Philippe CORBIN
Kévin RAYMOND

GEII 2ème année
option EAS

La Télécommande Interactive

Contrôler son ordinateur à distance

Tuteurs:
M. Jean Noël MARTIN
M. Gilles SICARDI

Année 2006-2007

Remerciements

Nous voulons remercier
M. Jean Noël Martin et M. Gilles Sicardi
pour leurs nombreux conseils techniques et leurs réflexions pertinentes
qui nous ont permis de faire avancer le projet plus rapidement.
Microchip pour les nombreux circuits offerts qui nous ont permis de commencer plus tôt,
et divers forums électroniques pour leurs archives
dont abcelectronique et Futura-Sciences.

Sommaire

Listes des tableaux et des images

Les différentes abréviations utilisées :

ASK	Amplitude Shift Keying
AN	Application Note (Microchip)
EN	Enable
CDC	Communication Device Class
HF	Haute Fréquence
HID	Human Interface Devices
I/O	Input/Output
LCD	Liquid Crystal Display
MFC	Microsoft Foundation Class
NRZI	Non Retour à Zéros Inversé
PCB	Printed Circuit Board
PID	Product IDentification
R/W	Read/Write
RS	Register Select
USB	Universal Serial Bus
VID	Vendor IDentification

CCS	pour Custom Computer Services est un compilateur C professionnel pour PIC
C18	est le compilateur pour PIC18 distribué par MICROCHIP
EasyHID	est un générateur de code HID distribué par MECANIQUE
HIDmaker	est un générateur de code HID professionnel distribué par TRACE Systems
PIC	est le nom des microcontrôleurs distribués par MICROCHIP
Winamp	est un lecteur multimédia distribué par NULLSOFT

Introduction

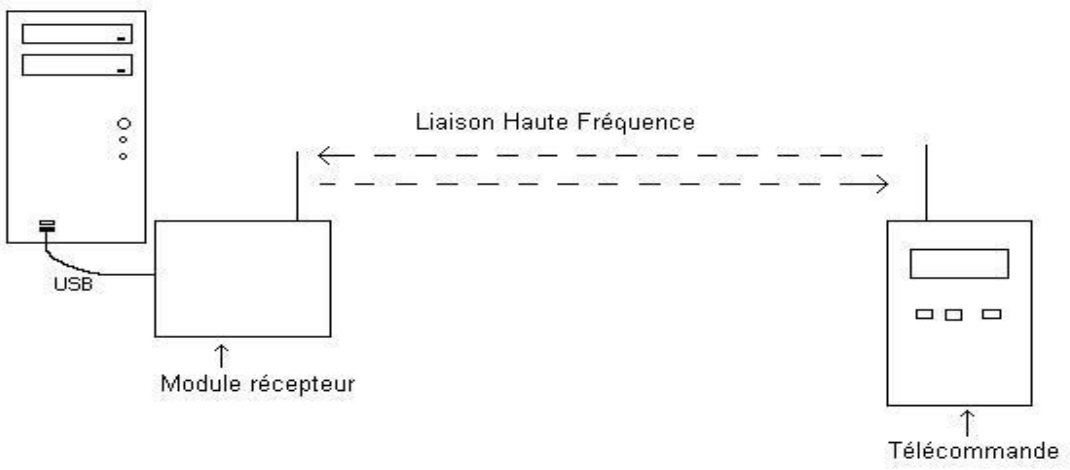
Dans ce monde de plus en plus modernisé, pouvoir commander son PC à distance peut devenir une nécessité.

Nous avons perçu ce manque dans le cadre des projets de deuxième année et nous avons décidé de nous lancer dans la conception d'une télécommande pour ordinateur. Une fois terminée, nous pourrons nous en servir dans notre quotidien pour diverses tâches. Celle-ci devra avoir une utilisation simplifiée tout en restant efficace. De plus, elle devra afficher différents états de l' ordinateur.

Pour la réaliser, nous nous intéresserons principalement à la liaison USB pour un dialogue simplifié avec le PC ainsi que la liaison haute fréquence (HF) pour transmettre les informations à la télécommande.

Nous pourrions améliorer notre projet en réduisant le volume occupé par les composants ainsi qu' en augmentant le potentiel de la télécommande ou en changeant de mode de communication.

Nous réaliserons notre projet en suivant le schéma ci-dessous :



Cahier des charges

Le projet est composé de deux parties, la télécommande et le récepteur (qui se branchera sur l' ordinateur). Nous devons réaliser ces deux modules avec une utilisation la plus simple possible. Pour cela, nous avons plusieurs éléments à notre disposition.

Pour commencer, le module qui se branchera sur l' ordinateur devra savoir utiliser la liaison USB et être capable d' émettre et de recevoir des informations par liaison HF. Ce module devra être reconnu comme périphérique HID pour une utilisation simplifiée. Le coeur de cette carte sera réalisé autour d' un PIC18F2550. Ce module ne devra pas être trop volumineux pour mieux s' insérer dans la vie quotidienne en perturbant le moins possible l' environnement de l' utilisateur.

Ensuite, la télécommande doit être la plus petite et la moins lourde possible pour un meilleur confort pour l' utilisateur. Ce module devra être capable d' envoyer et de recevoir des informations HF et de les afficher sur un écran LCD intégré à celle-ci. La télécommande sera munie de quelques boutons pour effectuer des actions à distance. La gestion de tous ces paramètres sera effectuée par un PIC16F877.

Le PC devra reconnaître les informations envoyées par la télécommande et agir en conséquence. Il devra aussi envoyer quelques informations à celle-ci régulièrement, comme la piste et l' album jouer dans un lecteur multimédia, la gestion des emails, etc. Le programme créé pour ces actions, devra être facile à installer, à utiliser et ne pas prendre trop de ressources.

Pour résumer le kit complet devra, être compact, simple d' utilisation pour un budget maximum de 70€.

La liaison Haute fréquence

1 / Principe

La télécommande devra pouvoir envoyer des données à l' ordinateur et vice versa. Nous avons choisi d' équiper nos modules avec la liaison Haute Fréquence, plus communément appelée HF. En effet, ce support de transfert nous permet de ne pas être forcément dans la même pièce que l' ordinateur à contrôler. On pourra recevoir des informations à une centaine de mètres en terrain vague ou à une vingtaine de mètres à travers les murs. C' est le gros avantage de cette transmission. Par exemple, la liaison par infrarouge oblige à avoir la télécommande et le récepteur dans la même pièce. Ce qui est un peu gênant si l' on veut s' éloigner de son ordinateur tout en surveillant l' arrivée de nouveaux emails par exemple. Mais ce système est un peu encombrant, car il faut aux émetteurs et récepteurs une antenne. On a donc quatre antennes en tout.

Pour recevoir ces données, puis en transmettre d' autres, nous avons implanté un pic capable de gérer l' USB, le 18F2550. Son rôle est de dialoguer correctement avec l' ordinateur et d' extraire les informations utiles de la longue chaîne de caractères. Après une brève remise en forme, les données sont traitées pour être mises sous la forme du protocole définit pour le transfert des données en HF. Ainsi, tant que l' on reçoit rigoureusement le même protocole, nous pouvons assurer la validité des données qui ont été transférées. De plus, ce système nous assure un minimum de sécurité dans nos chaînes d' information et évite les interférences avec d' autres émetteurs travaillant sur la même bande de fréquence.

Nos deux modules vont dialoguer en Half Duplex. Il s' agit d' une communication à double sens, mais non simultanée, contrairement au Full Duplex. Cette technique nous permettra d' éviter toutes interférences entre nos deux sources. Mais cela nous obligera à avoir un « maître » qui ordonnera la communication et un « esclave » qui répondra ou transmettra les informations.

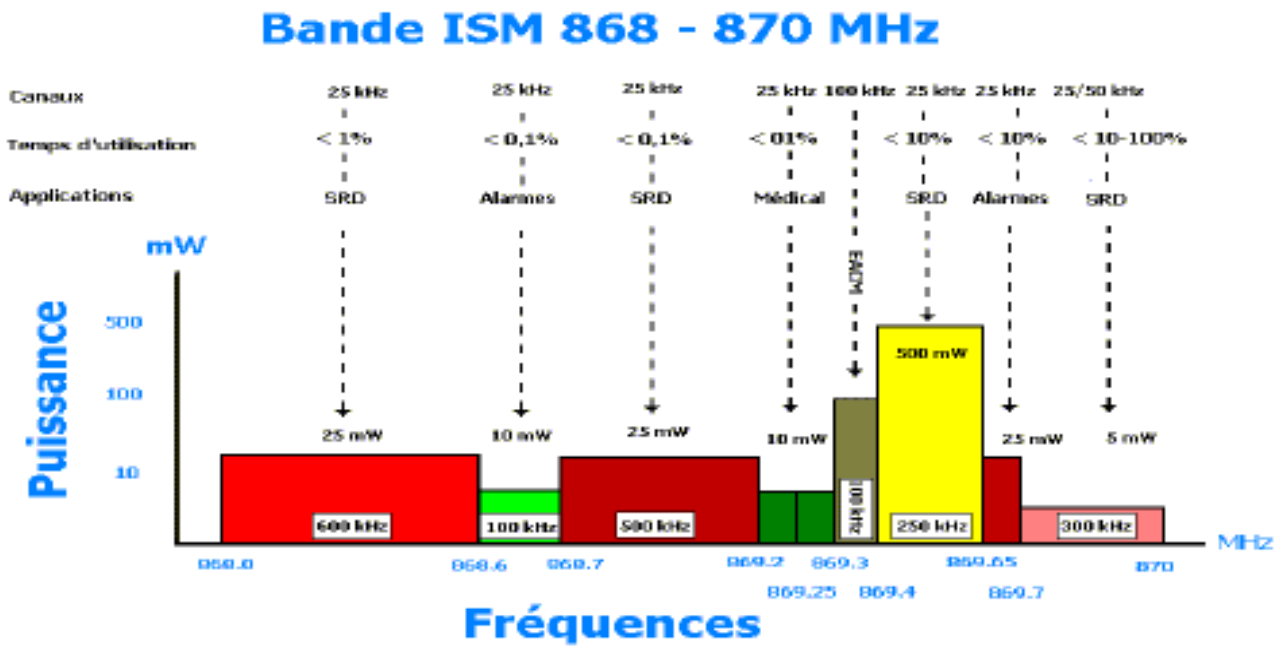
Les kits qui permettent à nos cartes de s' échanger des informations s' appellent des modules Aurels.

