Interface homme/machine universelle pour transceiver

Dominique Boell, F4FEI

1ère partie - Plate-forme de développement. Les fonctions de base : affichage, boutons poussoirs.

2ème partie - Les fonctions avancées : encodage optique, synthèse en fréquence (DDS), sauvegarde des données.

3ème partie - Gestion des menus - Les fonctions annexes : scanneur, horloge, thermomètre...

INTRODUCTION

Dans la première partie, nous avons vu la façon d'utiliser la plate-forme de développement logiciel ARDUINO UNO pour concevoir le programme d'une interface qui permet à un opérateur d'activer les fonctions d'un transceiver et d'obtenir de ce dernier des informations en retour à afficher sur la face avant.

Les fonctions de base étudiées étaient l'affichage sur un écran LCD de ces informations, le traitement des niveaux du S-mètre et du ROS-mètre, le RIT, la détection des boutons poussoirs et leur utilisation.

Une première maquette a été réalisée sur une platine d'essais pour valider la faisabilité d'une telle interface tant sur le plan électrique que logiciel, la réalisation finale restant à l'initiative de chacun. Pour ma part, cette interface sera au format d'une face avant d'autoradio (*figure 2-1*).

Figure 2-1 :

Dans la deuxième partie de cet article vont être étudiées des fonctions dites avancées, telles que la synthèse directe en fréquence (DDS), la gestion de périphériques reliés à un bus I2C, avec comme première application, la sauvegarde des données, et un encodeur optique permettant d'accorder en fréquence le transceiver à l'aide d'un bouton rotatif.

1. ENCODEUR OPTIQUE

Pour commander l'accord en fréquence sur le transceiver, il est plus ergonomique d'utiliser un bouton rotatif couplé à un encodeur optique. Ce bouton rotatif peut aussi servir à monter (↑) ou descendre (↓) dans les différentes structures de menus. Il va ainsi remplacer les touches 5 et 6 et les laisser libres pour activer d'autres fonctions.

Pour réaliser cet encodeur optique, plusieurs options sont possibles : en acheter un, en fabriquer un de A à Z, le récupérer dans une souris obsolète d'ordinateur. C'est cette dernière solution que j'ai choisie (*figure 2-2*) sur la base de plusieurs réalisations trouvées sur Internet (taper « mouse encoder » dans votre moteur de recherche).

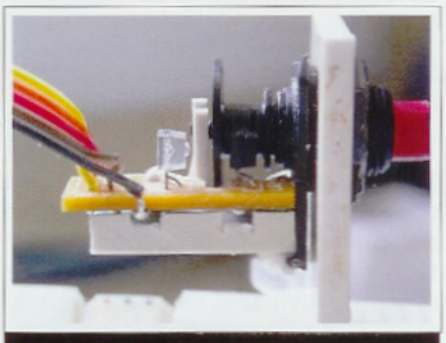

Figure 2-2 :

La partie optique est prélevée sur la souris, ici une Logicom à fils. La partie du circuit imprimé qui supporte la diode IR, le double phototransistor et quelques pastilles de connexion est découpée avec une mini perceuse équipée d'une petite scie circulaire.

L'axe de commande et son support proviennent d'un potentiomètre démonté, ce qui va permettre de fixer facilement l'encodeur sur la face avant du transceiver. L'axe du disque perforé de l'encodeur est enfoncé en force dans l'axe en plastique du potentiomètre percé au diamètre de l'axe du disque, moins 1/10 de mm. Le circuit imprimé est collé au pistolet à colle, ou mieux à la colle bi-composant de façon à ce que le double phototransistor voie la diode à travers les trous. Le mieux est de respecter la position originale dans la souris en découpant également la partie du fond de la souris qui comporte le support de l'axe du disque perforé et l'ergot de positionnement du circuit imprimé.

Le schéma électrique de l'encodeur est représenté *figure 2-3* :

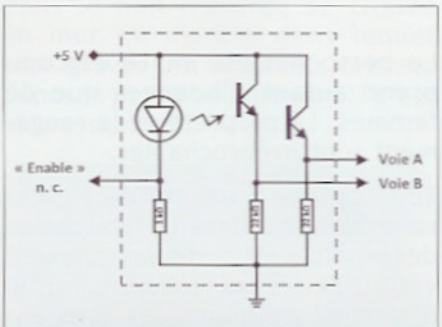

Figure 2-3 :

Les trois résistances (CMS) sont directement soudées sur le circuit imprimé.

5 fils sont disponibles :

- l'alimentation, en principe +5 V,
- une entrée « enable »,
- une sortie A,
- une sortie B,
- la masse.

Dans cette application, l'entrée « enable » n'est pas utilisée.

Deux phototransistors placés judicieusement dans un seul boîtier face à la photodiode fournissent des signaux en quadrature quand le disque perforé tourne.

Le programme va devoir déterminer le sens de rotation du disque ainsi que le nombre d'impulsions générées par sa rotation. Ces impulsions serviront soit à aug-

menter ou à diminuer la fréquence, soit à naviguer dans les menus.

Le disque est muni de 64 trous, ce qui va générer 64 impulsions par tour. En comparaison, les encodeurs des transceivers du commerce fournissent en général 200 impulsions par tour.

