).

Bekijken we nu enkel wat we nodig hebben voor dit project, namelijk het sturen vanuit de "controller", in dit geval dus de Arduino.

Ieder bericht aan de transceiver bestaat uit een aantal vast opgelegde bytes, aangevuld met een **commando**. Sommige commando's hebben bovendien nog data nodig (een frequentie, een status).

IC-7300 Paddle ou clef Morse ? Une solution simple !

par ON7DQ Luc – traduit par ON7BAU Luc

Je possède depuis quelques mois un émetteur Icom IC-7300. Un appareil merveilleux, conçu comme un TRX SDR à échantillonnage direct, équipé de tous les gadgets et d'un écran tactile facile à manipuler, et bien plus encore.

Une chose me dérangeait depuis le début : les concepteurs n'ont clairement pas pensé à l'opérateur morse versatile. Celui qui, comme moi, change parfois entre le paddle et la "pioche" sera déçu par cet appareil : l'IC-7300 n'a qu'un seul connecteur KEY et il se trouve à l'arrière ! Pour employer cette connexion avec le paddle ou la clef on peut évidement connecter les deux en parallèle mais on est oblige de changer les paramètres via un menu, ce qui revient à triturer 8 fois les boutons !
(Pour ceux qui ne savent pas comment s'y prendre, pressez MENU > KEYER > EDIT/SET > CW-KEY SET > KEY TYPE > puis choisissez KEY ou PADDLE > EXIT > HOLD M.SCOPE … C'est tout HI.)

Cela devait aller plus vite et être plus facile. Tous les transceivers d'ICOM peuvent être contrôlés via le bus CI-V. L'IC-7300 possède même 2 de ces circuits de contrôle dont un via la connexion USB, qui peut servir pour le contrôle CAT depuis un logiciel "Logger" ou un programme pour les digimodes. Il y a aussi le connecteur classique Icom CI-V, celui-ci va nous servir pour notre projet.

Le circuit CI-V

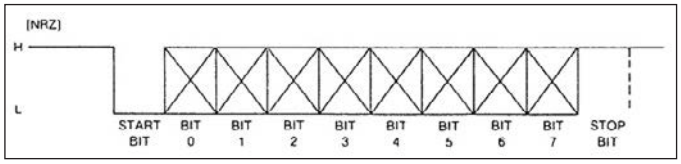
Pour comprendre le projet Arduino, voici d'abord une petite introduction du CI-V bus.

Beaucoup d'informations viennent de chez Jean-Jacques ON7EQ (voir http://www.qsl.net/on7eq/en/ sous projets > Arduino) mais aussi de la "bible de CI-V" écrit par Ekki, DF4OR (voir http://www.plicht.de/ekki/civ/index.html).

Le CI-V (Computer Interface 5) est employé dans les TRX Icom depuis les années 80. C'est un "bus" unifilaire au niveau TTL. La connexion se fait via un jack stéréo de 3.5 mm dont la masse et la boule sont uniquement connectés. Pour le contrôle à partir d'un port RS-232 (PC) on a besoin d'une interface (p. e. à partir d'un MAX232) mais, avec l'Arduino, ceci n'est plus nécessaire puisqu'on travaille déjà au niveau TTL.

Au repos, le niveau du bus est haut ou flottant (+5v). Si le computer veut envoyer quelque chose c'est lui qui descend le niveau du bus ; quand le TRX envoie des datas c'est lui qui va forcer le niveau bas. Le format data est NRZ (non retour au zéro).

Regardons maintenant ce dont on a besoin pour le projet, à savoir le contrôle à partir du contrôleur, dans ce cas, l'Arduino.



Een bericht start steeds met twee maal **$FE** ($ of 0x is het voorvoegsel om een hexadecimaal getal aan te duiden). Dan komt het adres van de transceiver, voor een IC-7300 is dit bvb. **$94**.

In mijn code zal je zien dat ik het adres $76 gebruik, dit komt omdat ik voor de WSPR en JT65 software het adres van een IC-7200 moest instellen om de CAT sturing te doen werken (de IC-7300 wordt nog niet in alle programma's ondersteund).

