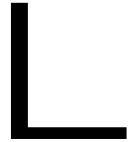


Fabien Chevalier  
Sylvain Richerieux  
Nicolas Sinègre



## PROJET DE CONCEPTION

---

# Commande IR par port USB

---

## *Sommaire*

Introduction .....	2
I. Déroulement du projet .....	3
II. Choix effectués et outils utilisés pour la réalisation .....	4
1. Choix effectués .....	4
2. Outils utilisés .....	4
III. Elaboration d'une méthode de communication .....	5
1. Information de paramétrage .....	5
2. Signal échantillonné .....	6
IV. Dispositif électronique de réception IR .....	7
V. Programme du PIC16C765 .....	8
1. Modification du firmware .....	8
2. Création du HID Report Descriptor .....	8
3. Réalisation du programme principal .....	9
VI. Application de test pour Windows .....	10
VII. Service Windows .....	11
1. Modélisation des télécommandes .....	11
2. Description de l'application .....	11
3. Paramétrage d'une télécommande .....	13
4. Identification des touches .....	15
Conclusion .....	16
Bibliographie .....	17

## ***Introduction***

En ce début de 21<sup>ème</sup> siècle, l'informatique ne cesse de se démocratiser. Dans l'esprit du néophyte, l'ordinateur est passé progressivement du gros calculateur des années 50 à l'IBM PC des années 80, jusqu'à l'ordinateur hyper communiquant et hyper convivial des années 90.

Une des caractéristiques de cet ordinateur convivial est la facilité avec laquelle on peut lui connecter d'autres périphériques. Pour permettre de gagner en simplicité, un nouveau protocole d'interconnexion a été défini à partir de 1995. Il s'agit du protocole USB, pour Universal Serial Bus. Depuis cette date, quasiment tous les périphériques courants sont connectables par l'intermédiaire de l'USB : claviers, souris, enceintes, scanners, caméscopes, appareils photos... On parle même actuellement d'adjoindre un mini connecteur USB aux téléphones portables.

A l'époque où la spécification du protocole USB commençait à être rédigée, un Unix pour PC était activement développé par une équipe d'informaticiens indépendants. Ce nouveau système visait à atteindre les objectifs suivants : plus de souplesse, plus de fiabilité et de meilleures performances. Il allait commencer à être massivement utilisé dans le milieu universitaire, puis dans l'industrie à la fin du 20<sup>e</sup> siècle. Vous l'aurez deviné, il s'agit du système d'exploitation Linux.

Dans le cadre d'un projet mixte électronique/informatique, nous nous proposons d'étudier certains aspects de ces deux révolutions du monde de la micro-informatique. Nous allons développer un récepteur infrarouge se connectant sur ordinateur par protocole USB, et disposant d'un logiciel de configuration convivial, généré autant que faire se peut avec un code source identique pour les plateformes Windows et Linux, selon le paradigme « write once, run anywhere ». Ce récepteur permettra de piloter un PC à partir de n'importe quelle télécommande universelle.

## ***I. Déroulement du projet***

Nous nous sommes organisés suivant l'emploi du temps suivant :

- Rédaction d'un cahier des charges
- Documentation sur le protocole USB et la spécification HID
- Analyse des signaux émis par des télécommandes
- Elaboration d'un protocole de communication
- Réalisation d'une maquette électronique de test
- Réalisation du programme embarqué
- Création d'un programme Windows de test de la maquette
- Développement d'un service Windows
- Développement d'une DLL interfaçant la maquette USB
- Développement d'une application multi plateforme Windows/Linux

La réalisation du rapport et de la documentation s'effectue tout au long de ces étapes.

## II. Choix effectués et outils utilisés pour la réalisation

### 1. Choix effectués

Pour la partie électronique, nous avons à notre disposition une maquette USB basée sur un PIC16C765. Nous utiliserons un récepteur infrarouge de type TSOP1836 pour réaliser le montage électronique qui réceptionnera les signaux infrarouges et qui les enverra au PC par le port USB.

Nous avons décidé également de recourir à la spécification HID, offrant une grande souplesse dans le formatage des données. Pour obtenir plus de précisions, à la fois sur le protocole USB, mais aussi sur cette spécification HID, vous pouvez vous reporter aux pages web de l'électronique à Supélec sur le campus de Rennes : <http://www.supelec-rennes.fr/ren/fi/elec>.

### 2. Outils utilisés

Nous allons être amenés à utiliser différents logiciels que ce soit pour programmer le micro-contrôleur, développer les programmes de test ou l'application PC.