La conception d'un tel programme pourrait rebuter le débutant, mais la communauté ARDUINO est encore là pour nous aider. En effet, sur le site ci-dessous se trouve un tutoriel qui traite le sujet en détail et propose plusieurs exemples de codes dont un a été adapté ici.

Voir le programme ci-contre (fichier IHM_Partie_2_Encodeur)

Adaptation du site :
http://playground.arduino.cc/Main/RotaryEncoders

Commentaires sur le programme :

1. Une fois l'encodeur correctement relié au module ARDUINO, ce programme de test permet de voir fonctionner l'encodeur. En tournant le bouton, la première ligne indique le nombre de pas générés, la seconde ligne le sens de rotation.
2. La déclaration volatile, avant les variables position_encodeur et sens_encodeur, indique au microcontrôleur qu'il doit mettre ces variables en mémoire RAM plutôt que dans des registres, en particulier lors des traitements par interruption.
3. Les ports digitaux D2 et D3 sont les seuls ports utilisables pour gérer des interruptions. Voir le site ARDUINO pour connaître les autres fonctions utilisables pour gérer les interruptions.
4. Lorsqu'un des deux ports reçoit une transition montante, le traitement de l'interruption est déclenché en fonction de l'état de l'autre port. Dans l'exemple, le traitement de chaque interruption consiste à augmenter ou à diminuer la position de l'encodeur. Dans le programme complet de la partie 2 IHM_Partie_2, le traitement des interruptions est affecté au réglage de la fréquence et à celui du décalage du RIT.