Daarna volgt het afzender adres, dit is steeds **$E0** (maar wordt eigenlijk nergens gebruikt in de transceiver). Nu komt het commando, gevolgd door een eventueel sub-commando en de data bytes.

Als afsluiting van het bericht moet een **$FD** gestuurd worden, en is de transceiver klaar voor een volgend bericht.

Er is nog een probleem met de data bytes, deze mogen niet in hexadecimaal verstuurd worden omdat er kans is dat de data een byte $FE of $FD zou bevatten (de start en stop codes). Daarom moet alle data in BCD codering gestuurd worden. Een getal 123 wordt dan bijvoorbeeld in **$01 $23** omgezet.

Een paar voorbeeldjes maken alles hopelijk duidelijk:

Set Mode:

$FE $FE $94 $E0 **$06 $00** $FD: zet de transceiver in de MODE LSB
$06 is het mode-set commando, **$00** = LSB, $01 = USB enz.
Je vindt alle modes in de uitgebreide manual van je transceiver of op de site van DF4OS.

Set USB AF/IF:

$FE $FE $94 $E0 $1A $05 $00 $59 **$00** $FD: zet de USB uitgang (signaal naar PC) in AF mode, met $01 stel je de IF mode in. Dit spaart alweer 5 toetsaanslagen op de transceiver!
Opmerking: voor wie hier nog niet mee gespeeld heeft: de AF mode gebruik je voor alle digimodes (bijv. met FLDIGI), met de IF mode kan je o.a. DRM ontvangen met het programma DReaM, zonder verdere hardware.

Set Power:

$FE $FE $94 $E0 $14 $0A **$00 $28** $FD: zet de power op 10 W
$FE $FE $94 $E0 $14 $0A **$02 $55** $FD: zet de power op 100 W

De power wordt eigenlijk als een % uitgedrukt. De data gaat van 0 tot 255, voor 10 watt heb je dus 10 % nodig, of ongeveer 255/10 = 26. Bij een test moest ik echter 28 sturen om op 10 W te komen.

Deze functie bespaart het eindeloze gedraai aan de multi-schakelaar, het zal allicht ook zijn levensduur verlengen…
Ik heb deze functie ook in mijn project verwerkt… en zonder extra knoppen! (zie code).
Door een lange druk op de KEY of PADDLE knop wordt het vermogen op respectievelijk 10 of 100 watt ingesteld.

Wie de moeite doet om eens de full manual te bekijken (op CD meegeleverd bij het toestel), zal in hoofdstuk 19 nog wel meer inspiratie vinden.

Nog een tip: om de verschillende commando's eens te testen voor je ze in de Arduino code opneemt, heeft DF4OS ook een handige Windows toepassing gemaakt: de CI-V Tester. Je kan deze hier downloaden: http://www.plicht.de/ekki/software/civtest.html.

Chaque message vers l'émetteur se compose d'un nombre fixe d'octets, complété par une **commande**. Certaines de ces commandes ont besoin en plus de data (p. ex. une fréquence ou un statut).

Un message démarre toujours avec deux fois **$FE** ($ ou 0x est le préfixe pour designer un nombre hexadécimal) puis l'adresse du TRX, pour l'IC-7300 c'est **$94**.

Dans mon code vous allez voir que j'emploie l'adresse $76, ceci parce que pour le WSPR et le JT65 je devais programmer l'adresse d'un IC-7200 pour que le contrôle CAD fonctionne (l'IC-7300 n'est pas encore repris dans tous les programmes).

Suit alors l'adresse de l'expéditeur. Ceci est toujours **$E0** (n'est, en fait, employé nulle part dans le transceiver) suivi de la commande et, éventuellement, suivi par une sous-commande et les octets de data.

A la fin du message, un **$FD** est émis et le TRX est de nouveau prêt pour le message suivant.

Il y a un autre problème avec les octets de données : ceux-ci ne peuvent pas être envoyés au format hexadécimal car les données pourraient contenir un octet $FE ou $FD (les codes Start et stop). Pour éviter cela, toutes les données doivent être émises en BCD. Le chiffre 123, par exemple, est convertit en **$01 $23**.