Voici la liste des outils que nous utiliserons :

- Maquette USB basée sur un PIC16C765 réalisé par J. Weiss
- Carte de programmation pour PIC réalisée par J. Weiss
- Logiciel de programmation pour PIC16F628 : PIC Prog réalisé par J. Weiss
- MPLab 6.10 de Microchip : environnement de développement intégré pour réaliser le programme embarqué
- Borland Delphi 7 pour Windows
- Borland Kylix 3 pour Linux
- Microsoft Visual C++ 6
- HIDKomponente : Composant Delphi gérant les périphériques HID

### III. Elaboration d'une méthode de communication

Avant d'élaborer une méthode de communication, nous avons d'abord analysé les signaux émis par différentes télécommandes. Nous avons remarqué que chaque télécommande émet suivant un protocole différent même pour des télécommandes de même marque. Les temps d'émission des signaux ainsi que le temps de latence entre deux signaux peuvent varier facilement dans un rapport 10.

Au départ nous avons pensé analyser les signaux reçus sur le PIC et transmettre au PC un code de télécommande et un code de touche. Vu la disparité des protocoles, cette technique n'est pas viable. Le PIC ne peut pas contenir une base donnée des différents protocoles existants. Nous avons donc choisi d'échantillonner brutalement le signal reçu du récepteur infrarouge et de le transmettre au PC qui se chargera alors de l'analyser.

Deux méthodes sont maintenant envisageables : soit transmettre le signal échantillonné au PC au fur et à mesure de la réception, soit de le stocker en mémoire sur le PIC et de le transmettre au PC une fois échantillonné. La faible durée de la période d'échantillonnage ne permet pas de traiter le code de capture du signal et d'envoi de paquet USB simultanément, nous choisissons donc de transmettre le signal au PC une fois l'échantillonnage terminé.

Deux types d'informations doivent donc circuler sur la ligne USB :

- Les informations de paramétrage pour l'échantillonnage (PC vers maquette USB)
- Les données reçues du récepteur infrarouge (maquette USB vers PC)

#### 1. Informations de paramétrage

Déterminons les paramètres d'échantillonnage à envoyer au PIC.

La période d'échantillonnage est le premier paramètre. Pour échantillonner le signal sur le PIC nous utilisons le timer0. L'échantillonnage se fait donc à chaque interruption. Pour régler le délai entre deux interruptions, il faut utiliser les registres prescaler et timer0. Le prescaler permet de déterminer le nombre d'instructions qui doivent s'exécuter avant d'incrémenter le registre timer0. A chaque débordement de timer0, une interruption est déclenchée, on réinitialise alors le registre timer0 à une valeur précise (timer0reset) qui permet de régler finement la période d'échantillonnage. La période d'échantillonnage peut ainsi varier de 50µs à 500µs.

Pour régler la période d'échantillonnage, on doit donc envoyer au PIC deux valeurs : prescaler et timer0reset. Ces deux valeurs sont codées sur 1 octet chacune.

La durée d'échantillonnage est le deuxième paramètre à envoyer au PIC. On ne transmettra pas au PIC la durée d'échantillonnage mais le nombre d'échantillons à capturer, ce qui est plus simple à gérer au niveau de la programmation PIC. La petite quantité de mémoire disponible sur le PIC pour sauvegarder le signal échantillonné ne permet pas de travailler sur un très grand nombre d'échantillons. Nous avons choisi de capturer le signal sur un nombre d'échantillons d'au maximum 448, ce qui est suffisant pour avoir un niveau de détail correct sur le signal. Nous justifierons ce nombre dans la partie suivante. Le nombre d'échantillons du signal est codé sur 2 octets.

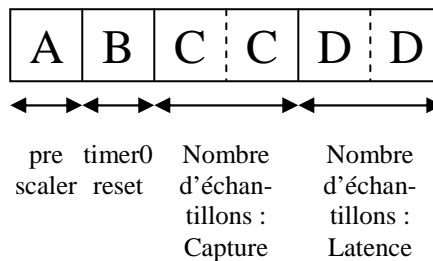
Ces paramètres peuvent à priori suffire, mais après analyse plus poussée des signaux des télécommandes, on s'aperçoit que la télécommande peut émettre un nouveau signal avant que le

PIC soit prêt à en recevoir un nouveau (du fait du temps d'envoi du signal au PC). Dans ce cas le PIC pourrait perdre le début du signal suivant. Pour éviter ce problème on spécifie au PIC un nombre minimum d'échantillons « blancs » consécutifs devant être reçus avant de démarrer une nouvelle capture. Cette valeur est codée sur 2 octets.

Techniquement parlant, ces informations seront envoyées sous forme d'un paquet unique de type feature. Les données contenues dans un tel paquet sont donc :

- Valeur du prescaler (entre 0 et 7) : 1 octet
- Valeur de timer0reset (entre 0 et 255) : 1 octet
- Nombre d'échantillons à capturer (entre 1 et 448) : 2 octets
- Nombre d'échantillons de latence (entre 0 et 65535) : 2 octets

Un paquet est constitué de la manière suivante (décomposé en octets) :



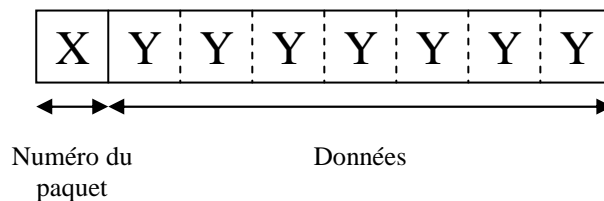
## 2. Signal échantillonné

Le PIC fournit au PC seulement les signaux échantillonnés suivant les paramètres précédemment fournis par le PC.