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(8, 9, 4, 5, 6, 7);
#define encodeur_A 2 // Déclaration de la constante encodeur_A égale à 2
#define encodeur_B 3 // Déclaration de la constante encodeur_B égale à 3
volatile int position_encodeur ; // Variable indiquant la position de l'encodeur
volatile int sens_encodeur ; // Var iable indiquant le sens de rotation de l'encodeur
boolean etat_A ; // Etat de la voie A
boolean etat_B ; // Etat de la voie B
void setup()
 {
  lcd.begin(16, 2) ;
   pinMode(encodeur_A, INPUT) ; // Le port digital D2 (car encodeur_A = 2) est déclaré en entrée
   pinMode(encodeur_B, INPUT) ; // Le port digital D3 (car encodeur_B = 3) est déclaré en entrée
  // Interruption 0 provoquée par le changement d'état du port digital D2 traitée par Traitement_A
 attachInterrupt(0, Traitement_A, CHANGE) ;
  // Interruption 1 provoquée par le changement d'état du port digital D3 traitée par Traitement_B
 attachInterrupt(1, Traitement_B, CHANGE) ;
 }
void loop()
{
//Indique le sens et la position de l'encodeur
  lcd.clear() ;
  lcd.setCursor(0,0) ;
  lcd.print(position_encodeur) ;
  lcd.setCursor(0,1) ;
  if (sens_encodeur >0)
  {
  lcd.print("Sens horaire") ;
  }
  else
  {
    lcd.print("Anti horaire");
  }
  delay(10);
  }
// Traitement de l'interruption générée par le changement d'état de A
void Traitement_A()
{
  // Transition montante?
  if ((digitalRead(encodeur_A) == HIGH)&(!etat_B))
  {
    etat_A = true ;
    // Instructions à exécuter, par exemple :
    position_encodeur = position_encodeur + 1 ;
    sens_encodeur = +1 ;
    // Fin des instructions à exécuter.
  }
  // Transition descendante?
  if (digitalRead(encodeur_A) == LOW)
  {
    etat_A = false ;
  }
}
// Traitement de l'interruption générée par le changement d'état de B
void Traitement_B()
{
  // Transition montante?
  if ((digitalRead(encodeur_B) == HIGH)&(!etat_A))
  {
    etat_B = true ;
    // Instructions à exécuter, par exemple :
    position_encodeur = position_encodeur - 1;
    sens_encodeur = -1;
    // Fin des instructions à exécuter.
  }
  // Transition descendante?
  if (digitalRead(encodeur_B) == LOW)
  {
    etat_B = false ;
  }
}
```

2. LA SYNTHÈSE DIRECTE EN FRÉQUENCE - DDS

Maintenant que l'afficheur et l'encodeur rotatif sont opérationnels, nous allons nous intéresser à la synthèse directe en fréquence.

Le choix s'est porté sur le circuit DDS d'Analog Digital AD9851.

Il est bien sûr possible de se procurer le composant seul, mais il existe des modules tout câblés et fonctionnels que l'on peut acquérir sur un site d'enchères bien connu pour un prix inférieur au circuit intégré seul acheté sous nos latitudes.

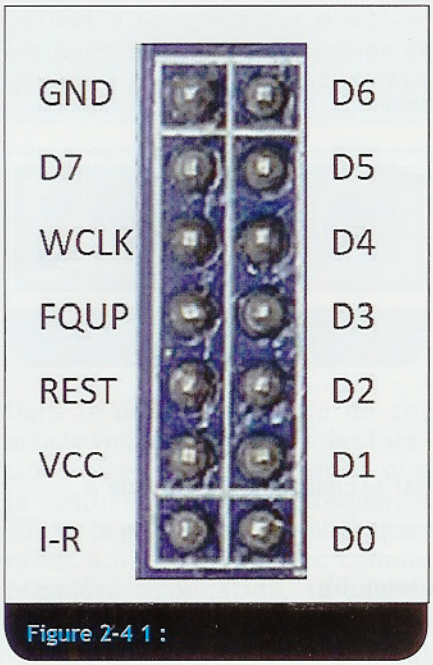

Figure 2-4 1 :

Le module retenu ici est représenté *figure 2-4* ainsi que son schéma *figure 2-5* :