Quelques exemples pour vous éclairer :

Set Mode :

$FE $FE $94 $E0 **$06 $00** $FD : met le TRX en MODE LSB
$06 est la commande mode-set. **$00** = LSB, $01 = USB etc.
Tous les modes se retrouvent dans le manuel complet de votre émetteur-récepteur ou sur le site de DF4OS.

Set USB AF/IF :

$FE $FE $94 $E0 $1A $05 $00 $59 **$00** $FD : met la sortie USB (signal vers le PC) en mode AF ; avec $01 on détermine le mode IF. Ceci nous épargne 5 manipulations de touches.
Remarque : pour ceux qui n'en ont pas encore l'expérience, le mode AF s'emploie pour tous les modes digitaux (ex : avec FLDIG). En mode IF vous pouvez recevoir le DRM avec le logiciel DReaM sans d'avantage de matériel hardware.

Set Power :

$FE $FE $94 $E0 $14 $0A **$00 $28** $FD : met la puissance à 10 W
$FE $FE $94 $E0 $14 $0A **$02 $55** $FD : met la puissance à 100 W

La puissance d'émission s'exprime comme un pourcentage. La valeur va de 0 a 255, pour 10 W on a besoin de 10 %, ou environ 255/10= 26. Durant un test je devais monter la valeur à 28 pour obtenir 10 W.

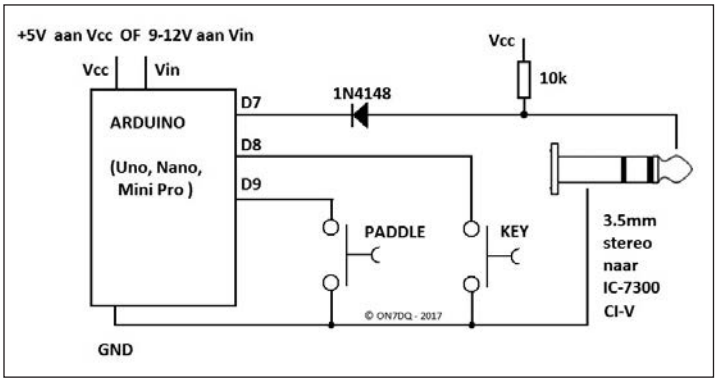
Cette fonction nous épargne la manipulation sans fin du commutateur multi et va certainement prolonger sa vie …
J'ai incorporé cette fonction dans le projet et … sans bouton supplémentaire ! (voir le code).
Quand on appuie longtemps sur la clef ou le paddle la puissance se règle respectivement à 10 ou 100 watt.

Ceux qui se donnent la peine de regarder le manuel (sur le CD livré avec l'appareil) trouveront encore plus d'inspiration au châpitre 9.

De Arduino Hardware

Het schema kan niet eenvoudiger zijn: een Arduino UNO of een van de kleinere varianten (Nano of Pro Mini), een diode, een weerstand en twee drukknoppen. Verder een 3.5 mm jack met een eind afgeschermde kabel.

Ook een eenvoudig project mag in een mooie behuizing zitten, en ik vond in mijn junkbox een oude Nintendo controller, met twee rode knoppen… ideaal!

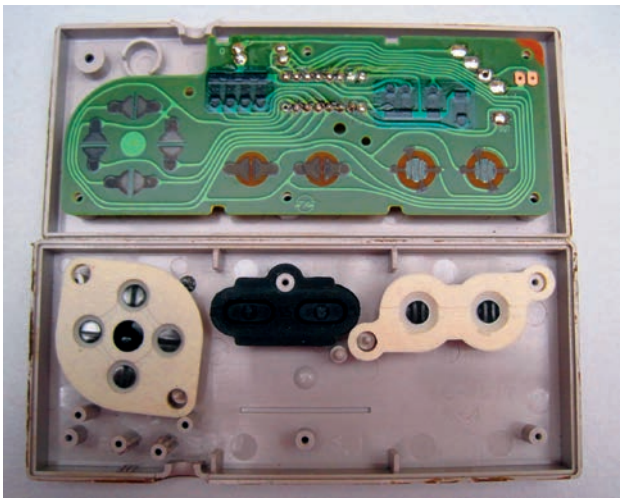


Encore une astuce : pour tester toutes les commandes avant de les mettre dans le code Arduino, DF4OS a crée une application Windows : le testeur CI-V. Vous pouvez le télécharger ici : http://www.plicht.de/ekki/software/civtest.html.