Justifions maintenant la valeur de 448 échantillons. Un échantillon est codé sur un bit, on prend donc un multiple de 8 échantillons pour occuper un nombre entier d'octets. Le protocole USB basse vitesse permet de transférer des paquets d'au maximum 8 octets. Comme un signal échantillonné ne tient pas dans un paquet unique, on le transmet en plusieurs paquets. On réserve alors le premier octet de chaque paquet pour identifier le paquet (numéro du paquet). Les 7 autres octets de chaque paquet contiennent les données utiles. Le nombre d'échantillons doit donc être aussi divisible par 7 pour optimiser le remplissage d'un paquet. A la vue des signaux émis par les télécommandes, on choisit un nombre maximal d'échantillons de 448 qui est suffisant pour bien visualiser un signal. Les 448 échantillons occupent pleinement 8 paquets.

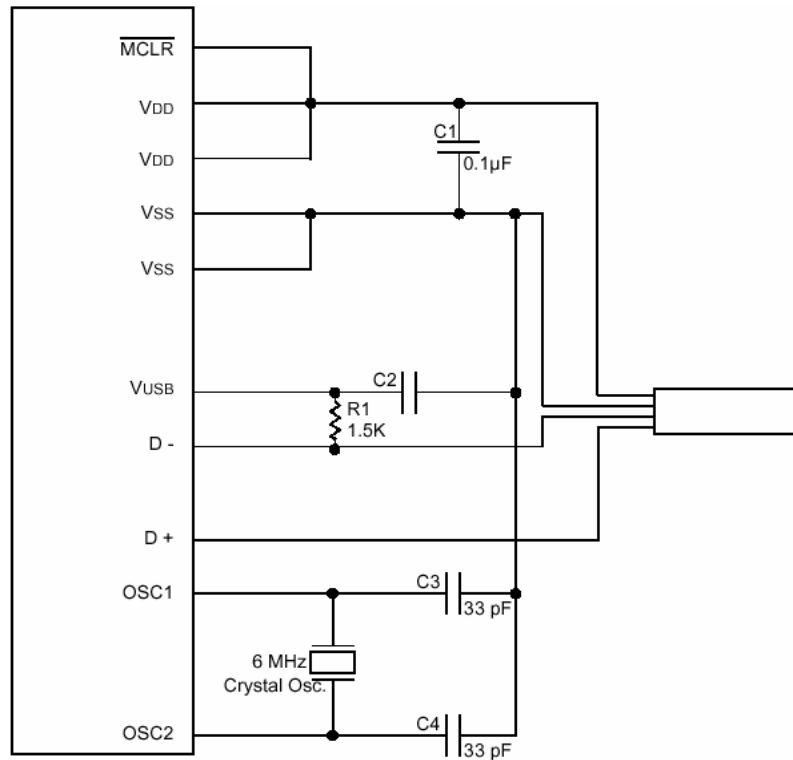
Les paquets transmis au PC sont de type input.

Un paquet est constitué de la manière suivante (décomposé en octets) :



#### IV. Dispositif électronique de réception IR

Le montage que nous allons utiliser se base sur la plaque USB réalisé par J. Weiss. Cette plaque comporte un PIC16C765 câblé selon les indications fournies dans la datasheet du PIC16C765 :



Nous ajoutons à ce montage le récepteur infrarouge. La broche de données du récepteur est connectée à la broche PORTC1 du PIC16C765. Les données issues du récepteur infrarouge sont directement exploitables. Le récepteur se charge de démoduler le signal.

## V. Programme du PIC16C765

Nous avons commencé par comprendre le fonctionnement du firmware USB fournit par Microchip pour le PIC16C765. Un programme d'exemple de souris USB HID est fourni avec le firmware.

Le firmware a été développé dans le but de fonctionner avec le programme d'exemple de la souris. Il ne gère donc pas la réception de données de type output sur la terminaison 0. Nous devons apporter les modifications nécessaires afin que le PIC reçoive des données en provenance du PC.

Les étapes pour développer le programme sont donc :

- Modification du firmware pour gérer la réception de données
- Création d'un nouveau Report Descriptor conforme aux spécifications HID
- Réalisation du programme réalisant la capture des signaux

### 1. Modification du firmware

Le firmware gère la réception de jeton d'installation (setup token) et de jeton d'entrée (in token) mais ne gère pas la réception de jeton de sortie (out token). Cependant la structure de gestion est déjà présente dans le firmware. Nous avons donc rajouté du code assurant la réception des données en provenance du PC sur la terminaison 0 (terminaison de contrôle pour le protocole HID).