Attention : il y a une petite erreur sur le schéma fourni. En effet, GND et D7 doivent être intervertis sur le connecteur 7 (J5). Se référer à la sérigraphie sur le circuit imprimé pour un câblage correct.

Le grand intérêt de ce module, c'est qu'il ne demande que 3 fils de commande (plus la masse et le + 5 V) et quelques lignes de programme pour synthétiser une fréquence jusqu'à 70 MHz, qui servira d'oscillateur local après filtrage et amplification en fonction de votre application.

D'autres modules à base d'AD9851 sont disponibles. Je ne les ai pas testés, mais ils devraient être compatibles.

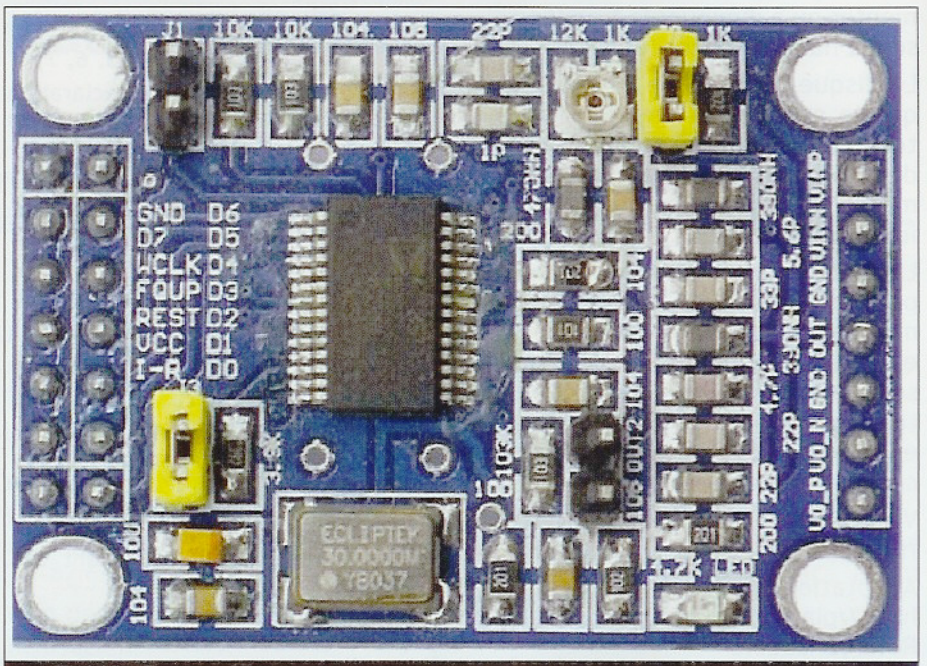

Figure 2.4 2 :

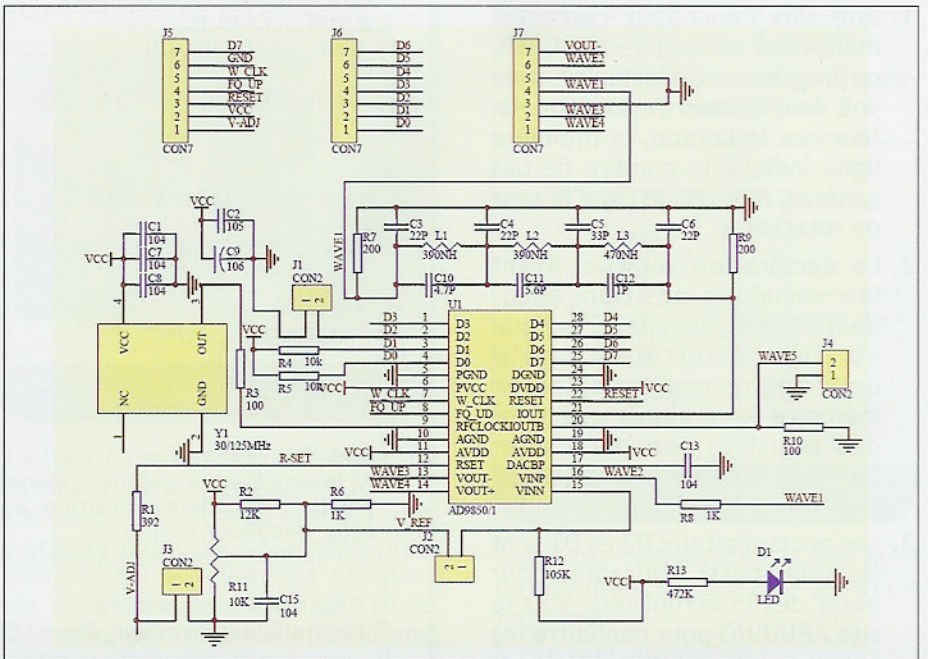

Figure 2.5 :

Remarque : on pourrait objecter, non sans raison, que la synthèse directe en fréquence avec un AD9851 n'est pas forcément le meilleur choix en termes de bruit de phase et de pureté spectrale. Pour l'application qui en est faite, il reste tout de même très intéressant compte tenu de ses performances et de sa simplicité de mise en œuvre.

Pour la programmation, nous n'allons pas faire appel à une bibliothèque, mais plutôt à la communauté ARDUINO qui s'est bien penchée sur le sujet, pour notre bonheur.

Le code retenu est celui de George Smart, M1GEO réalisé à partir de celui publié sur le blog de Peter B Marks : http://blog.marxy.org/2008/05/controlling-ad9851-dds-with-arduino.html.

Grâce à l'encodeur optique, la fréquence synthétisée peut être augmentée ou diminuée selon le pas en fréquence désiré (10 Hz, 100 Hz, 1kHz ou 10 kHz).

Le câblage du module DDS est des plus simples (*figure 2-6*).

(voir page suivante)

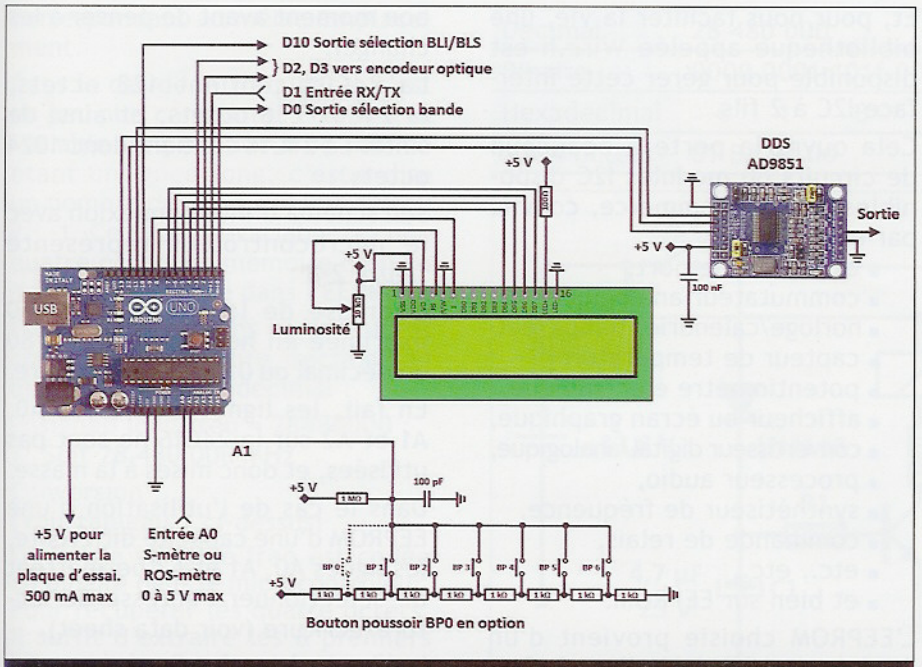

Figure 2.6 :

Attention : le module DDS ci-dessus ne s'enfiche pas directement sur toutes les plaques d'essais sans soudure. Dans ce cas, il faut se procurer un connecteur, par exemple, récupéré avec sa nappe sur un vieux téléviseur obsolète.

Le programme IHM_Partie_2_DDS_Encodeur comprend le programme de la partie 1, plus la gestion de l'encodeur optique et du synthétiseur de fréquence.

Au passage, la gestion BLI/BLS a été légèrement modifiée pour garder en mémoire, lors du changement de bande, l'état BLI ou BLS de chaque bande.

Comme indiqué dans l'article précédent, les programmes sont disponibles sur le site du REF-Union à la rubrique « Compléments des articles techniques », sous la forme de fichiers .ino directement utilisables avec la plate-forme de développement ARDUINO.

Commentaires sur le programme :

1. Dans le programme de l'encodeur optique ci-dessus, les instructions de comptage des pas et du sens de rotation dans le traitement des interruptions sont remplacées par l'incrémentation ou la décrémentation en fréquence.
2. Le programme principal void loop () fait appel à la fonction envoyerFrequence (freq) pour que le module DDS synthétise la fréquence demandée (variable : freq). La fiche technique de l'AD9851 indique que la fréquence est calculée selon la formule suivante : freq = [(mot de synthèse sur 32 bits) x horloge] / 2^{32}