2 / Les modules Aurels

Après une recherche sur différents catalogues et forum, nous avons trouvé, un émetteur et un récepteur miniaturisés qui conviennent parfaitement à notre utilisation. Nous pouvons les voir sur les images suivantes. A gauche, il y a l' émetteur et à droite, le récepteur.

Ces produits sont livrés avec une documentation technique assez complète pour une prise en main facile du matériel. Comme on peut le voir sur les images, ces composants sont assez simples d' utilisation, visible au faible nombre de pattes. En effet, l' émetteur se contente d' être alimenté correctement et d' avoir une antenne pour fonctionner. Il ne reste plus qu' à lui envoyer les informations utiles sous forme de trame série pour qu' il les transmette. Le récepteur possède la même simplicité. Il dispose en plus après démodulation, d' une sortie analogique et une numérique pour s' adapter à tout type de montage. Comme nous utilisons la liaison série, nous utiliserons la sortie numérique. Ce type d' appareil utilise la technologie ASK ou Amplitude Shift Keying. On peut traduire littéralement par codage par changements d' amplitude ou plus simplement Modulation d' Amplitude.

Un petit rappel sur la législation pour l' utilisation des ondes Hautes fréquences ainsi que sur leur puissance en France. « La bande 433MHz ainsi que la bande 200MHz doivent être utilisées avec des modems radios agrées ne dépassant pas 10mW de puissance d'émission, pour être utilisable de plein droit et sans licence ni redevance annuelle. » Cette norme est effective depuis l' arrêté du 11 mars 1994. De plus, très récemment, un grand pas a été franchi, puisque une bande a été libérée (868 à 870 MHz), réservée aux transmissions de données, et segmentée par application :

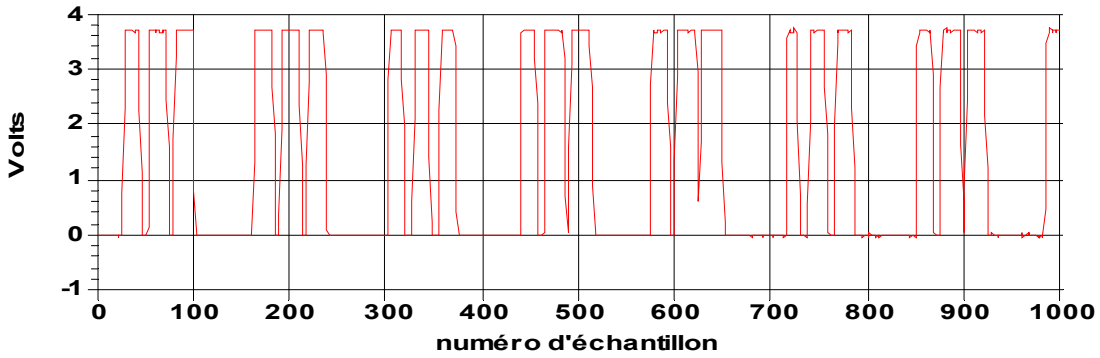


Les bandes 200MHz et 400MHz n' ont pas un découpage aussi strict. En effet, la fréquence à utiliser n' est pas déterminée par le type d' application, on peut prendre n' importe quelle valeur mais on est limité à 10mW en émission.

Le choix de l' une de ses plages a été fait lors de l' achat des modules Aurel. Ils étaient configurés en 433MHz. Une autre bande n' aurait rien changé, le montage aurait tout aussi bien marché.

Pour envoyer les données aux émetteurs et traiter celles reçues, nous utiliserons les entrées sorties câblées des pics mis en oeuvre dans le projet. La seule restriction que l' on ait avec ces modules, est qu' il faut envoyer des trames avec une vitesse minimum de 512bauds (soit 512 bit/s) et avec une vitesse maximum de 200Kbauds. Cela n' est pas extrêmement contraignant, mais comme par moment la transmission va s' arrêter (pour économiser les piles de la télécommande), il faudra alors envoyer plusieurs fois la même donnée avant de pouvoir recevoir une chaîne de caractères correctes. Il faudra envoyer des trames à une vitesse de 512 bauds pour que l' émetteur émette correctement.

Par exemple, en émettant le caractère 'A' en continue, on obtient la trame suivante en réception. Cette expérience a été réalisée sur une distance de vingt mètres et à travers trois murs. Le 'A' est codé en binaire par « 01000001 », on remarque que la liaison série nous ajoute les bits de start (un niveau bas : 0) et de stop (un niveau haut : 1). Le bit de poids faible est envoyé en premier.



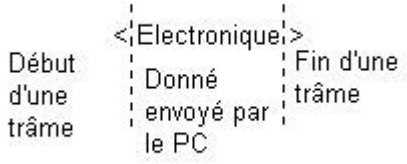
On remarque que le signal est bien reçu mais on voit quelques erreurs qui apparaissent. On peut le voir à 600ms où il y aura un bit faux après traitement du signal par le pic. On peut corriger cette erreur grâce à un trigger de schmitt. Celui-ci bascule a 2,5V, ce qui permettra de récupérer quelques bits erronés.

Nous pouvons alors récupérer notre trame série envoyée quelques temps plus tôt par l' émetteur. Il suffira juste de sortir les informations utiles de notre protocole.

3 / Le protocole

La plage de fréquence que nous utilisons est un domaine libre avec quelques conditions. Il y a donc beaucoup d' applications, notamment de grands industriels qui utilisent cette plage de fréquence. Par exemple, on peut trouver la commande de store électrique, de portail, velux, divers projets de deuxième année, etc. Pour pouvoir envoyer nos informations et les recevoir, il faudra donc les personnaliser pour les reconnaître parmi ce flot d'ondes.

Nous avons choisi de commencer notre chaîne de caractères par un « < » et de terminer celle-ci par un « > » . Notre chaîne de caractères ne devra pas dépasser 16 caractères, sinon, elle sera automatiquement découpée pour atteindre cette valeur et les données en trop seront perdues. Un exemple de transmission, on veut envoyer le mot « Electronique » . Avant d' être envoyé, il sera mis sous cette forme :



Il sera ensuite transmis à l' émetteur, et sera reçu par le récepteur. Finalement, après démodulation, le mot sera envoyé vers le pic.

Comme nous l' avons vu précédemment, par souci d' économie d' énergie, nous utiliserons l' émetteur que quand nous aurons besoin de transférer des données. Donc, dans notre échange, nous devons définir un « maître », qui ordonnera le transfert et un « esclave » qui se contentera de lui répondre. En plus du protocole pour les caractères, nous devons définir un ordre d' émission. Nous avons donc choisi le module USB pour être le « maître ». Il enverra des données dès qu' il y aura eu un changement d' état du PC. Quand la télécommande aura reçu la bonne trame, il lui dira qu' il peut cesser d' émettre. Ainsi, l' affichage est à jour et nous utilisons au minimum les batteries.

La télécommande

1 / Gestion de l' afficheur LCD

Tout d' abord il faut bien choisir son afficheur LCD. On le choisi bien entendu par son nombre de ligne et de caractère, mais plus particulièrement par son contrôleur. Le plus utilisé et donc le plus facile à mettre en oeuvre (point de vue source sur internet) est le contrôleur **HD44780 d' Hitachi**. C' est lui qui travaille pour nous : on lui envoie des instructions simplistes, et il les transmet à l' afficheur.

Le principe de contrôle est défini dans **AfficheurLCD.h**. On choisit de le contrôler sur quatre bits de données (au lieu de huit) pour alléger le routage du PCB. Il faut donc faire un masquage des donnée à afficher qui sont en *char* et donc sur huit bits. Attention, certain LCD se contrôlent en huit bits par les bits D0 à D3, et d' autres par D4 à D7 (comme dans notre application). Il y a également trois bits de contrôle (EN, R/W, RS) qui permettent de gérer la position du curseur, l' affichage de celui-ci, l' effacement du LCD, etc.
Il y a aussi la possibilité de tester le busy flag du LCD en D7, mais là n' était pas notre priorité. Pour tester le flag, il faut positionner la sortie D7 en entrée sur le PIC, et envoyer en contrôle RS à 0 et RW à 1. Comme notre afficheur provient d' une récupération, nous n' avions pas la documentation technique permettant de connaître les temps de pauses nécessaires au bon enregistrement des instructions avec le LCD. Nous les avons donc volontairement augmentés.

Voici le brochage des afficheurs contrôlé par le HD44780:

Broche	Nom	Niveau	Fonction
1	Vss	-	Masse
2	Vdd	-	Alimentation positive +5V
3	Vo	0-5V	Cette tension permet, en la faisant varier entre 0 et +5V, le réglage du contraste de l' afficheur.
4	RS	TTL	Sélection du registre (Register Select) Grâce à cette broche, l' afficheur est capable de faire la différence entre une commande et une donnée. Un niveau bas indique une commande et un niveau haut indique une donnée.
5	R/W	TTL	Lecture ou écriture (Read/Write) L : Écriture H : Lecture
6	E	TTL	Entrée de validation (Enable) active sur front descendant. Le niveau haut

			doit être maintenu pendant au moins 450 ns à l' état haut.
7	D0	TTL	Bus de données bidirectionnel 3 états (haute impédance lorsque E=0) Pour les afficheurs acceptant les données en 4 bits, D0-D3 ou D4-D7
8	D1	TTL	
9	D2	TTL	
10	D3	TTL	
11	D4	TTL	
12	D5	TTL	
13	D6	TTL	
14	D7	TTL	
15	A	-	Anode rétroéclairage (+5V)
16	K	-	Cathode rétroéclairage (masse)

Le jeu de commandes standard:

RS	RW	OPERATION
0	0	écriture dans le registre d' instruction et exécution commande
0	1	lecture busy flag d7 et compteur d' adresse d0-d6
1	0	écriture registre de données
1	1	lecture registre de données

instructions, avec RS et RW à 0 : (x=indifférent)

d7-------d0	
00000001	Efface l' écran et ramène le curseur au début (adresse 00) - 1,64 ms
0000001x	Ramène le curseur à l' origine (en haut à gauche), sans effacer - 1.64 ms
000001ab	Définit le sens de déplacement du curseur après l' apparition d' un caractère (vers la gauche si a=1, vers la droite si a=0) et si l' affichage accompagne le curseur dans son déplacement ou non (b). - 40 µs
00001cde	activation affichage (c), du curseur (d), du clignotement (e) - 40 µs
0001fgxx	décalage le curseur (f=1) ou l' affichage (f=0) d' une position vers la gauche g=1) ou vers la droite (g=0) sans modifier la DD RAM- 40 µs
001hijxx	Définit l' interface (h=0 : mode 4 bits, h=1 mode 8 bits), le nombre de lignes (i=0 pour 1 ligne, i=1 pour 2 ou 4 lignes) et la taille des fontes (j=0 pour des caractères 5x7, j=1 pour des caractères 5x10). - 40 µs

Une fois que l' écran fonctionne correctement, celui-ci nous est d' une aide précieuse. En effet, il va nous permettre de voir et de debater le programme qui gère la transmission Haute Fréquence.