Le nouveau code est inséré au niveau du label TokenOutPID dans le fichier usb\_ch9.asm.

Pour se rendre compte du travail effectué, il est conseillé de se rendre sur les pages de Supélec <http://www.supelec-rennes.fr/ren/fi/elec>.

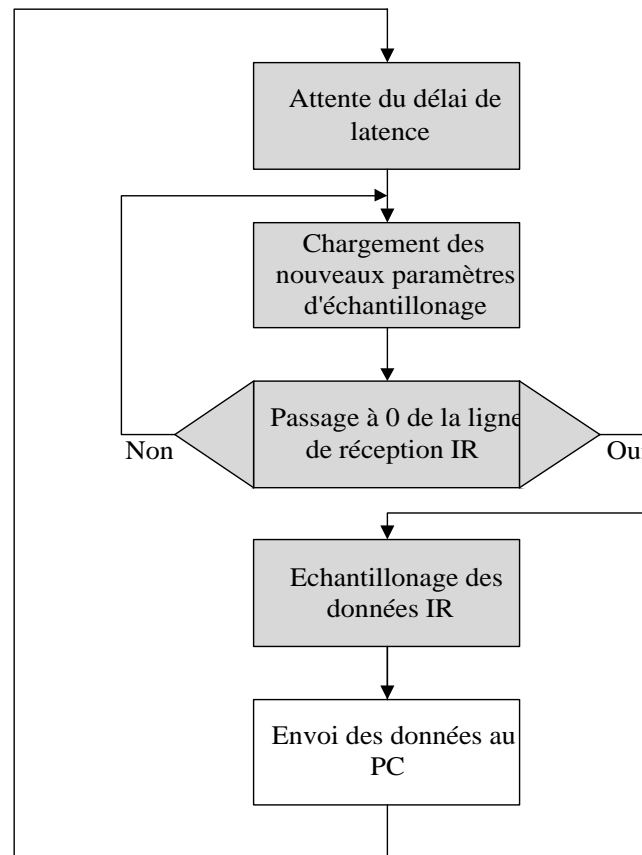
### 2. Création du HID Report Descriptor

Nous avons créé un nouveau report descriptor conforme aux normes HID représentant les données qui vont s'échanger entre le PC et le périphérique USB (en accord avec le protocole décrit dans la partie III). L'élaboration du HID Report Descriptor est expliquée dans les pages HTML.

Comme précédemment, pour plus de détails, consulter le site web.

### 3. Réalisation du programme principal

Le programme principal est chargé d'échantillonner le signal et de le transmettre au PC une fois échantillonné. Le déroulement du programme est représenté par l'organigramme suivant :



L'échantillonnage du signal se fait sous interruption (étapes représentées en grisée dans l'organigramme). Seul le transfert des données au PC via l'USB s'exécute hors interruption.

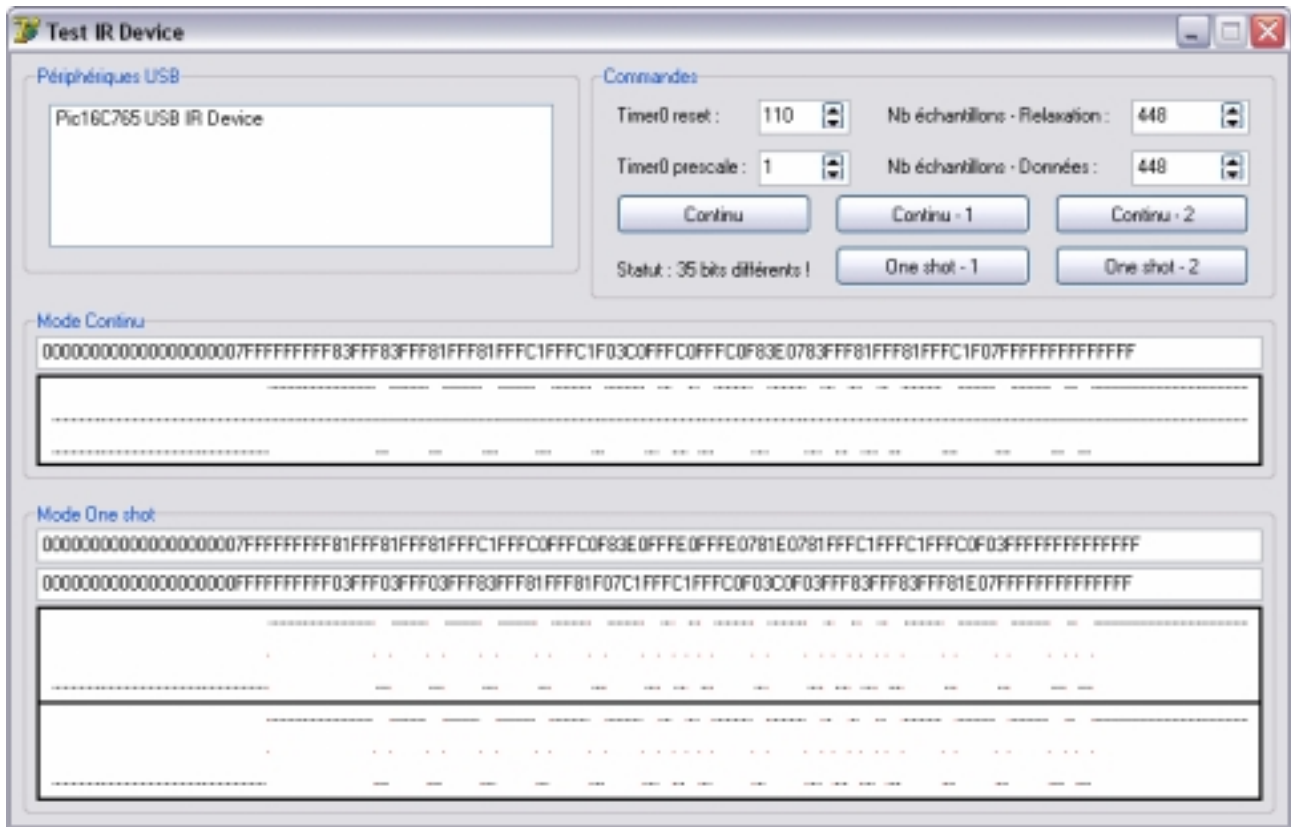
Il est à noter que la gestion de l'USB s'effectue aussi par interruption. Le gestionnaire d'interruption donne priorité à la procédure d'échantillonnage.