Le programme calcule le mot de synthèse (tuning_word) de la façon suivante :

tuning_word = (freq * pow(2, 32)) / DDS_CLOCK;

avec DDS_CLOCK égal à la fréquence d'horloge, soit 180 MHz.

Ce mot est envoyé bit par bit vers les ports du module avec l'aide des instructions byte_out (), outOne () et outZero ().

3. La précision du synthétiseur dépend fortement de l'horloge de référence. Sur le module utilisé, l'horloge à 30 MHz est multipliée par 6 pour obtenir le 180 MHz.

   S'il y a un écart sur le 30 MHz, il est multiplié par 6 à 180 MHz. Le mieux serait de pouvoir calibrer le synthétiseur par rapport à une fréquence étalon, ou de mesurer sa fréquence avec un fréquencemètre étalonné.

   En jouant sur la valeur de l'horloge fixée à 180 000 000 par la déclaration #define DDS_CLOCK 180000000 en début de programme, on peut obtenir une fréquence synthétisée identique à la fréquence affichée, au hertz près (*figure 2-7*). (voir ci-dessus)
4. La vitesse de synthèse en fréquence est temporisée par le retard de 100 ms introduit par l'instruction delay (100) à la fin de la fonction byte_out (). Pour augmenter ou diminuer cette vitesse, il faut jouer sur la valeur du retard tout en s'assurant que la synthèse se fait de façon continue (sans à-coup).

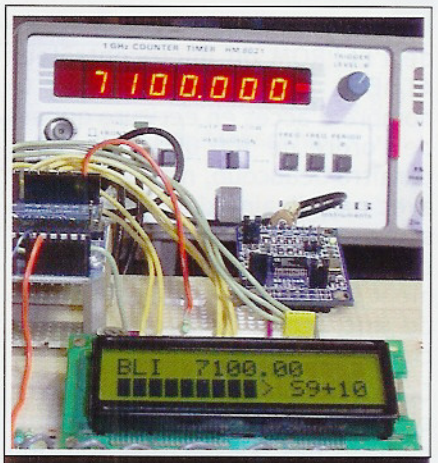

Figure 2-7 :

3. LA SAUVEGARDE DES PARAMÈTRES

• Principe :

La sauvegarde des paramètres est utile lorsque le transceiver comporte beaucoup de fonctions et de réglages dont l'opérateur souhaite retrouver instantanément la configuration pour être rapidement opérationnel lorsqu'il rallume son équipement.

Le projet d'interface homme-machine présenté ici va piloter un transceiver simple, sans beaucoup de paramètres à régler, ce qui signifie qu'on pourrait se passer de leur sauvegarde.

Mais cette interface étant appelée à évoluer et se voulant « universelle », il était important de la munir de la sauvegarde des paramètres.

Les paramètres de base retenus à ce stade sont les suivants :

- index du pas en fréquence,
- bande,
- mode BLU sur 40 m,
- mode BLU sur 10 m,
- fréquence courante sur 40 m,
- fréquence courante 10 m

D'autres paramètres, tels que ceux gérés par les menus seront sauvegardés ultérieurement, en particulier dans la troisième partie de cette série d'articles.

Il serait tentant d'utiliser la mémoire EEPROM du microcontrôleur qui a une taille de 1024 octets, largement suffisante pour stocker ces paramètres.

Malheureusement, la durée de vie de l'EEPROM du microcontrôleur ATMEGA 328P est limitée par le nombre d'accès en écriture qui est de l'ordre de 100 000. Cela peut paraître beaucoup, mais en réalité, avec une dizaine de paramètres à sauvegarder quand on éteint l'appareil, cela réduit considérablement la durée de vie de l'EEPROM du microcontrôleur qu'il faudra changer à terme.