2 / Transmission Haute Fréquence

Nous avons vu que nos modules s' occupent de transmettre les données envoyées, mais après, il faut traiter la chaîne de caractères.

Pour réaliser le rajout et la suppression des caractères définis dans le protocole, nous utilisons les fonctions du compilateur CCS. Il nous apporte une aide avec les fonctions comme « strcpy() » qui permet de copier un tableau dans un deuxième ou « strlen() » qui nous permet de connaître la taille d' un tableau. Ce programme gère aussi la liaison RS232. Il a en mémoire quelques fonctions utiles comme « puts() » qui permet d' envoyer une chaîne de caractères et « gets() » qui permet de récupérer les informations passant sur la liaison série. De plus, le logiciel gère la configuration de ce protocole. Nous n' avons donc pas besoin de rechercher dans la documentation technique du microcontrôleur utilisée pour savoir exactement quels registres sont à configurer. Pour bien comprendre comment utiliser ses fonctions, il ne faut pas hésiter à abuser de l' aide.

Toutes ces fonctions nous sont d' une aide précieuse et évitent ainsi de les créer. Mais il reste encore quelques points à gérer. En effet, nous recevons des données, mais nous ne savons pas quand, la réception asynchrone. Nous ne pouvons pas risquer de bloquer le pic en attendant l' arrivée d' une chaîne de caractères. Pour cela, nous utilisons les interruptions. Dans celle-ci nous devons tester si notre chaîne correspond à notre protocole et l' extraire si elle est bonne. De plus, il faut que tout ceci se passe le plus rapidement possible, sinon, il se peut que le pic se perde. Nous avons passé beaucoup de temps à tester et à programmer ces quelques lignes. En voici le code avec les explications :

```
#INT_RDA                                    //Fonction  d' interruption :  pas de no_clear => acquittement automatique
void RecepIT(void)
{
int8 val =0;                                // déclaration d' une variables de type entier sur 8 bits
int1 debut = 0;                             ///déclaration d' une variable de type entier sur 1 bit


        string[it] = RCREG;                 // remplissage du tableau avec la valeur contenu dans le buffer de la liaison série
        if((string[it] == '>')  &&  (debut ==1)) //test de présence d' un caractère de fin de chaîne de notre protocole et que l' on a déjà
        {                                   reçu le premier caractère
                val = it;                   //On mémorise la valeur de l' index du tableau
                string[val] = '\0';         //Remplacement du signe supérieur par caractère de fin de chaîne => on affichera
                                            les caractères du tableau jusqu' à cette case
                strcpy(strval , string);      //copy du tableau de remplissage dans un autre tableau => il contient que la donnée
                etat = RECEPTION;           //On passe dans l' état réception du graphe d' états
        }
```

```
        if(string[it] == '<')                   //test de présence du caractère de début de chaîne
        {
                it=0;                           //on remet l' index du tableau à 0
                debut=1;                        //mémorisation de la réception du début de chaine
        }
        else                                    //sinon, on incrémente l' index
        {
                it++;
                if(it >= 16)                    //si l' index est a 16 (atteinte de la valeur limite de stockage) on remet à zéro
                {
                        it = 0;
                }
        }
}
```

Avec ces quelques lignes, on arrive à éliminer notre protocole, et à stocker la chaîne de caractères dans un tableau où le premier caractère est dans la première case de celui-ci. Ces données seront traitées dans la fonction principale. Mais si les caractères reçues ne correspondent pas au protocole alors on remplit simplement le tableau et les données contenues dans celui-ci ne passent pas dans la fonction principal.

Pour rajouter les caractères de notre protocole au début, il suffit de faire une petite boucle pour décaler le tableau puis de mettre nos caractères dans les espaces ainsi libérées. On peut le voir facilement sur cet exemple :

```
void CreatDataProto(char cTab)                  //le tableau modifié est déclaré en variable globale
{
        int8 IndexMax = 0, ibcl = 0;            //déclaration de 2 variables

        IndexMax = strlen(cTab);                //calcul de la longueur de la chaîne de caractère
        cTabRet[0] = ' ';
        for(ibcl = 0 ; ibcl <= IndexMax ; ibcl++)   //décalage du tableau pour mettre le premier caractère
        {
                cTabRet[(ibcl+1)] = cTab[ibcl];
        }
        cTabRet[(IndexMax+1)] = '>';            //rajout du dernier caractère
        cTabRet[0] = '<';                       //rajout du premier caractère
}
```

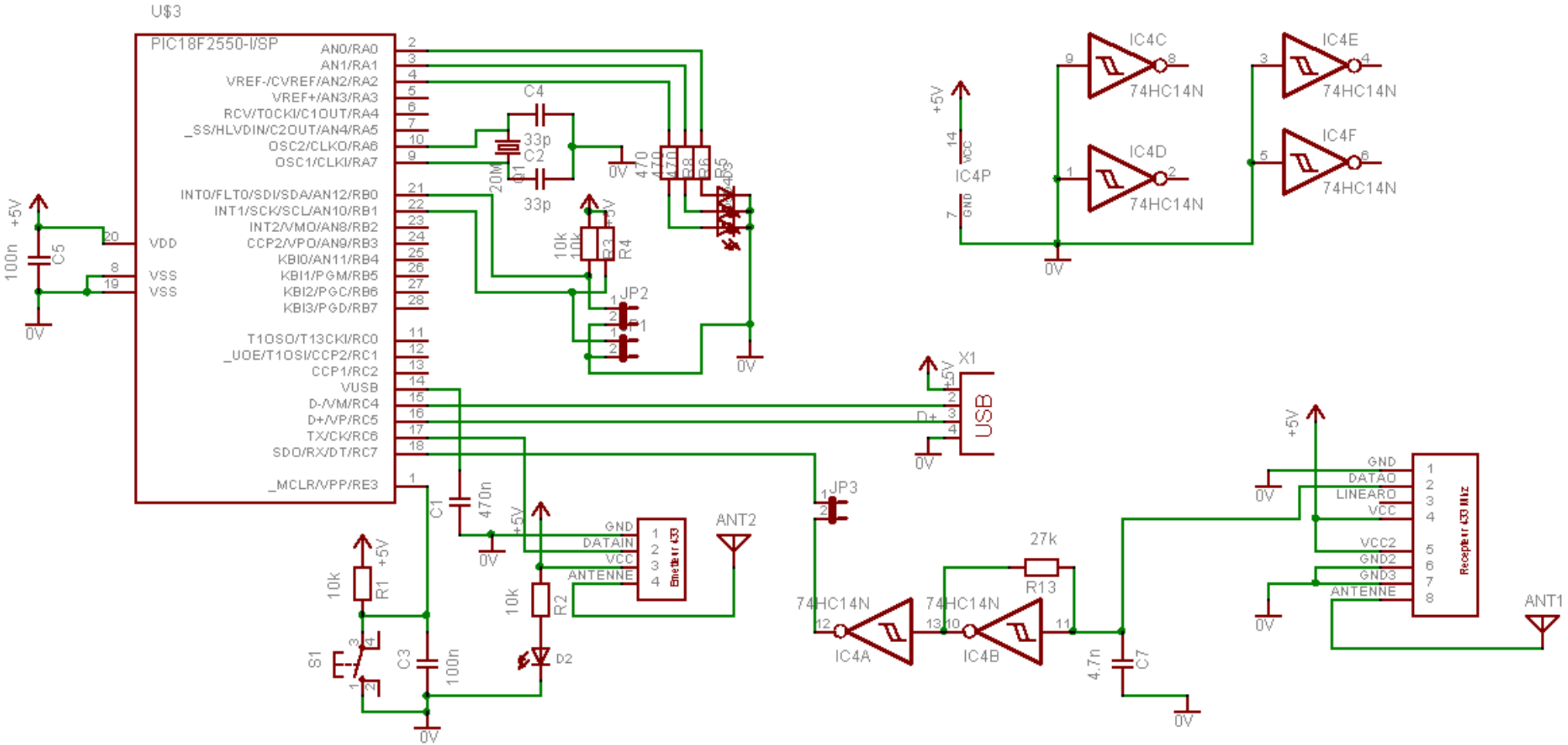
Avec toutes ces précisions techniques et informatiques, nous pouvons élaborer une carte pour la télécommande où l' on pourra effectuer quelques tests.

3 / Le Prototype

Pour nos différents tests et pour rester fidèle au cahier des charges, nous aurons besoin de plusieurs fonctionnalités. Par exemple, nous avons besoin de plusieurs boutons poussoirs et une LED. Le noyau de cette carte est le pic 16F877. Il possède de grandes capacités mais est très volumineux comme composants. C' est pour cela, que nous l' avons implanté en PLCC. Il est plus petit ainsi et la carte est d' autant réduite.

Le schéma de la carte est présenté page suivante. On peut remarquer la présence d' un trigger de Schmitt TTL qui est là pour mettre en forme le signal reçu du récepteur. Nous avons besoin de deux triggers car ils sont inverseurs. Mais ceux-ci sont intégrés à un composant, le 74HC14. On trouve les éléments nécessaires au fonctionnement de la télécommande. Il y a un émetteur et un récepteur HF ainsi que des boutons poussoirs et un écran LCD. La carte est alimentée avec une tension supérieure à 7V, par exemple avec une pile 9V. Cette tension d' entrée est régulée par le LM7805 pour alimenter correctement le PIC. Pour faire démarrer le PIC, il faut mettre un passe bas sur l' entrée MCLR pour éliminer les hautes fréquences dues au changement brusque de tension lors de la mise sous tension.

Schéma de la carte 16F877



La communication via l' USB

1 / Introduction à l' USB et au HID

L' Universal Serial Bus est apparut il y a quelques temps pour normaliser la transmission avec divers périphériques tels que les imprimantes, les appareils photos numériques, les clefs USB etc.