Nous ne détaillerons pas plus le programme.

## VI. Application de test pour Windows

Nous avons développé rapidement une application de test afin de pouvoir vérifier le bon fonctionnement du programme embarqué du PIC. A l'aide de cette application, nous pouvons définir les paramètres d'échantillonnage ainsi que visualiser les signaux issus du récepteur infrarouge.

Voici l'interface de cette application :



On a pu remarquer que l'échantillonnage d'une touche ne donne pas toujours strictement le même signal. En effet, certaines transitions peuvent être décalées d'un ou deux bits dans le temps (points rouges sur la ligne centrale dans la capture d'écran ci-dessus).

## VII. Service Windows

### 1. Modélisation des télécommandes

L'application est capable de gérer plusieurs télécommandes. Nous avons donc défini une structure assurant la gestion des télécommandes.

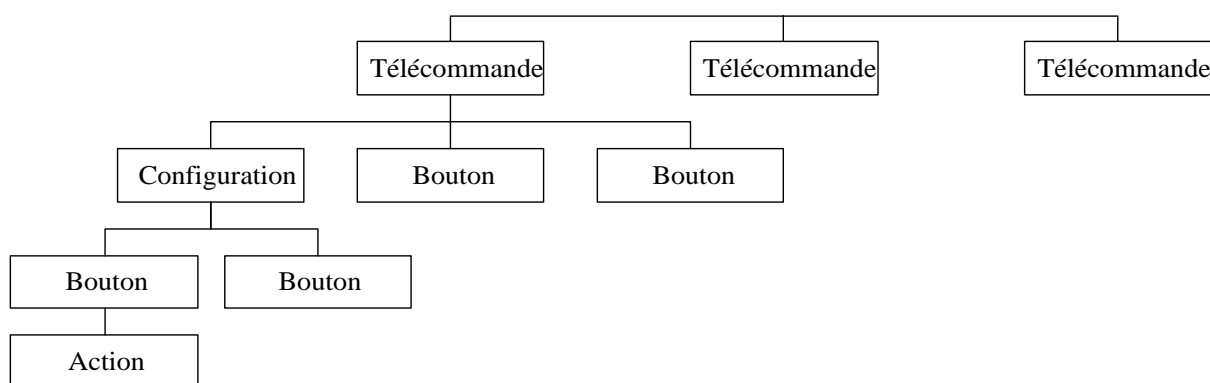
Chaque télécommande peut contenir des touches et des configurations. Les configurations peuvent elles-mêmes contenir des touches. Une touche représente physiquement une touche d'une télécommande. Par contre, les configurations représentent un ensemble abstrait de touches. Les touches attachées directement à une télécommande font partie implicitement d'une configuration nommée « Configuration commune ».

Vu que les différentes télécommandes n'utilisent pas les mêmes paramètres d'échantillonnage, une seule télécommande peut être active à la fois. Pour cette télécommande, une seule configuration est active. Les touches d'une configuration sont accessibles si cette configuration est active. Les touches de la configuration commune sont accessibles quelle que soit la configuration active.