Compte-tenu de son faible coût, c'est envisageable, mais pourquoi ne pas utiliser une solution plus pérenne, avec une EEPROM externe.

C'est la solution retenue ici, d'autant qu'elle va nous permettre d'apprendre à utiliser une fonctionnalité du microcontrôleur très intéressante qui est le bus I2C.

En effet, le microcontrôleur dispose de 2 entrées/sorties en A4 et A5 sur son port analogique qui sont dédiées au bus I2C.

Et, pour nous faciliter la vie, une bibliothèque appelée Wire.h est disponible pour gérer cette interface I2C à 2 fils.

Cela ouvre la porte à beaucoup de circuits ou modules I2C disponibles dans le commerce, comme par exemple :

- extension de port,
- commutateur analogique,
- horloge/calendrier temps réel,
- capteur de température,
- potentiomètre électronique,
- afficheur ou écran graphique,
- convertisseur digital/analogique,
- processeur audio,
- synthétiseur de fréquence,
- commande de relais,
- etc., etc.,
- et bien sûr EEPROM.

L'EEPROM choisie provient d'un fond de tiroir, il s'agit d'une 24C16, mais tout autre modèle de cette série 24C01, 24C02, 24C04, 24C08, etc. ou équivalent devrait convenir. La première chose à faire est de regarder sa fiche descriptive (data sheet) et de déterminer son brochage et son adresse.

Ces EEPROM peuvent endurer au minimum un million de cycles d'écriture, ce qui nous laisse un bon moment avant de penser à les changer.

La 24C01 contient 128 octets, la 24C02 256 octets, et ainsi de suite. La 24C16 contient donc 1024 octets.

Son schéma d'interconnexion avec le microcontrôleur représenté *figure 2-8* :

L'adresse de la 24C16 est 0x50 exprimée en hexadécimal, ou 80 en décimal ou 01010000 en binaire.

En fait, les lignes d'adresses A0, A1 et A2 sur la 24C16 ne sont pas utilisées, et donc mises à la masse.

Dans le cas de l'utilisation d'une EEPROM d'une capacité différente, les lignes A0, A1 et A2 permettent de sélectionner l'adresse de lecture/écriture (voir data sheet).

Les paramètres à sauvegarder sont en fait les variables utilisées dans le programme.

Prenons l'exemple de la bande. La variable associée est bande. Elle est déclarée de type byte et occupe donc un octet en mémoire. Sa sauvegarde va consister à écrire sa valeur, telle quelle, dans l'EEPROM à une adresse unique. Pour la récupérer il suffit de lire l'octet

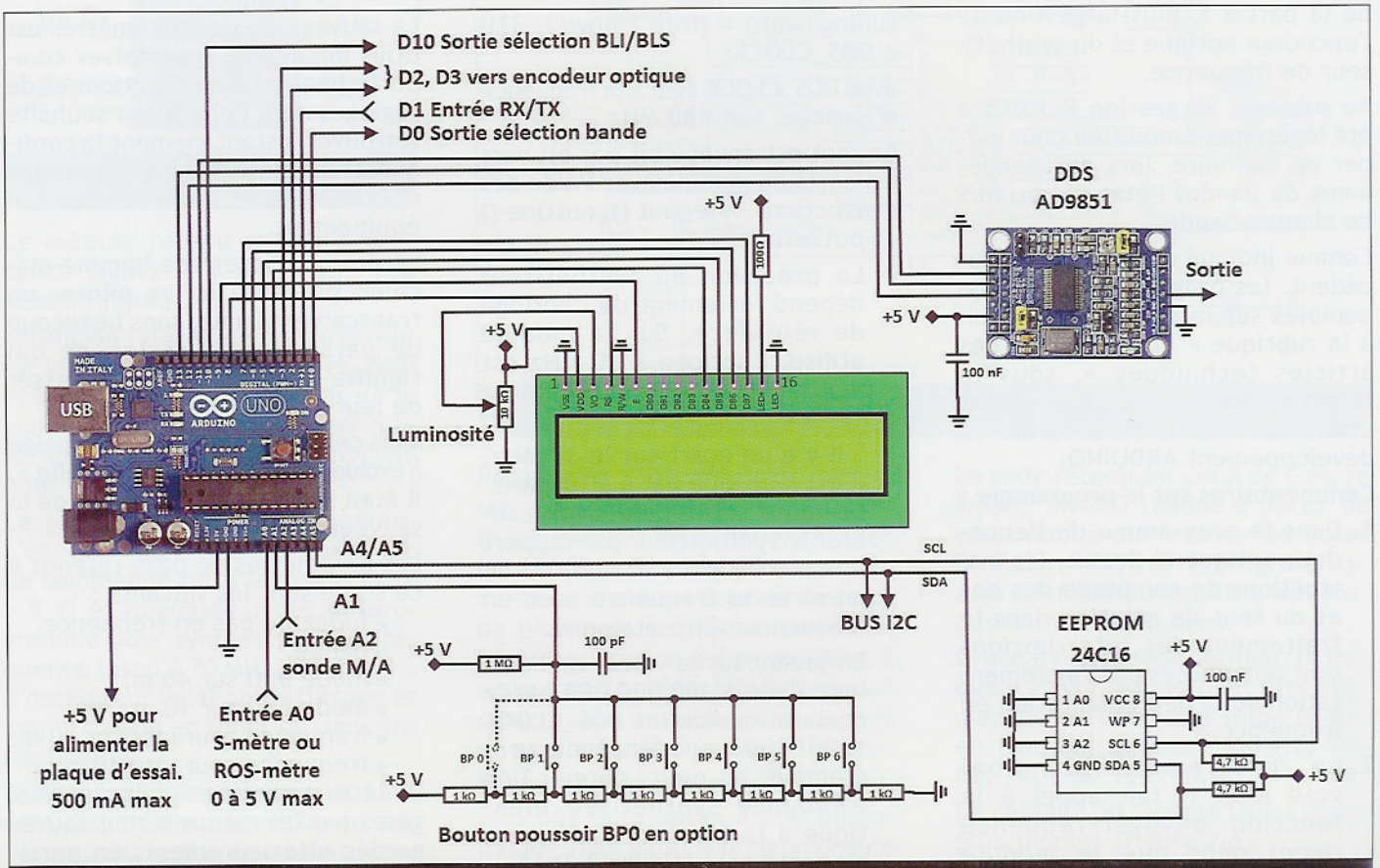

Figure 2.8 :

correspondant au même emplacement.

Dans le cas de la fréquence, c'est un peu plus compliqué. En effet, la variable freq est déclarée comme étant unsigned long, c'est-à-dire un nombre de 0 à 4 294 967 295 ou 2^{32} -1. Ce type de variable occupe quatre octets en mémoire qu'il va donc falloir écrire dans l'EEPROM.

Pour cela regardons comment est converti un nombre décimal en binaire et en hexadécimal :