Contrairement à la liaison RS232 et des interfaces sérielles similaires où le format des données envoyées n' est pas défini, l' USB est composé de plusieurs couches de protocoles.

Chaque transaction USB consiste d' un

- Paquet Jeton (Token) (en tête définissant ce qu' il attend par la suite)
- Paquet DATA optionnel (contenant la "charge utile " (payload))
- Paquet d' Etat (utilisé pour valider les transactions et pour fournir des moyens de corrections d' erreurs).

Sur le bus USB, c' est l' hôte qui gère la transmission. Il s' agit de l' ordinateur. Il envoie le premier paquet avec le jeton au nouveau périphérique détecté pour qu' il puisse s' identifier et ainsi créer la connexion. Pour les communications entre périphérique (sans passer par un ordinateur) il n' y a pas d' hôte. Il faut donc travailler en OTG (On The Go).

Les diverses normes amènent à trois vitesses principales:

- Vitesse Haute - High Speed – 480Mbits/s
- Vitesse Pleine - Full Speed – 12Mbits/s
- Vitesse Basse - Low Speed – 1,5Mbits/s

Pour les personnes intéressées, la spécification technique de l' USB 2.0 ne contient que 650 pages. Nous ne traiterons donc ici que des possibilités restreintes à l' utilisation des microcontrôleurs PIC. Nous retiendrons quatre principales solutions :

- Classe HID : vitesse limitée à 64ko/s, quartz de 20Mhz, aucun pilote nécessaire
- Emulation port COM (CDC) : voir AN956 , sans pilote
- USB low-speed : avec un quartz de 20Mhz, nécessite un pilote
- USB full-speed : avec un quartz de 48Mhz, nécessite un pilote

L' émulation du port COM (port série, protocole RS232) est assez facile à mettre en oeuvre: elle permet la compatibilité avec d' anciennes applications PC. Surtout utilisée dans le domaine industriel pour ne pas devoir développer d' autres applications devant gérer l' USB, quand les périphériques migrent du RS232 à l' USB.

Si une plus grande vitesse de transfert est nécessaire, et donc un plus grand débit, cela oblige à utiliser la Pleine ou Haute vitesse. Ainsi il faudra développer le pilote nécessaire à l' application. Au vu des sources sur internet, ce n' est pas chose facile, surtout pour atteindre 12Mb/s. l' USB Haute vitesse (480Mbits/s) arrive à peine sur le

marché des PIC.

La solution d' une interface HID a donc été retenue dans notre projet pour sa relative facilitée de mise en oeuvre, la vitesse de transmission étant amplement suffisante pour une télécommande. Le protocole HID (Human Interface Device) est de plus en plus utilisé puisque c' est la manière la plus simple d' utiliser le port USB. Sa seule limitation est la vitesse de transmission. Cette solution est utilisée pour les claviers, les souris, les manettes de jeux vidéo, etc. Compatible Windows 98 et supérieur.

La vitesse de transmission étant limitée à 64ko/s, si l' ordinateur envoie trop d' informations à la carte de telle manière que celle-ci soit saturée, Windows les placera dans un buffer en attendant de les traiter.

2 / Visual C++ et l' USB

Avant tout, il faut savoir que le principe de pilote n' est pas utilisé pour dialoguer avec les périphériques HID. En effet, l' application Windows se servira d' une dll se qui allège énormément les développeurs. Surtout qu' il existe des programmes permettant de les générer.

Voici les principaux outils permettant de générer cette dll et un début de code :

- EasyHID USB Wizard (version gratuite)
- HIDmaker FS (version commerciale)

Microchip met également à notre disposition un squelette (voir HID Setup.EXE) avec la dll associée. Ceci est moins pratique puisque l' on a pas autant de possibilités qu' avec les précédents programmes. Il s' agit en fait d' un exemple exploitable, mais pas autant personnalisé que ce que l' on pourrait faire avec EasyHID.

Le plus simple est la solution EasyHID qui permet entre autre de générer le squelette pour les pics 18F2550 et 18F4550 (malheureusement en PicBasic) et le squelette de l' application PC (en Delphi, Basic ou C++). EasyHID est un programme élaboré par mécanique (http://www.mecanique.co.uk/) et distribué gratuitement pour une utilisation non commerciale.

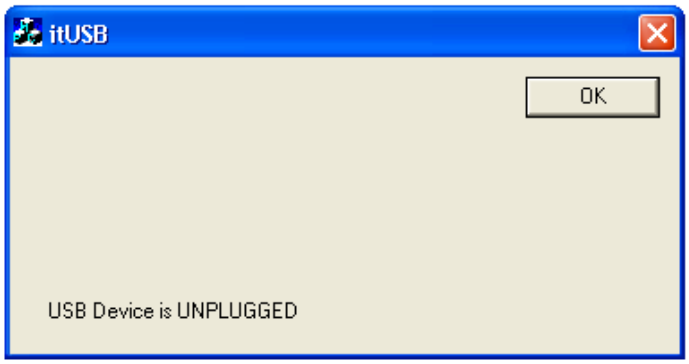
HIDmaker créer le squelette pour C18 (compilateur de Microchip pour les PIC18) côté PIC et Delphi, Basic ou C++ côté PC. Ce logiciel est payant. De ce fait il a été rapidement écarté des solutions envisageables.

C' est donc la solution EasyHID qui a été choisie pour sa relative simplicité et sa plus grande portabilité. En effet, la solution de Microchip nécessite d' installer plus de programmes pour développer et génère également un plus lourd code (mois optimisé par la dll). Programmant en C avec CCS, nous ne nous servirons pas de la source générée par EasyHID pour le « Hardware » (partie carte électronique), mais seulement pour le « Software » (parte application Windows) dont le squelette est écrit en C++.

Mieux vaut partir sur de bonnes bases pour débuter. Il vaut donc mieux commencer par réaliser la carte électronique qui s' identifie correctement à l' hôte. Se reporter à la partie « 3 / PIC18F2550 avec CCS et l' USB » pour la réaliser.

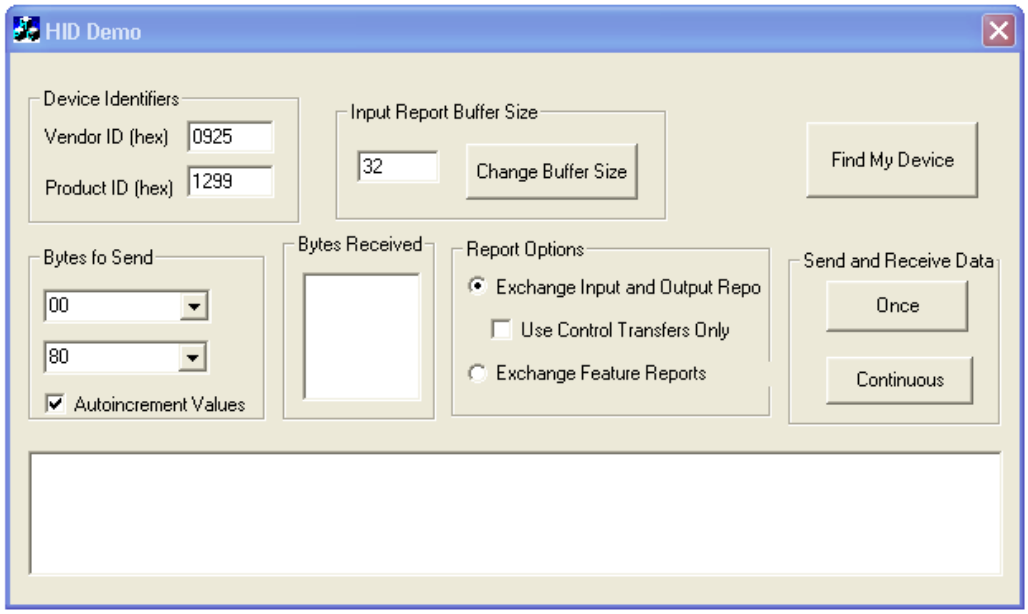
Une fois que la partie Hardware est correcte, on peut commencer à programmer côté Windows avec Visual C++. Le point de départ étant la source générée par EasyHID, il faudra implémenter les parties importantes au fur et à mesure que la partie visuelle est modifiée. En effet, la source est générée en MFC ce qui permet de créer facilement une interface graphique conviviale.

Voici ce qui est généré par EasyHID :



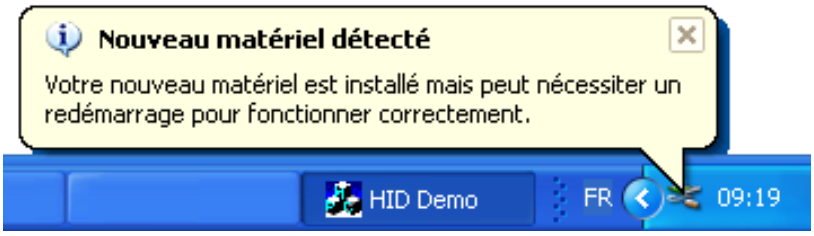
Pour créer rapidement une application assez poussée, reportez-vous à l' exemple « usbhidio » (voir http://www.Lvr.com) qui permet de dialoguer par l' USB en HID. La seule complexité est qu' il faille adapter la source fournit pour la dll de EasyHID. En effet, le programme usbhidio est basé sur le « Firmware » de Microchip et est également codé en MFC. Néanmoins, c' est un bon exemple pour voir à quoi une source USB avec Visual C++ peut ressembler. Hidtest.exe peut également est pratique pour détecteur les erreur d' identification.

Voici à quoi ressemble le programme usbhidioc.exe :



On remarque que ce programme est facilement adaptable à d'autres périphériques HID. en effet, il est possible de spécifier le Vendor Identification et Product Identification. Ainsi que la taille des données à transmettre. C' est pourquoi il est indispensable pour tester la carte électronique!

Voici ce que cela donne avec notre carte à la fin de l'installation :



Une fois la que la communication fonctionne, il est possible de compliquer le programme indéfiniment ! Attention, il est fréquent que les périphériques USB fonctionnent mieux sur un port USB 2.0. L' identification s' effectue plus proprement. En effet, Windows effectue une demande à l' utilisateur pour installer le périphérique automatiquement ou en spécifiant le pilote. Pour le HID, choisir la facilitée.

Maintenant que les connaissances théoriques sont posées, il n' y a plus qu' à passer à la pratique.