Arduino Hardware

Le schéma ne peut être plus simple : un Arduino Uno ou une de ses variantes plus petites

Binnenin zat een printplaatje met een 8-bits schuifregister (voor de 8 knopjes). Die chip heb ik verwijderd en zo had ik mooie soldeereilandjes die naar alle knopjes leiden. Voorlopig gebruik ik enkel de twee rode knoppen die je op de foto ziet, maar de andere kan ik allicht nog voor een andere functie gebruiken.
Wegens de beperkte plaats in het doosje, gebruikte ik een Arduino Pro Mini zonder pinnetjes, deze paste juist onder de originele printplaat. De nodige draadjes werden rechtstreeks aan de Arduino print gesoldeerd. Een stukje plakband zorgt ervoor dat er geen kortsluitingen optreden.



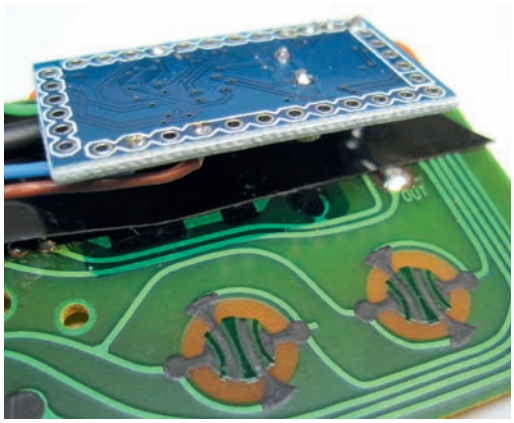
(Nano ou Pro Mini), une diode, une résistance et deux boutons-poussoirs, un jack de 3.5 mm et un bout de fil blindé.

Un simple projet peut aussi avoir une belle boîte. J'ai trouvé, dans ma boîte fourre-tout, un vieux contrôleur Nintendo avec deux boutons rouges. L'idéal !

A l'intérieur, se trouvait une plaquette avec un registre 8 bit (pour les 8 boutons), après l'enlèvement de ce chip il me restait des petits îlots de soudure qui menait aux boutons. Actuellement, je n'emploie que les boutons rouges que vous voyez sur les photos, les autres sont en réserve pour d'autres fonctions.
A cause de la place limitée dans la boite, j'emploie un Arduino Pro Mini sans ses broches, ceci passe tout juste en dessous de la plaquette d'origine. Les fils nécessaires sont directement soudés sur le circuit imprimé de l'Arduino. Un bout de tape évite les courts circuits.

Hier een foto van het afgewerkte product

Photo du produit fini



En een detail van de montage binnenin, met de twee "koolstof" schakelaars

Un détail du montage a l'intérieur, avec les deux interrupteurs "carbone"

Arduino Software

Wie er reeds mee werkte, weet dat een programma voor Arduino een "sketch" genoemd wordt. Een sketch wordt geschreven in een speciaal dialect van de taal C, met gebruik van libraries in de taal C++.

Een sketch bestaat steeds uit drie delen: eerst de declaraties, een setup() functie en een "loop()" functie. Deze laatste blijft eeuwig draaien… tot je de spanning aflegt.

Zowel de setup als de loop kunnen natuurlijk nog bijkomende functies oproepen, in dit project bijvoorbeeld de functie blink(), om de ingebouwde LED even te laten pinken.

Al bij al valt de code heel goed mee, je kan dit rustig als een beginners-project beschouwen.
De meeste uitleg staat in de code als commentaar (na de // tekens).

Arduino Software

Celui qui a déjà travaillé avec Arduino sait qu'un programme Arduino s'appelle un "sketch".