```
unsigned long freq = 28480000 ;
// Soit 28,480 000 MHz
```

Conversion :
(Voir tableau 1 ci-dessus)

En fait la variable freq est codée en binaire dans le microcontrôleur. Pour récupérer les quatre octets, il suffit d'extraire les 8 premiers bits, puis les 8 suivants, etc. C'est le rôle des lignes de code suivantes en utilisant le décalage à droite (>>) du mot binaire de la fréquence et le masque 0xFF :

```
Octet3_PF = (freq>>24) & (0xFF);
// Poids Forts
Octet2_PF = (freq>>16) & (0xFF);
// Poids Forts
Octet1_Pf = (freq>>8) & (0xFF);
// Poids faibles
Octet0_Pf = freq & (0xFF); // Poids faibles
```

La reconstitution de la fréquence après sauvegarde consiste à assembler les 4 octets sauvegardés en utilisant le décalage gauche (<<) de la façon suivante :

```
Octets_PF = ((Octet3_PF<<8) & 0xFF00) + (Octet2_PF & 0x00FF);
// Assemblage des poids forts
Octets_Pf = (Octet1_Pf <<8) & 0xFF00) + (Octet0_Pf & 0x00FF);
// Assemblage des poids faibles
```

Et enfin :

```
freq = (((Octets_PF<<16) & 0xFFFF0000) + (Octets_Pf & 0x0000FFFF));  // Poids forts + Poids faibles.
```

La sauvegarde et la récupération des paramètres sont contenues dans le programme IHM_Partie_2 dans deux parties distinctes :

1. la récupération des paramètres se trouve dans le setup () avec le plan de mémoire de l'EEPROM sous forme d'un tableau déclaré byte memoire [12] contenant tous les octets des paramètres (ici 11, chaque position mémoire étant indexée de 0 à 11).

   La récupération des paramètres consiste à lire les octets de la mémoire un par un grâce à :

```
for (byte i = 0; i < 12; i=i+1) //
  Lecture octet par octet
    {
    memoire [i] = eeprom_read (0x50, i);
    }
```

   La fonction eeprom_read (int device_address, int mem_address) utilise les fonctions de la bibliothèque Wire.h pour lire un octet à son adresse mem_address dans l'EEPROM (ici de 0 à 11), l'EEPROM ayant pour adresse device_address, ici 0x50.
2. la sauvegarde des données se trouve dans la partie « Gestion de l'EEPROM » à la fin du programme.

Trois fonctions y sont définies :

- void sauve_param (),
- void eeprom_write(byte device_address, byte mem_address, byte data),
- byte eeprom_read(int device_address, int mem_address) expliquée ci-dessus.

La fonction void sauve_param () convertit les fréquences courantes des bandes 40 m et 10 m vers 4 octets pour chacune, tandis que l'écriture en mémoire se fait à l'aide de la fonction void eeprom_write (byte device_address, byte mem_address, byte data) selon le même principe que la lecture décrite ci-dessus.

Note : les fonctions eeprom_write () et eeprom_read () ont été récupérées à partir de plusieurs tutoriels ARDUINO disponibles sur la toile, sans qu'il soit possible d'en nommer l'auteur initial. Donc, encore une fois, merci à la communauté ARDUINO.

Décimal	28 480 000
Binaire	0000 0001 1011 0010 1001 0010 0000 0000
Hexadécimal	0 1 B 2 9 2 0 0
Sur 4 octets	01 B2 92 00

Tableau 1

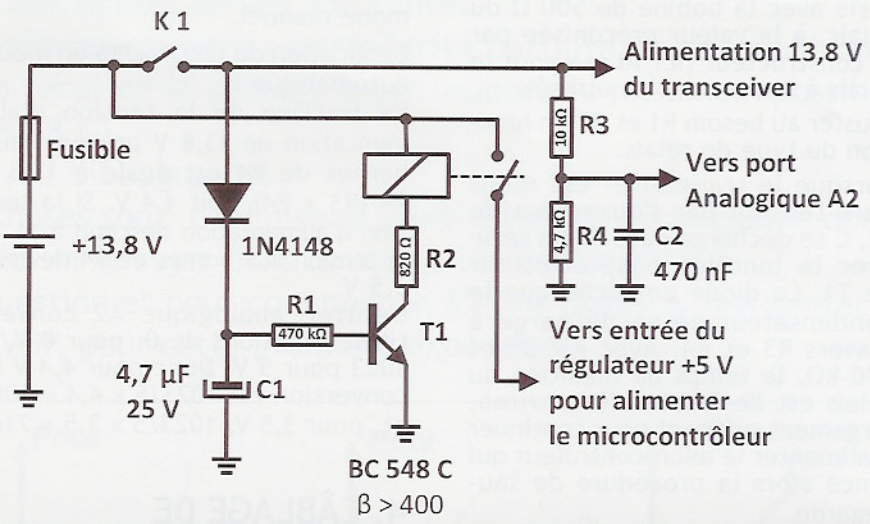

Figure 2.9 :

• Déclenchement de la sauvegarde des données :

Le microcontrôleur est informé de la mise hors tension par le diviseur résistif R3/R4 (sonde Marche/Arrêt) qui fournit, sur le port analogique A2, une fraction de la tension d'alimentation (de l'ordre de 4,4 V pour 13,8 V d'alimentation et de 3,5 V pour 11 V d'alimentation).

La sauvegarde est déclenchée lorsque le port analogique A2 passe sous le seuil prédéfini, mais modifiable, correspondant à une tension d'alimentation du transceiver inférieure à 11 V.

Lors de la mise hors tension par le bouton Marche/Arrêt, il faut donner le temps au microcontrôleur d'effectuer la sauvegarde avant que son alimentation ne soit coupée.

C'est le rôle du petit montage de la *figure 2-9*.
(Voir figure ci-dessus)

Lorsque le transceiver est éteint depuis au moins quelques secondes, K1 est ouvert, le condensateur C est déchargé et le transistor T1 bloqué. Le relais est donc ouvert.

À la mise sous tension du transceiver, K1 est fermé, C se charge quasiment instantanément à la

valeur de la tension d'alimentation du transceiver (13,8 V - le seuil de la diode). Le courant de base atteint alors une valeur suffisante limitée par R1 pour faire conduire le T1. Le relais colle. Le courant est limité par R2 en série avec la bobine de 500 Ω du relais, à la valeur préconisée par le constructeur (ici 10 mA pour le relais à lames souples utilisé).

Ajuster au besoin R1 et R2 en fonction du type de relais.