A chaque touche, on associe une action qui peut se traduire par :

- Choix de la configuration active
- Commande système (Arrêter l'ordinateur...)
- Basculer de tâche
- Frappe d'une touche du clavier
- Déplacement, clic de souris
- Augmenter, diminuer, couper le son

On peut représenter la structure gérant les télécommandes comme ceci :

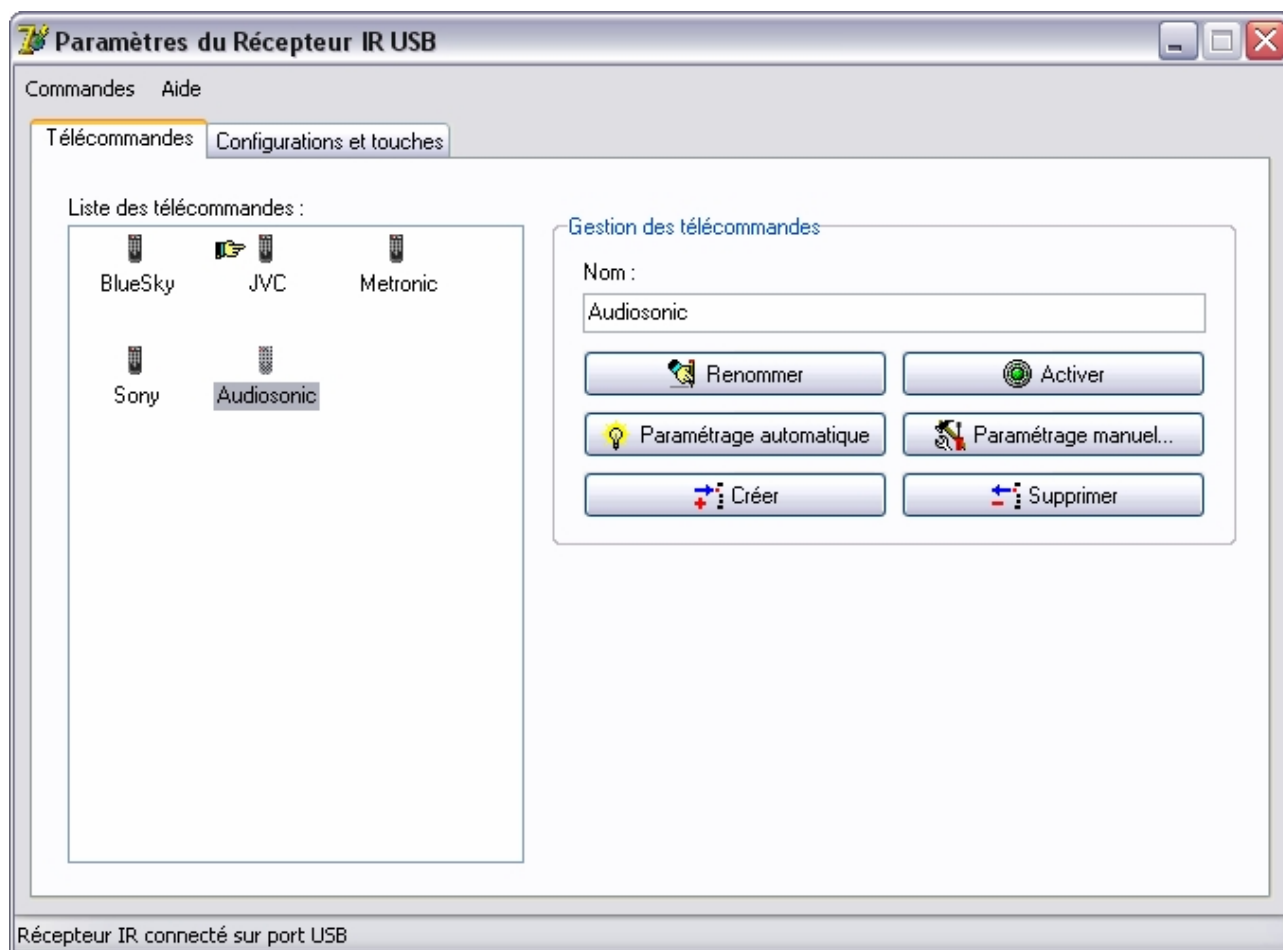


### 2. Description de l'application

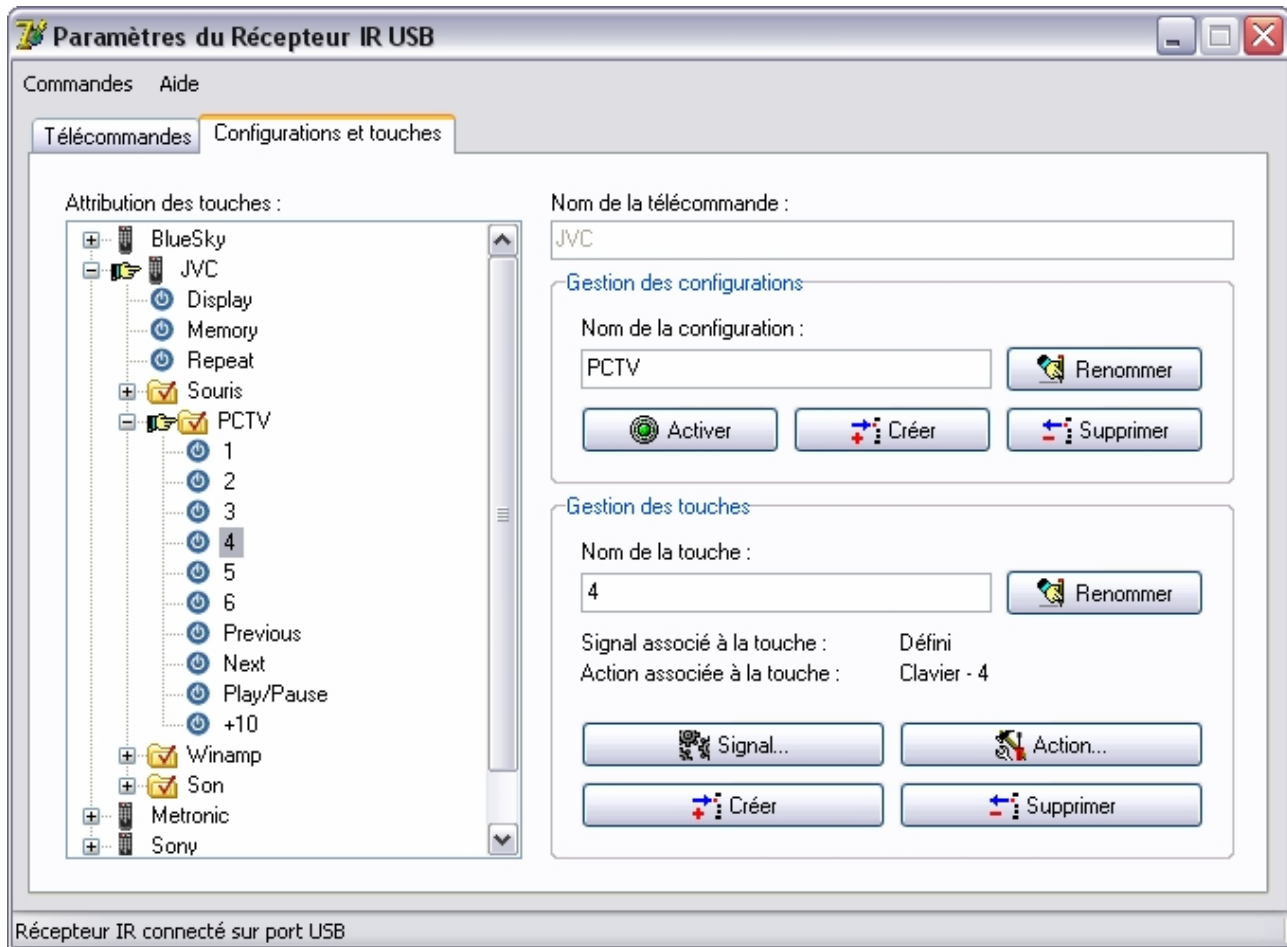
L'application se présente sous forme d'un service tournant sous le compte Système local de Windows 2000/XP. Pour gérer les périphériques HID, on utilise un composant Delphi HIDKomponente que l'on a trouvé sur Internet.

Une icône est présente en permanence dans la barre des tâches indiquant l'état de l'application. Cette icône sert à accéder à la fenêtre de paramétrage qui comporte deux onglets permettant de modifier la configuration des télécommandes.

La fenêtre de paramétrage se présente sous la forme suivante :



Cet onglet permet de créer, de modifier ou de supprimer des télécommandes.



Cet onglet permet d'assurer la gestion des configurations et des touches des télécommandes. C'est ici que l'on définit le signal et l'action de chaque touche de télécommande.