3 / PIC18F2550 avec CCS et l' USB

l' USB est un bus série. Il utilise quatre fils isolés dont deux sont l' alimentation (+5V et GND). Les deux restants forment une paire torsadée qui véhicule les signaux de données différentielles. Il utilise un schéma d' encodage NRZI (Non Retour à Zéro inversé) pour envoyer des données commencent par un champ *Sync*.

Tous les paquets doivent commençer avec un champ Sync. Le champ fait de 8 bits de long pour la basse et pleine vitesse ou 32 bits pour la haute vitesse est utilisé pour synchroniser l' horloge du récepteur avec celle de l' émetteur / récepteur. Les 2 derniers bits indiquent l' endroit ou le champ Paquet Identification commence.

Numéro de broches	Couleurs des câbles	Fonction
1	Rouge	V_{BUS} (5 volts)
2	Blanc	D-
3	Vert	D+
4	Noir	Masse

Tous périphérique USB doit s' identifier à l' hôte lors de sa connexion. L' ordinateur envoie une requête d' énumération à la détection d' une connexion de périphérique. Pour ce faire il test les impédances des lignes de données D+ et D-.

Une énumération sous Windows ordinaire implique les étapes suivantes :

- L' hôte ou Hub détecte la connexion d' un nouvel appareil via les résistances de rappel de l' appareil reliées sur les 2 fils de données (internes au PIC). L' hôte attend au moins 100ms, le temps que la prise soit insérée complètement et que l' alimentation de l' appareil soit stabilisée.
- L' hôte émet un " reset " mettant l' appareil dans l' état par défaut. L' appareil peut maintenant répondre à l' adresse zéro par défaut.
- L' hôte (sous MS Windows) demande les 64 premiers octets du descripteur d' appareil.
- Après avoir reçu les 8 premiers octets du descripteur d' appareil, l' hôte émet immédiatement un autre reset sur le bus.
- L' hôte émet maintenant une commande *SetAdress*, mettant l' appareil dans l' état
  adressable.
- L' hôte demande la totalité des 18 octets du descripteur d' appareil.
- Puis il demande les 9 octets du descripteur de configuration pour déterminer la taille totale.
- L' hôte demande les 255 octets du descripteur de configuration.
- L' hôte demande l' un des descripteurs de chaînes s' ils étaient indiqués.

A la fin de la dernière étape, Windows demandera un pilote pour le périphérique détecté. Il est alors courant de le voir redemander tous les descripteurs avant d' émettre une requête *SetConfiguration*.
Le processus d' énumération ci-dessus est courant dans Windows 2000, Windows XP et Windows 98 SE.

Généralement quand il y a un problème avec le descripteur ou sur la façon dont il est envoyé, l' hôte tentera de le lire trois fois avec de longues pauses entre les requêtes. Après la troisième tentative, l' hôte abandonne signalant une erreur au niveau de l' appareil.

Il existe plusieurs possibilités de dialogue par PIC avec l' USB. Dont celles-ci:

- Pic18Fx5xx conçu pour la liaison USB
- Pic16F877 dialoguant par I2C avec un contrôleur USB Full Speed (USBN9603 de National Semi-conducteur
- Pic16C765 l'un des premiers PIC USB (n'est plus actuel)

Voici le code tiré d' un exemple CCS (ex_usb_mouse.c) :

```
//set to 1 to use a PIC's internal USB Peripheral
//set to 0 to use a National USBN960x peripheral
#define __USB_PIC_PERIF__ 1

#if __USB_PIC_PERIF__
 #DEFINE LED1  PIN_A5
```

```
 #if defined(__PCM__)
   #include <16C765.h>
   #device *=16
   #fuses HS,NOWDT,NOPROTECT
   #use delay(clock=24000000)
 #elif defined(__PCH__)
   #include <18F4550.h>
   #fuses HSPLL,NOWDT,NOPROTECT,NOLVP,NODEBUG,USBDIV,PLL5,CPUDIV1,VREGEN
   #use delay(clock=48000000)
 #endif
#else //use the National USBN960x peripheral
 #DEFINE LED1  PIN_B3
 #if defined(__PCM__)
  #include <16F877A.h>
  #device *=16
  #fuses HS,NOWDT,NOPROTECT,NOLVP
 #elif defined(__PCH__)
  #include <18F452.h>
  #fuses HS,NOWDT,NOPROTECT,NOLVP
 #endif
 #use delay(clock=20000000)

#endif   //endif check to see which peripheral to use
```

La difficulté étant d' adapter cette source pour le 18F2550, remplacer par le code ci-dessous. La difficulté est d'adapter la vitesse de cadence du PIC.

```
//set to 1 to use a PIC with an internal USB Peripheral
//set to 0 to use a National USBN960x peripheral
#define __USB_PIC_PERIF__ 1

  #include <18F2550.h>

  //SLOW SPEED
  #fuses HSPLL,NOWDT,NOPROTECT,NOLVP,NODEBUG,USBDIV,PLL5,CPUDIV3,VREGEN
  #use delay(clock=20000000)
  #define USB_USE_FULL_SPEED  FALSE
```

A la suite vous devriez trouver ce code, à défaut modifier en conséquence :

```
////////////////////////////////////////////////////////////////////////////
//
// CCS Library dynamic defines.  For dynamic configuration of the CCS Library
// for your application several defines need to be made.  See the comments
// at usb.h for more information
//
////////////////////////////////////////////////////////////////////////////
#DEFINE USB_HID_DEVICE  TRUE  //Tells the CCS PIC USB firmware
                        //to include HID handling code.

//turn on EP1 for IN interrupt transfers.  (IN = PIC -> PC)
#define USB_EP1_TX_ENABLE  USB_ENABLE_INTERRUPT
#define USB_EP1_TX_SIZE 8     //max packet size of this endpoint

////////////////////////////////////////////////////////////////////////////
//
// Include the CCS USB Libraries.  See the comments at the top of these
// files for more information
//
////////////////////////////////////////////////////////////////////////////

#include <pic18_usb.h>   //Microchip PIC18Fxx5x hardware layer for usb.c
```

```
#include <usb_desc_mouse.h>    //USB Configuration and Device descriptors for this UBS device
#include <usb.c>        //handles USB setup tokens and get descriptor reports
```

Les includes sont utiles pour compiler la source de l'exemple d' une souris HID.

Avant de dialoguer avec l' hôte, le périphérique doit s' identifier. Tout périphérique USB a un code VID&PID unique pouvant l' identifier. Le VID est le Vendor IDentification (pour Microchip VID = 0x04D8) et PID est le Product IDentification. A la connexion, le périphérique s' identifie et spécifie le protocole (volumes des paquets transmis, le débit, le courant maximum à débiter, etc.).
Toute la description du périphérique se trouve dans **usb_decr_hid.h** .

La vitesse de la communication ne dépend que de deux paramètres, la taille du paquet envoyé (maximum 64 octets) et la période séparant chaque demande du pic (« pooling » minimum : une milliseconde). On aura donc 64 ko/s dans les meilleures conditions.

Au cours d' une énumération, l' hôte demande la description du périphérique. Le schéma représenté ci-dessous montre le principe de configuration du périphérique pour créer l' interface. Suivant cet ordre :

1. Get_Device_Descriptor
2. Get_Configuration_Descriptor
3. Get_Report_Descriptor
4. Get_String_Descriptor

Une interface peut avoir plusieurs terminaisons (« endpoint »). A noter que CCS ne sait gérer qu' une seule terminaison. Un endpoint est un tampon où les données attendent d' être transmises sur le bus, ou réceptionnées par le PIC. Plusieurs endpoints permettent plusieurs connexions et donc un dialogue plus aisé, mieux construit. La réception est gérée par l' endpoint0 IN et l' émission par l' endpoint0 OUT.
Le schéma présenté ci-dessous nous montre les diverses étapes pour la création de l' interface.