Lorsque le transceiver est remis hors tension par l'ouverture de K1, C se décharge dans R1 en série avec la jonction base-émetteur de T1. La diode empêche que le condensateur ne se décharge à travers R3 et R4. Avec 4,7 µF et 470 kΩ, le temps de maintien du relais est de 3 secondes environ, largement suffisant pour continuer à alimenter le microcontrôleur qui lance alors la procédure de sauvegarde.

Remarque : le programme de sauvegarde étant dans le programme principal, il est exécuté en boucle pour éviter d'utiliser une interruption. De ce fait, il y a un risque que plusieurs sauvegardes soient effectuées pendant la mise hors tension, et surtout, que la dernière sauvegarde soit interrompue ce qui pourrait entraîner une écriture erronée dans l'EEPROM. C'est pour cette raison qu'après la première sauvegarde effectuée, une temporisation de 3 secondes est activée jusqu'à ce que la tension du microcontrôleur soit effectivement coupée.

Remarque importante : lors de la conception avec le module ARDUINO, l'alimentation du microcontrôleur se fait via le connecteur USB et l'utilisation du relais de sauvegarde n'est pas possible puisque la tension de 13,8 V n'existe pas pour les essais.

Pour tester le montage avec le relais de sauvegarde, le câbler sur la platine d'essai et l'alimenter provisoirement avec une alimentation extérieure de 13,8 V. Attention toutefois de ne pas connecter le 13,8 V sur la ligne + 5 V de la platine d'essais...

A la place du régulateur +5 V, connecter une diode électroluminescente en série avec une résistance de 4,7 k Ω pour vérifier que le relais colle pendant 3 secondes environ. Une fois vérifié le fonctionnement, débrancher le 13,8 V.

Durant les essais qui suivent, le déclenchement de la sauvegarde est simulé sans avoir recours au montage ci-dessus, mais grâce au bouton poussoir 0 câblé sur la platine d'essais. La partie de programme « Mise hors tension et sauvegarde des données » est modifiée selon le tableau 2 ci-dessous :

Par défaut, le programme complet de la partie 2 IHM_Partie_2 est en mode manuel.

Explication du chiffre 716 en mode automatique :

La fraction de la tension d'alimentation de 13,8 V prélevée aux bornes de R4 est égale à 13,8 x R4/(R3 + R4), soit 4,4 V. Si la tension d'alimentation descend à 11 V, la tension aux bornes de R4 devient 3,5 V.

L'entrée analogique A2 convertit ces tensions de 0, pour 0 V, à 1023 pour 5 V. Donc pour 4,4 V la conversion est 1023/5 x 4,4 = 900, et, pour 3,5 V, 1023/5 x 3,5 = 716.

4. CÂBLAGE DE LA PARTIE 2

La figure 2-10 représente les composants câblés sur la platine d'essais. L'EEPROM est câblée à droite du module DDS.

L'utilisation des boutons Sélect. et Menu sera vue dans le prochain article.

Pour visualiser la commutation de bande et la commutation BLI/BLS, on peut insérer une résistance de 1 kΩ en série avec une diode électroluminescente (cathode à la masse) sur les ports D0 et D10.

5. CODES SOURCES

Les programmes présentés dans cet article sont disponibles sur le site du REF-Union, dans la rubrique « Compléments des articles techniques » :

- Encodeur optique : IHM_Partie_2_Encodeur.ino
- Encodeur et DDS : IHM_Partie_2_DDS_Encodeur.ino

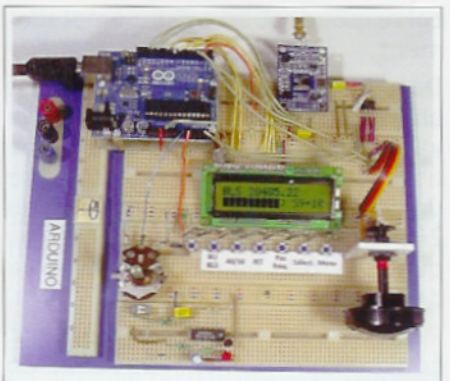

Figure 2-10 :

- Programme complet de la partie 2 : IHM_Partie_2.ino

CONCLUSION

Avec la synthèse en fréquence et l'encodeur optique, l'interface homme-machine commence à prendre tournure et peut d'ores et déjà être utilisée telle quelle pour piloter un récepteur simple à conversion directe. On peut maintenant afficher la fréquence, la régler avec un encodeur optique, changer de bande et de type de modulation, activer le RIT et sauvegarder les paramètres.

Dans le troisième article de cette série, nous aborderons la gestion de menus pour activer des fonctions supplémentaires, comme un scanneur, une horloge, une sonde de température et élargir les possibilités de cette interface homme-machine.

Mode manuel avec le bouton poussoir 0

```
// Mise hors tension et sauvegarde des données
// Via bouton 0 (provisoire)
   if (bouton_0 == true)
// En automatique via le port analogique A2
// tension_alim = analogRead (2);
// if (tension_alim < 716)
  {
   Instructions d'affichage et de sauvegarde
  }
```

Mode automatique

```
// Mise hors tension et sauvegarde des données
// Via bouton 0 (provisoire)
//  if (bouton_0 == true)
// En automatique via le port analogique A2
  tension_alim = analogRead (2) ;
  if (tension_alim < 716)
 {
  Instructions d'affichage et de sauvegarde
 }
```

Tableau 2