

Systeme Temps Réel

Décodeur de télécommande infrarouge

1 Sommaire

1	Introduction	3
2	La partie Hardware.....	3
2.1	Le récepteur infrarouge :	3
2.2	Le port parallèle du PC	3
2.3	Le montage.....	3
3	La partie temps Réel.....	3
3.1	L'acquisition	3
3.1.1	Le module d'acquisition	3
3.1.2	La télécommande	3
3.2	Le décodage	3
4	La partie Apprentissage.....	3
5	Les divers problèmes rencontrés	3
6	Codes Sources	3
6.1	Module d'acquisition temps réel.....	3
6.2	Processus utilisateur.....	3

2 Table des figures

Figure 1 :	Vue d'ensemble du projet.....	3
Figure 2 :	Récepteur infrarouge Siemens de type SFHxxxx	3
Figure 3 :	Schéma bloc d'un récepteur SFHxxxx.....	3
Figure 4:	Port parallèle	3
Figure 5:	Adresses des différents registres de l'imprimante.....	3
Figure 6:	Registre de données	3
Figure 7:	Registre d'état.....	3
Figure 8:	Le dispositif	3
Figure 9:	Boitier Récepteur	3
Figure 10 :	Télécommande JVC.....	3
Figure 11:	diagramme module d'acquisition temps réel.....	3
Figure 12 :	diagramme processus utilisateur	3

3 Introduction

Le projet consiste, à l'aide d'un récepteur connecté sur un PC via le port parallèle, de recevoir et décoder les trames émises d'une télécommande infrarouge.

Le travail est divisé en trois parties:

- une partie hardware qui consiste à effectuer la connexion du récepteur à infrarouge à l'ordinateur,
- une partie Temps Réel qui accède à ce dispositif,
- une partie apprentissage des différents codes émis par la télécommande.

Pour la réalisation du projet nous avons besoin de très peu de matériel, un récepteur infrarouge, une source d'alimentation pour celui-ci, un PC possédant un port parallèle et fonctionnant sous RT-linux et une télécommande infrarouge.

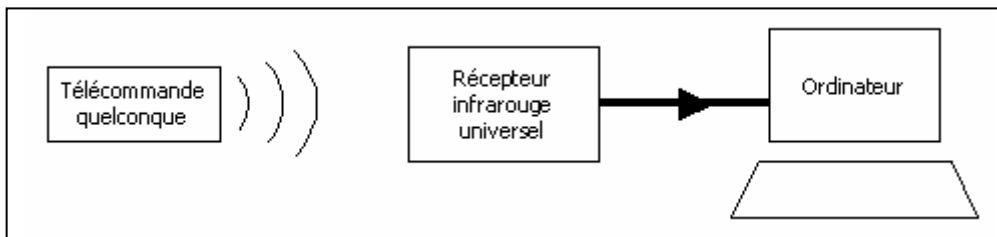


Figure 1 : Vue d'ensemble du projet

4 La partie Hardware

Pour réaliser le dispositif, nous utilisons un récepteur infrarouge que nous connectons au PC via le port parallèle.

4.1 Le récepteur infrarouge :

Il faut trouver un récepteur infrarouge adapté à la télécommande. La plupart des télécommandes émettant leur porteuse entre 35 et 40kHz, nous utilisons un récepteur 36kHz. Très peu de télécommande utilisent la porteuse de 32 KHz.

Un récepteur infrarouge de type SFHxxxx de Siemens correspond à nos attentes.



Figure 2 : Récepteur infrarouge Siemens de type SFHxxxx

Le récepteur se charge du filtrage, de l'amplification et de la démodulation du signal, et fournit un signal série TTL (0-5V) sur une broche.

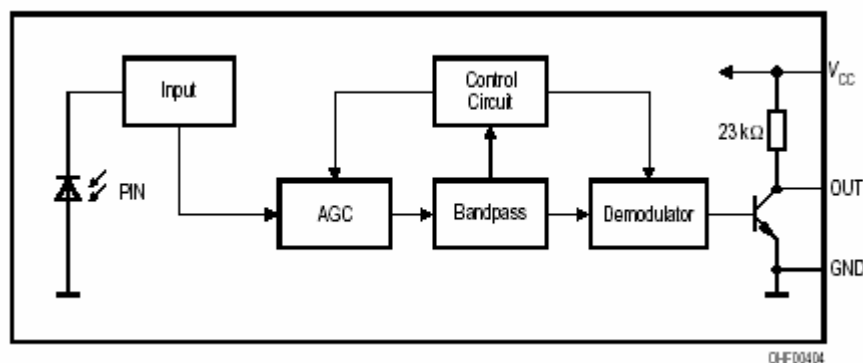


Figure 3 : Schéma bloc d'un récepteur SFHxxxx

Le récepteur SFHxxxx dispose de 3 broches :

- Vcc pour l'alimentation 5V, 5 ma
- GND pour la masse
- OUT pour les données

4.2 Le port parallèle du PC

Le port parallèle du PC a été spécifiquement conçu pour y brancher des imprimantes comportant une interface de communication parallèle pour pouvoir dialoguer avec l'ordinateur. Mais on peut aussi utiliser ce port comme un port général d'entrées-sorties pour une infinité d'autres applications. Le port présente 12 sorties et 4 entrées qui peuvent être accédées par le processeur par des instructions de lecture et d'écriture sur le port désiré.