Le sketch est écrit dans un dialecte du langage C, avec emploi de bibliothèques en C++ et consiste en 3 parties. Premièrement les déclarations, une fonction setup () et une fonction loop (). La dernière tourne éternellement jusqu'à ce que l'on coupe le courant.

Aussi bien le setup que la loop peuvent appeler des fonctions supplémentaires. Dans ce projet, par exemple, la fonction Blink(), pour faire clignoter la LED incorporée.

Dans l'ensemble, le code n'est pas trop difficile. On peut considérer ceci comme un projet de débutant. La plupart des explications se trouvent dans le code comme commentaire (après les signes //)

```
// IC7300 Paddle <-> Straight Key switch
// by Luc - ON7DQ/KF0CR
// 20170105 started project with only 2 functions
// 20170201 extra code for : hold buttons sets PWR to 10W or 100W

#include <SoftwareSerial.h> // for comms to IC7300
#define BAUD_RATE 9600      // CI-V speed

// serial connection
// RX = IC7300 to Arduino : to pin 6 via resistor 4k7 (not used now)
// TX = Arduino to IC7300 : direct to pin 7, with pull up 10k to Vcc (5V)
SoftwareSerial mySerial(6, 7); // (RX, TX)

int keyPin     = 8;   // key button to GND
int paddlePin  = 9;   // paddle button to GND
int ledPin     = 13;  // internal LED
int count      = 0;   // counter for press time
```

Dans le code, on relève une commande assez bizarre :
mySerial.write((byte)00);
Celle-ci s'écarte des autres commandes parce qu'on ne peut pas écrire un caractère **NUL** ; **write(00)** causerai une faute (error).

Conclusion

Bien sur on peut agrandir le hardware et le software avec d'autres commandes ; c'est une question de quelques boutons supplémentaires et quelques lignes de code.

La première extension est déjà incorporée dans le projet : le double emploi des boutons (appui court ou long). Dans mon application j'ai mis les deux positions de puissance 10/100 watts mais, naturellement, on peut y assigner d'autres fonctions.

Encore une idée pour les spécialistes : une version auto sens. Si l'on appuie brièvement sur le paddle, le TRX change vers paddle,

In de code zie je hier en daar een eigenaardige opdracht: **mySerial.write((byte)00);**
Deze wijkt af van de andere write opdrachten, omdat je geen **NUL** karakter mag wegschrijven, met **write(00)** zou je een foutmelding krijgen.

Conclusie

Uiteraard kan je de hardware en software uitbreiden met de andere besproken commando's, het is enkel een kwestie van wat extra knopjes en wat lijnen code.

De eerste uitbreiding heb ik reeds in het project verwerkt: het dubbel gebruik van de knopjes (kort of lang indrukken). In mijn toepassing heb ik er de twee vermogensstanden 10/100 watt in gestopt, maar je kan er natuurlijk ook iets anders aan toekennen.

Nog een idee voor de bollebozen: een "auto sense" versie. Als je even op de paddle drukt, schakelt de transceiver naar paddle, idem voor de key. Ik heb dit geprobeerd, maar er is natuurlijk een probleem. Het contact voor de punten van de paddle, en de gewone key staan parallel. Bij het eerste punt dat je seint ziet de Arduino dit aan voor een "key press" en schakelt terug om naar key mode. Om dit op te lossen zal je dus paddle en key apart moeten aansluiten en via een relais omschakelen, zodat de juiste sleutel aan de transceiver hangt, en de andere door de Arduino "gemonitord" wordt. Dit ging al vlug een ingewikkeld ding worden, en dat idee heb ik dus maar snel laten varen.

Nog een ander idee kon ik helaas niet uitvoeren. Ik had ook graag 4 knopjes in dit project opgenomen om de eerste vier keyer memories te laten uitzenden. Helaas… blijkbaar zijn er wel CI-V commando's om de memories te programmeren (de tekst die erin zit), maar er zijn GEEN commando's om de berichten te doen uitzenden. Hiervoor is er wel een schakeling in de manual, om dit via de microfoon connector te doen, maar dan heb je weer een ander bakje nodig. Oplossing: integreer beide projecten in één bakje. Allicht levert dat weer voer voor een volgend artikel.