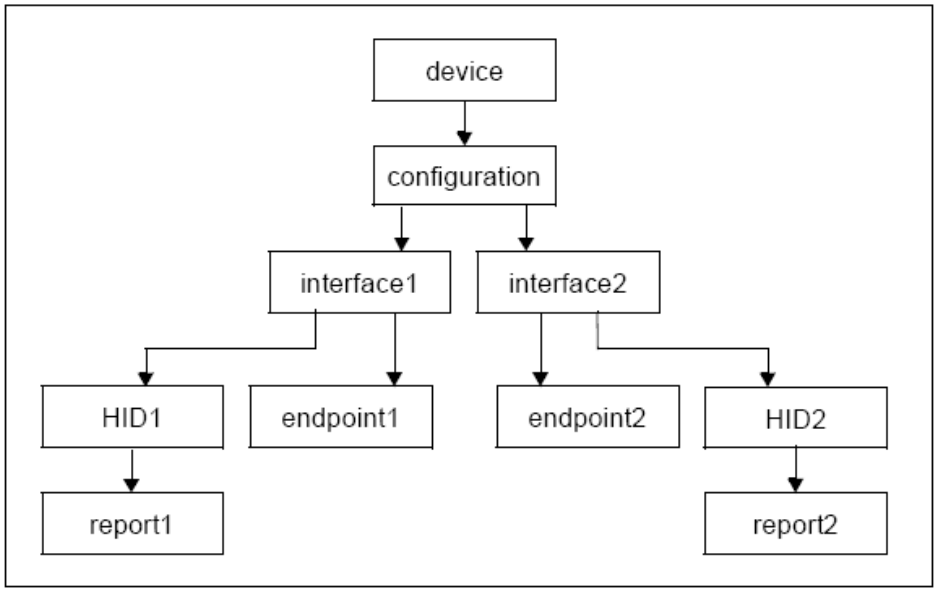
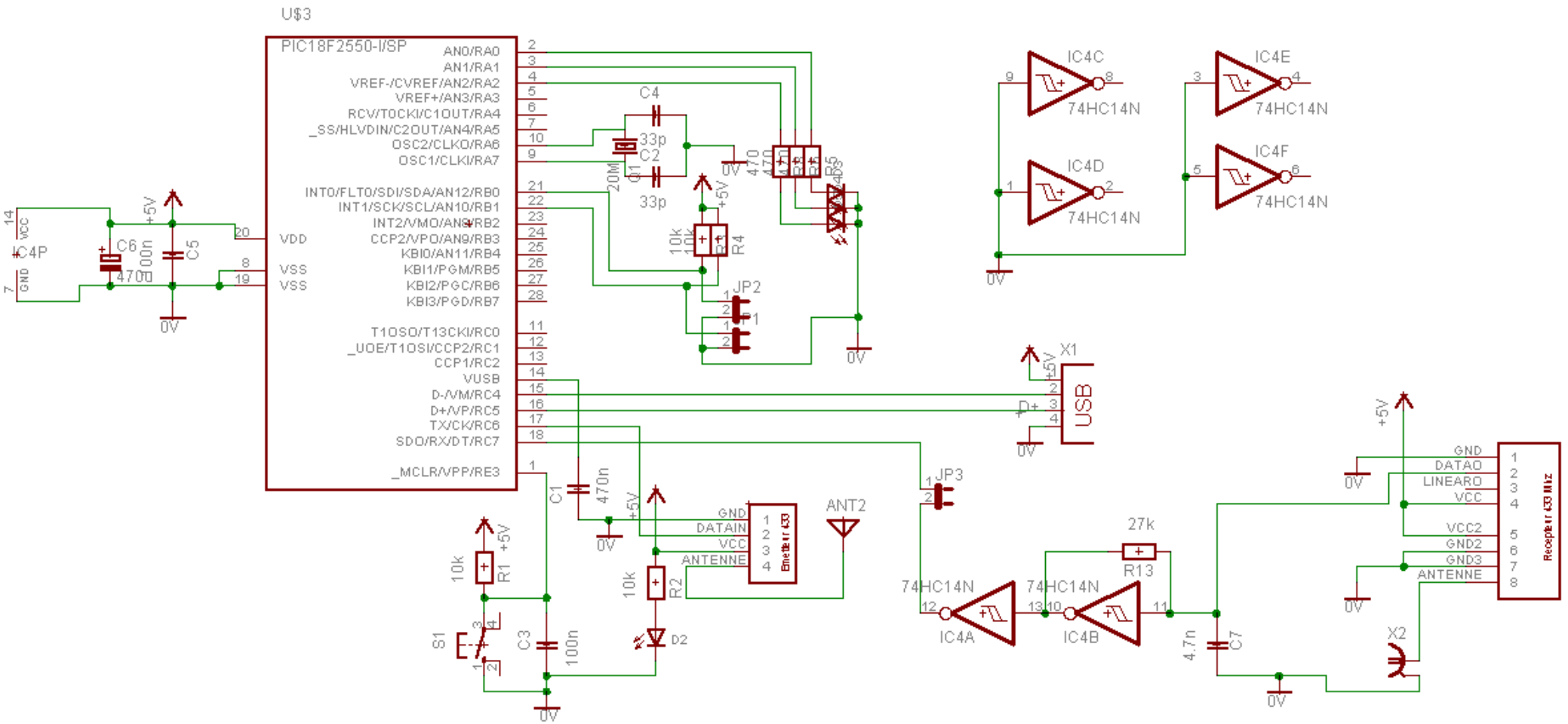


Schéma de la carte 18F2550



Description:

On retrouve sur cette carte les composants permettant de dialoguer par HF, comme dans la carte du PIC16F877. Pour l' USB, il s' agit d' un connecteur type B femelle.

Pour la reconnaissance du périphérique, il est nécessaire de positionner un condensateur d' environ 470nF sur Vusb du PIC (patte 14). Les trois LED nous permettent de savoir où l' on en est dans le programme et ainsi le débugger.

Le Reset peut être pratique: en cas de bug, un reset et la connexion USB est réinitialisée. Cela évite de brancher/débrancher la carte.

Bien entendu, il faut positionner les entrées des circuits non utilisées à la masse (voir pour le trigger 74HC14). Pour le PIC, il suffira de configurer les I/O non utilisées en sortie.

Maintenant que ces deux cartes sont réalisées, il ne reste plus qu'à continuer les différents programmes en donnant à cette télécommande un plus grand champ d'action.

Conclusion

Etant étudiants en seconde année en section GEII et pour améliorer nos connaissances acquises, nous avons choisi un projet que nous devions réaliser dans un temps imparti. Nous avons choisi ce projet car il permet d' aborder des outils technologiques actuels tels que la liaison sans fil et la liaison USB. En effet, ce projet est assez complet. Il aborde une bonne partie informatique ainsi qu' un bon développement électronique.

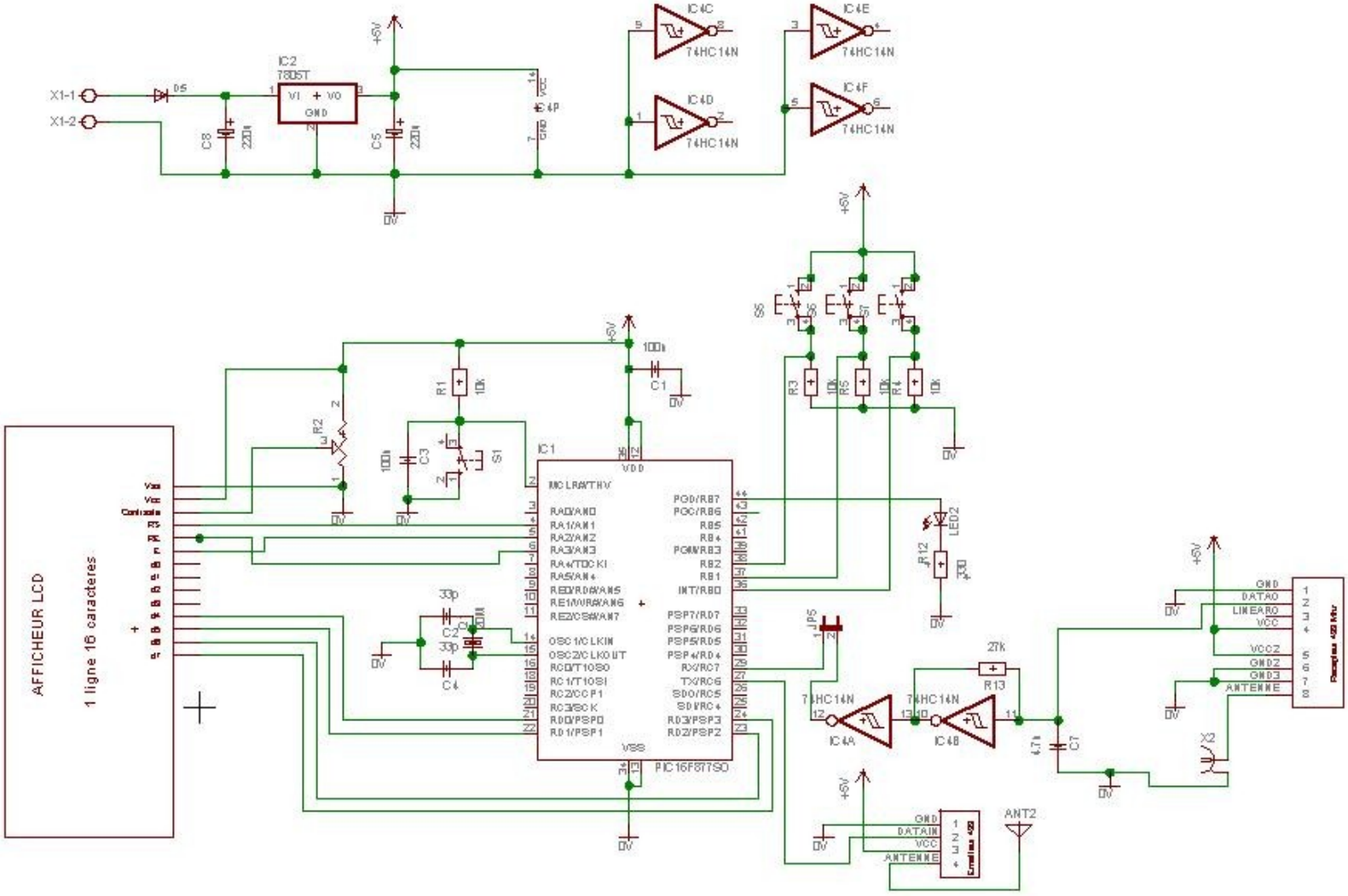
Cette expérience nous a appris à travailler en autonomie et à gérer notre temps de travail. Elle nous a permis de mettre nos connaissances acquises durant ces deux années d'études supérieures en oeuvre. De plus, de nombreuses méthodes de travail ont été mises en place comme la rédaction d' un cahier de bord ainsi que des rapports d' étapes structurant nos recherches. Le projet n' est pas entièrement fini au niveau de la programmation mais on peut déjà avancer le prix de revient à moins de 60 euros pour le prototype, respectant ainsi le cahier des charges. Cette somme est dû à l' achat d' émetteur et récepteur HF ainsi que leurs antennes respectives, il prend également en compte l' afficheur et les pics. Le cahier des charges a été en général bien respecté, il reste à réaliser la partie développement sur l' ordinateur, pour récupérer différentes valeurs courantes des applications Windows.

Pour améliorer notre projet, il serait possible d' augmenter le nombre d' applications Windows compatibles, augmenter le nombre de bouton pour gérer l' ordinateur et passer à un écran graphique 2D. La miniaturisation de nos cartes serait un point essentiel à toute amélioration de notre projet. Celui-ci donne un bon point de départ à celui qui cherche à le continuer. De plus, il est possible de gérer plusieurs ordinateurs avec une seule télécommande. Ainsi, ce projet pourrait devenir une centrale domotique.