### 3. Paramétrage d'une télécommande

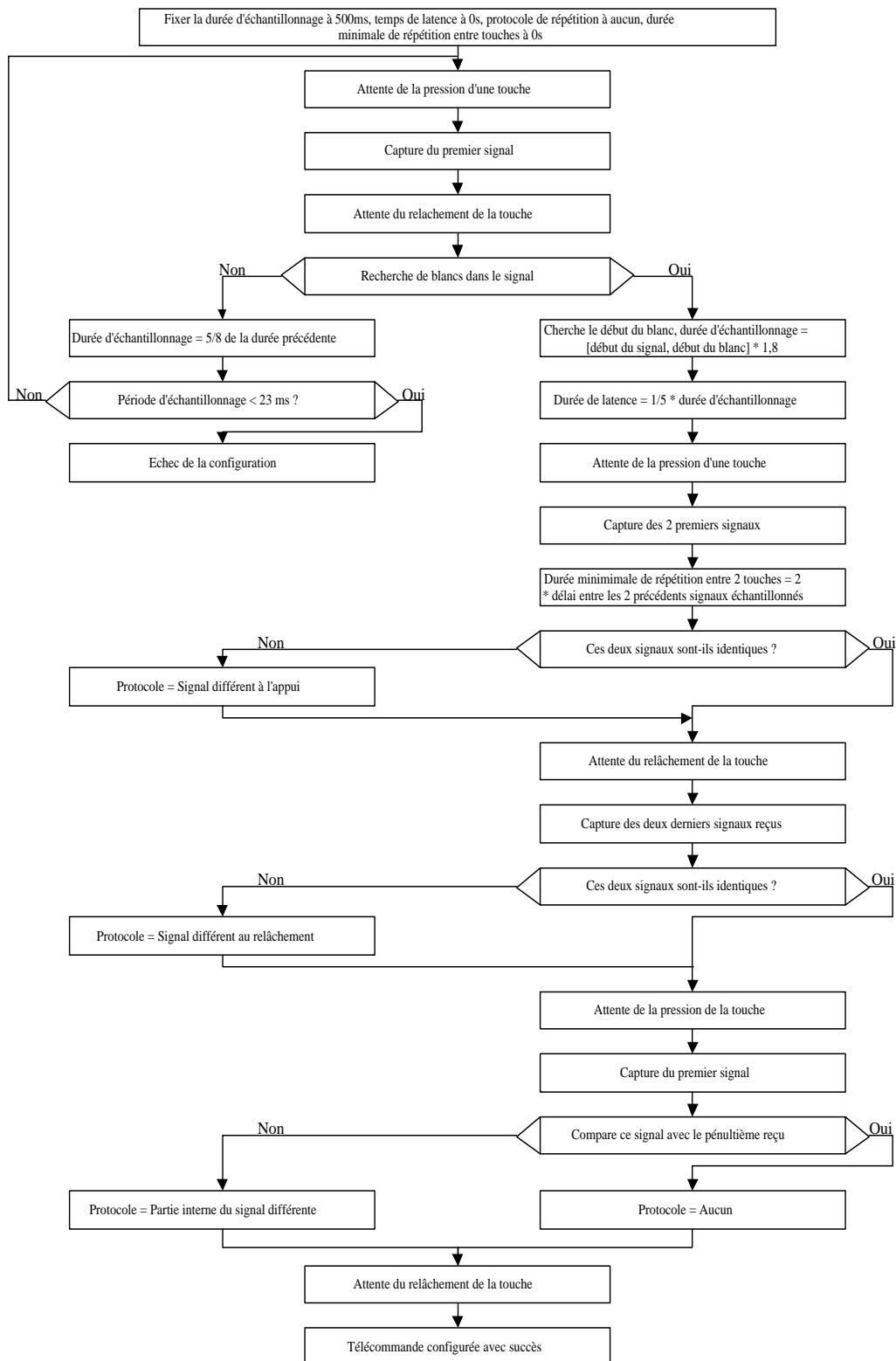
Nous avons implémenté une détection automatique des paramètres de télécommande. La détection automatique des paramètres de la télécommande se charge de déterminer les paramètres suivants :

- Durée d'échantillonnage d'un paquet
- Durée de latence entre deux paquets
- Durée minimum avant répétition d'une touche (si le signal de la touche est identique)
- Protocole de répétition de la télécommande

Les différents protocoles de répétitions trouvés lors de l'analyse des signaux des télécommandes sont :

- Aucun protocole de répétition
- Signal différent à l'appui de la touche
- Signal différent au relâchement de la touche
- Partie interne du signal différente (bit de répétition)

Le déroulement de la détection automatique est décrit par l'organigramme suivant :



#### **4.     *Identification des touches***

L'identification d'une touche enfoncée par l'utilisateur se réalise en comparant le signal reçu à tous les autres signaux des touches de la télécommande et de la configuration active.

Deux signaux sont considérés comme identiques si toutes leurs transitions ne sont pas décalées temporellement de plus de deux échantillons (en retard ou en avance). Toutes autres différences impliquent que les signaux sont considérés comme non identiques.

## ***Conclusion***

A mi-parcours du projet, celui-ci nous a permis de comprendre en profondeur le fonctionnement du bus USB. Nous avons pu également étudier le fonctionnement d'un nombre important de télécommandes, ce qui nous a amené à constater la réelle diversité des protocoles infrarouges utilisés.

Nous avons programmé le PIC, et développé une maquette de l'application finale. Cette maquette a été développée au plus vite, et elle nécessite de profondes modifications pour devenir l'application finale (principalement réorganisation du code et retrait du composant USB utilisé, qui est spécifique à Delphi et ne fonctionne que sous Windows). Il y a par conséquent encore matière à travailler durant les deux mois qui nous séparent de la fin du projet.

## ***Bibliographie***

La norme du protocole USB 1.10

La spécification HID 1.11

Les pages web de Bernard Acquier : <http://perso.wanadoo.fr/bernard.acquier/index.html>

La datasheet du PIC 16C765

La documentation du firmware 1.25 pour le PIC 16C765

**Fin**