Veel knutsel & CW plezier!

Luc – ON7DQ

```
int longTime   = 500; // limit for short/long press
boolean paddle = true;  // start in paddle mode

void setup()
{
 // connect to IC7300
 mySerial.begin(BAUD _ RATE);
 //mySerial.write(0x00); //send any initialization command if needed, not used yet

 pinMode(keyPin, INPUT);
 digitalWrite(keyPin, HIGH); // turn on internal pull-up on the inputPin

 pinMode(paddlePin, INPUT);
 digitalWrite(paddlePin, HIGH); // turn on internal pull-up on the inputPin

 pinMode(ledPin, OUTPUT);
}

void loop()
{
 count = 0;
 while (digitalRead(keyPin) == LOW) {
  count++; delay(1);
 }
 if (count > 50 && count < longTime) {
  // set rig to straight key input
  blink();
  paddle = false;
  mySerial.flush();
  mySerial.write(0xFE); mySerial.write(0xFE); // start
  mySerial.write(0x76); // radio address, changed from normal address
  mySerial.write(0xE0);
  mySerial.write(0x1A); mySerial.write(0x05); // command    1A 05
  mySerial.write(0x01); mySerial.write(0x64); // subcommand 164 in BCD
  mySerial.write((byte)00);                   // data
  mySerial.write(0xFD);                       // stop
 }
 if (count > longTime) {
  // set PWR to 10W
  blink();
  mySerial.flush();
  mySerial.write(0xFE); mySerial.write(0xFE); mySerial.write(0x76);
  mySerial.write(0xE0); mySerial.write(0x14); mySerial.write(0x0A);
    mySerial.write((byte)00); mySerial.write(0x28); mySerial.write(0xFD);
 }

 count = 0;
 while (digitalRead(paddlePin) == LOW) {
  count++; delay(1);
 }
```

```
if (count > 50 && count < longTime) {
  // set rig to paddle input
  paddle = true;
  mySerial.flush();
  mySerial.write(0xFE); mySerial.write(0xFE); mySerial.write(0x76);
  mySerial.write(0xE0); mySerial.write(0x1A); mySerial.write(0x05);
  mySerial.write(0x01); mySerial.write(0x64); mySerial.write(0x02);
  mySerial.write(0xFD);
 }
 if (count > longTime) {
  // set PWR to 100W
  mySerial.flush();
  mySerial.write(0xFE); mySerial.write(0xFE); mySerial.write(0x76);
  mySerial.write(0xE0); mySerial.write(0x14); mySerial.write(0x0A);
  mySerial.write(0x02); mySerial.write(0x55); mySerial.write(0xFD);
 }
 delay(100);
}

void blink() {
 digitalWrite(ledPin, HIGH); delay(250);
 digitalWrite(ledPin, LOW);  delay(250);
}
```

idem pour la clef. J'ai essayé mais il y a un problème. Les contacts du paddle et du manipulateur sont en parallèle. Au premier dih envoyé, l'Arduino interprète cela comme un key press et change en KEY mode.
Pour résoudre cela on doit brancher paddle et key séparément et faire le switching par un relais pour que la clef correcte soit raccordée au transceiver et que l'autre soit sous monitoring de l'Arduino. Cela menait très vite a quelque chose de très compliqué, alors j'en ai laissé tomber l'idée.

Encore une autre idée non réalisable : j'aurais aimé incorporer 4 boutons pour émettre les 4 messages en mémoire. Hélas il semble qu'il existe des commandes CI-V pour programmer les mémoires (le texte) mais il n'y a pas de commande pour émettre ce texte.
Pour cela il y a un schéma dans le manuel pour réaliser cela par le connecteur du micro, ce qui nécessite encore une autre boîte.
Solution : intégrer les deux projets dans la même box. Cela constituera certainement un sujet pour un autre article.

Beaucoup de plaisir avec ce bricolage et avec le morse.

Luc – ON7DQ