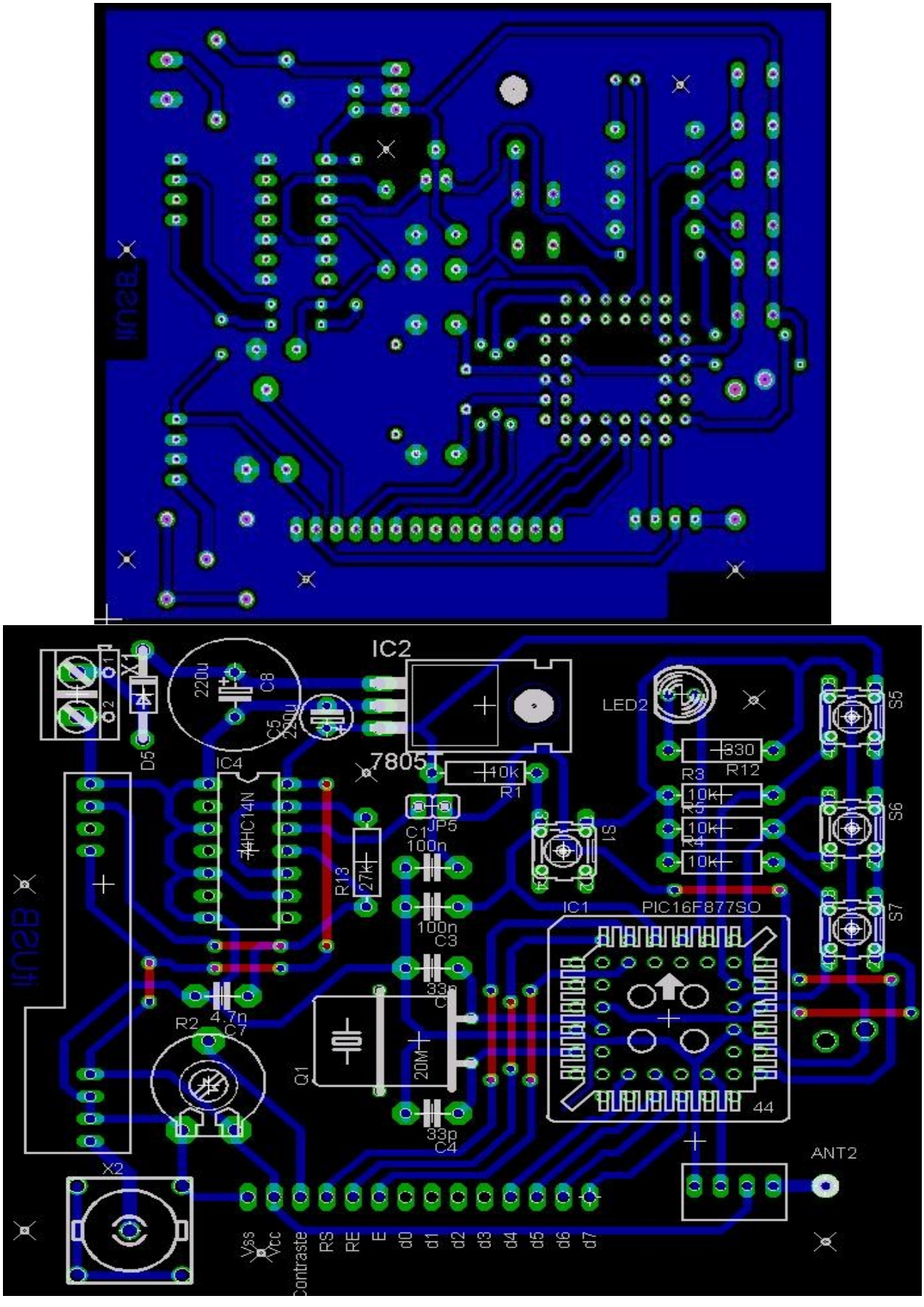
Annexes

Schéma Board et Typon

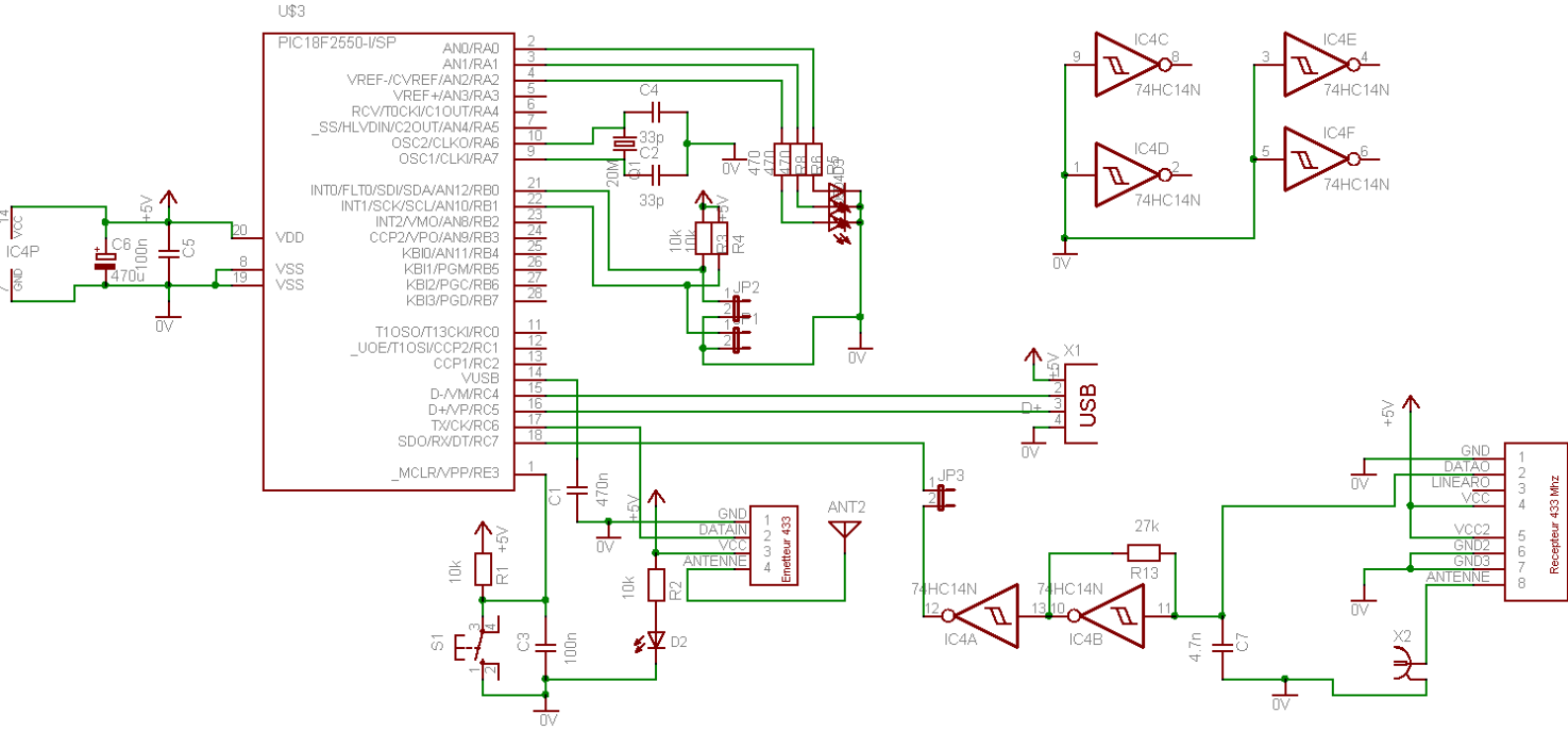
Voici le schéma de la télécommande :



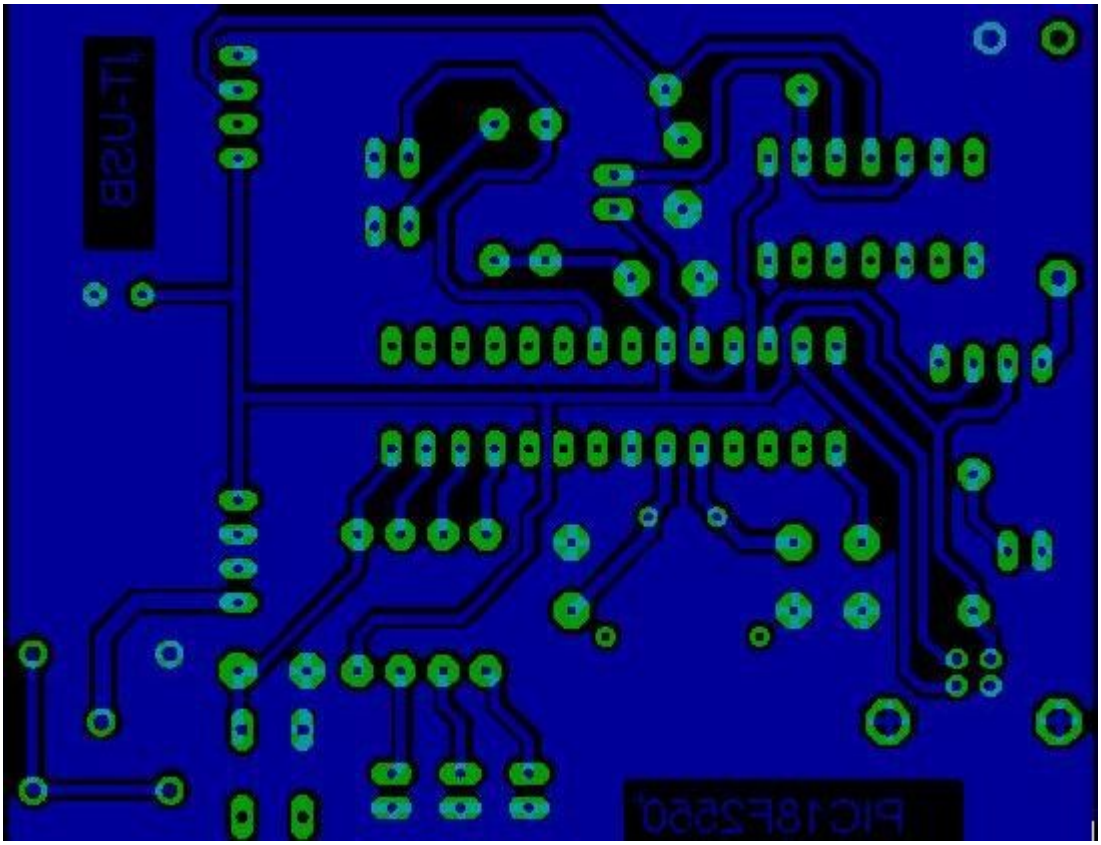
Voici le board et le schéma d'implantation de la télécommande:

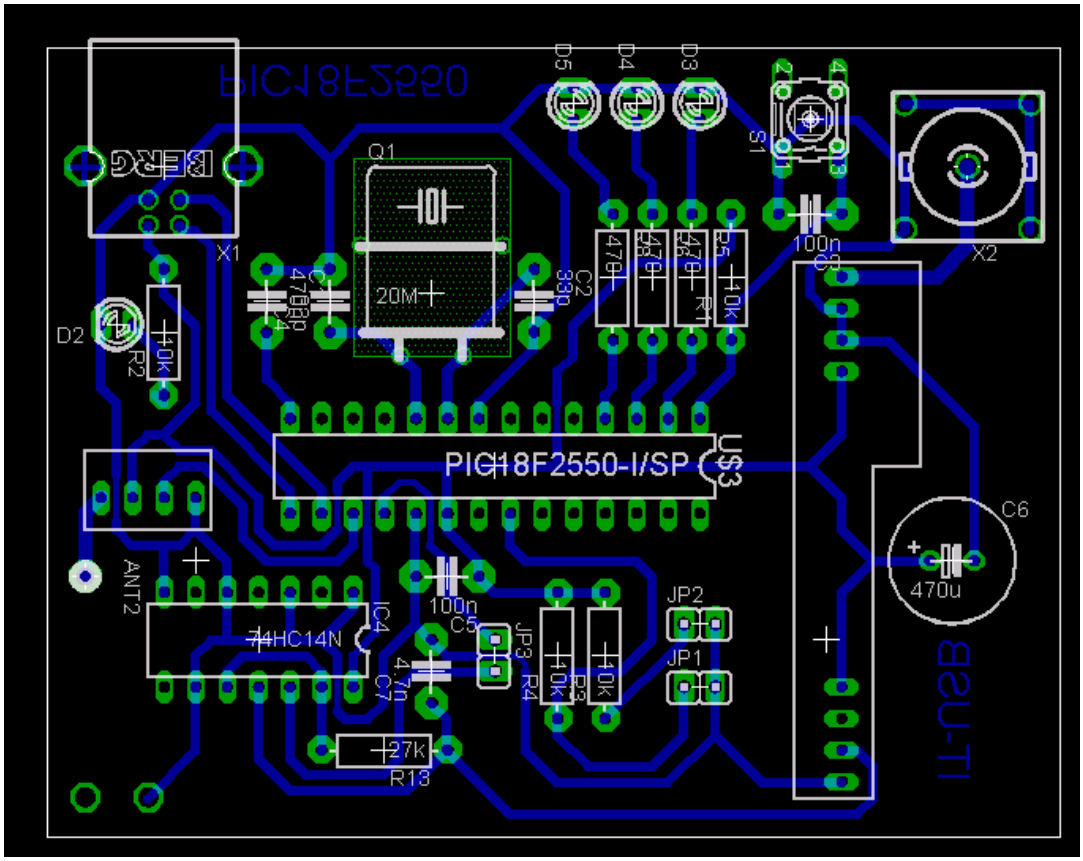

IC2
7805
LED2
330
R12
R3
10k
R1
40k
JP5
C1
100n
S1
IC4
74HC14N
R18
27k
C8
220u
C5
220u
X1
D5
IC1
PIC16F877SO
44
100n
C3
33p
C2
4.7n
C7
R2
Q1
20M
33p
C4
X2
ANT2
S5
S6
S7
Vss
Vcc
Contraste
RS
RE
E
d0
d1
d2
d3
d4
d5
d6
d7

Voici le schéma de la carte USB



Voici le typon et le schéma d'implantation des composants




PIC18F2550
PIC18F2550-I/SP
X1
X2
Q1
20M
Q2
330
Q3
470p
D2
D3
D4
D5
R1
R2
R3
R4
R5
R6
R7
R8
10k
S1
100n
C6
470u
ANT2
IC4
74HC14N
C5
C7
4.7n
JP1
JP2
JP3
R13
27k

Caractérisation technique des cartes

Carte du 16F877

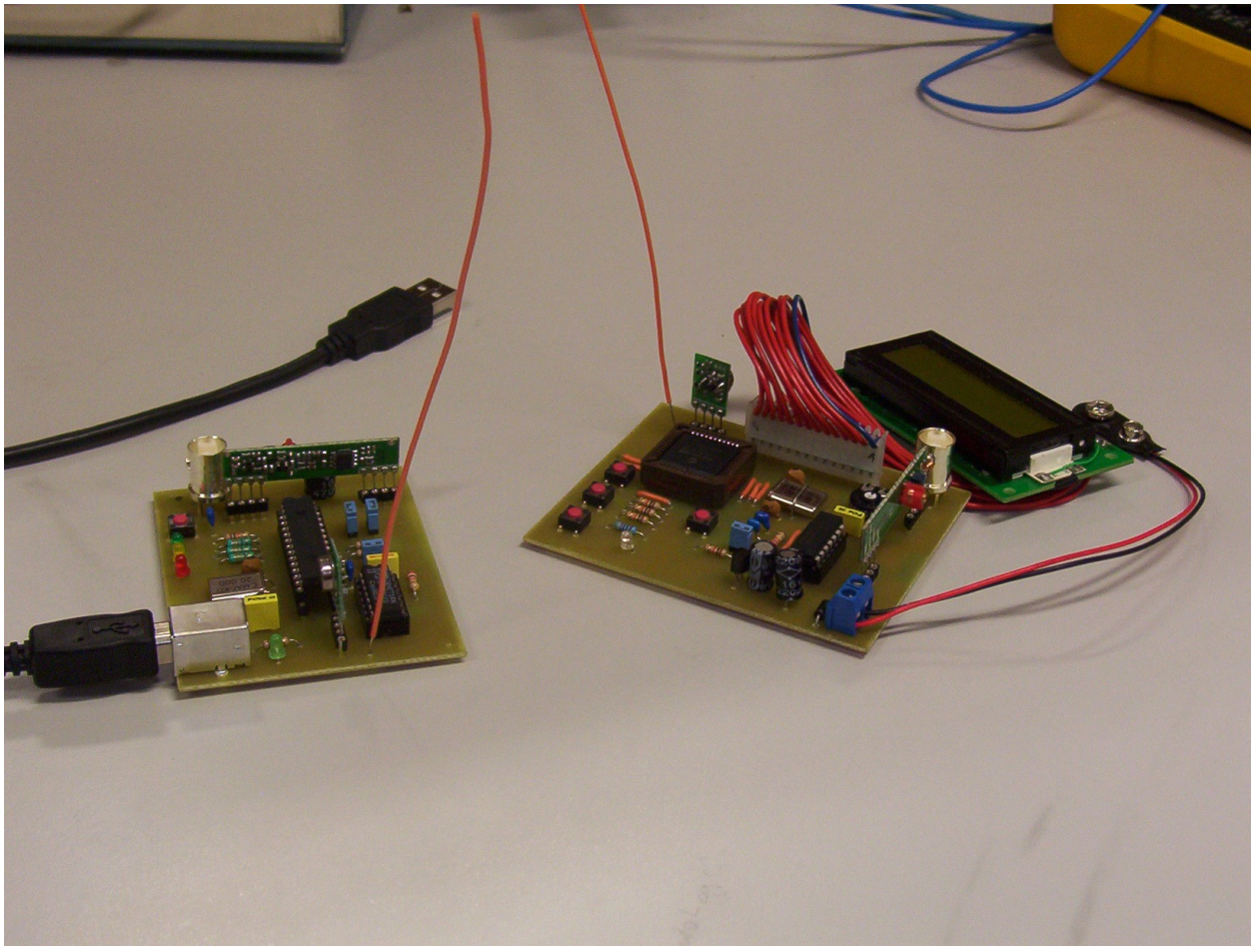
Consommation : 100 mA (avec le rétro-éclairage actif du LCD) à 5V. A savoir que le PIC dispose de la technologie « nanoWatt Technology » permettant une consommation très réduite en cas de veille.

Dimensions du prototype : 90*75*50*

Carte du 18F2550

Consommation : 60 mA à 5V.
Dimensions du prototype : 80*60*15

Voici nos prototypes :



Liens et documents incontournables

http://www.usb.org/developers/hidpage/ Description du HID
http://www.usb.org/developers/vendor/ Pour obtenir un VID
http://www.abcelectronique.com/acquier/USB.html Pour le Protocole USB
http://www.aurel32.net/elec/lcd.php Documentation sur les LCD
http://www.mecanique.co.uk/ EasyHID
http://forums.futura-sciences.com/ Pour le forum électronique
http://www.abcelectronique.com/forum/ Pour le forum électronique
http://www.roboticus.org/index.php?mod=articles&id=26 explication et exemple de l'HID (avec firmware Microchip)
http://pic18fusb.online.fr/ des exemples sur l'HID avec des PIC18F4550
http://www.gmidatabox.fr/gmidatabox/site/fr/infotechniques/guidesdeselection/radiosolutions.htm définitions et explication de la législation française en HF
http://microchip.com/ site de Microchip (tous sur les pic)
http://sample.microchip.com/ pour avoir des échantillons gratuits

Branchements des pattes de programmation des pic

PIC16F877

1	MCLR
32	+5V
31	GND
40	PGD
39	PGC
36	PGM

PIC18F2550

1	MCLR
20	+5V
19	GND
28	PGD
27	PGC
26	PGM

PGM est à positionner à la masse pour une programmation basse tension (fusible LVP désactivé) et à 5V pour un programmation basse tension (LVP activé).

Plusieurs schémas sont disponibles pour la programmation. Il faut entrer « Programmateur PIC JDM » dans un moteur de recherche.

Telecommande Interactive : remote control

This project can be a good start for those who wish to work on the USB port. It is a prototype of a computer remote control. As of now, Windows only is supported, but in the future the firmware for Linux systems will be created.

This first version of Interactive remote control connects to the computer, and sends or receives short data from HF peripheral. However in the near future, we would like to display the title of the song played bye Winamp Software on the remote control. We would also like to turn the volume up or down, change the song that is currently playing etc. Other functions will be possible to add, such as informative messages (« You have got a new email » or other).

If anyone is interested by this project, it would be a great pleasure to share our experience and motivation.

Télécommande Interactive

Ce projet peut être un point de départ pour ceux qui veulent débuter sur le port USB. C'est un prototype d'une télécommande pour ordinateur. Pour l'instant, seul Windows est supporté, mais plus tard, une application sera développée sous Linux.

Cette première version de la télécommande interactive se connecte à l'ordinateur et émet ou reçoit des chaînes de caractères venant d'une liaison HF. Mais dans un futur proche, nous voudrions afficher sur la télécommande le titre du morceau qui passe sur un lecteur multimédia comme Winamp. Nous voudrions également pouvoir modifier le volume, la piste audio, etc. D'autres fonctionnalités pourraient être ajoutées comme le message d'information : « Vous avez un nouveau message ».

Si vous êtes intéressés par ce projet, ce serai un réel plaisir de partager notre expérience et nos motivations.

Contacts :

kevin.raymond@etu.univ-savoie.fr
philippe.corbin@etu.univ-savoie.fr

Mots clés : USB, HF, PIC, HID, liaison série