Le port parallèle est disponible grâce à un connecteur de 25 fils de type D (DB25) femelle situé à l'arrière de l'ordinateur. Il est fait de 25 trous, disposés comme suit:

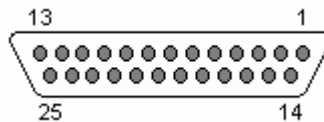


Figure 4: Port parallèle

Chaque broche a une fonction particulière, relative à son utilisation avec une imprimante:

1. STROBE : cette ligne active basse (donc a 0) indique à l'imprimante que des données sont présentes sur les lignes D0 à D7 et qu'il faut les prendre en compte.

2 à 9. D0 à D7 : c'est le bus de données sur lequel véhicule la valeur du caractère à imprimer. On ne peut écrire sur ce port, à moins d'avoir un port parallèle étendu (c'est le cas pour les ports de type ECP/EPP).

10. ACK : l'imprimante met à 0 cette ligne pour indiquer à l'ordinateur qu'elle a bien reçu le caractère transmit et qu'il peut continuer la transmission.

11. BUSY : cette ligne est mise à 0 par l'imprimante lorsque son buffer de réception est plein. L'ordinateur est ainsi averti que celle-ci ne peut plus recevoir de données. Il doit attendre que cette ligne revienne à 1 pour recommencer à émettre.

12. PE : signifie " paper error ". L'imprimante indique par cette ligne à l'ordinateur que l'alimentation en papier a été interrompue.

13. SELECT : cette ligne indique à l'ordinateur si l'imprimante est "on line" ou "off line".

14. AUTOFEED : lorsque ce signal est à 1, l'imprimante doit effectuer un saut de ligne à chaque caractère "return" reçu. En effet, certaines imprimantes se contentent d'effectuer un simple retour du chariot en présence de ce caractère.

15. ERROR : indique à l'ordinateur que l'imprimante a détecté une erreur.

16. INIT : l'ordinateur peut effectuer une initialisation de l'imprimante par l'intermédiaire de cette ligne.

17. SELECT IN : l'ordinateur peut mettre l'imprimante hors ligne par l'intermédiaire de ce signal.

18 à 25. MASSE : c'est la masse du PC.

Trois registres seulement sont nécessaires au contrôle total des signaux, que l'on contrôle par les ports d'entrée/sortie du PC :

Port parallèle	Port du registre de données	Port du registre d'état	Port du registre de commande
n°1	378h	379h	37Ah
n°2	278h	279h	27Ah
n°3	3BCh	3BDh	3BEh

Figure 5: Adresses des différents registres de l'imprimante

Le registre de Données :

Ce registre n'est accessible qu'en écriture.

Bit	7	6	5	4	3	2	1	0
Nom	D7	D6	D5	D4	D3	D2	D1	D0

Figure 6: Registre de données

Le registre d'état :

Ce registre, accessible uniquement en lecture au contraire du précédent.

Bit	7	6	5	4	3	2	1	0
Nom	/BUSY	ACK	PE	SELECT	/ERROR	X	X	X

Figure 7: Registre d'état

Le registre de Commandes :

Ce dernier registre est accessible à la fois en lecture et en écriture .

4.3 Le montage

L'alimentation du récepteur infrarouge est fourni par 4 piles de 1V5 (6V).

Pour la connexion sur le port parallèle, nous utilisons la pin 10 (ACK) accessible en lecture à partir du PC sur le registre d'état à l'adresse 0x379 (bit 6).

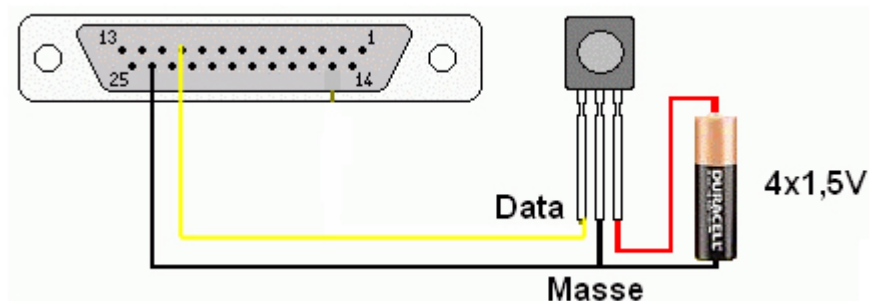


Figure 8: Le dispositif



Figure 9: Boîtier Récepteur

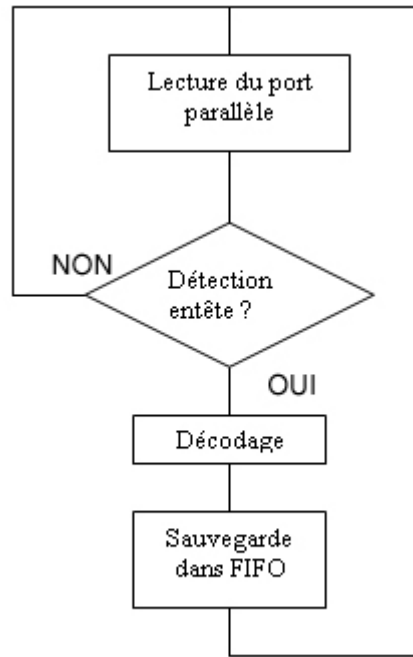


Figure 11: diagramme module d'acquisition temps réel

6 La partie Apprentissage

Le programme utilisateur lit dans la FIFO les bits envoyés par le programme temps réel. Il lit successivement l'octet codant l'appareil puis l'octet codant la commande. Il interprète ensuite ces octets à partir des informations stockées dans le fichier « commandes.jvc »

Le programme utilisateur comporte deux modes :

- Apprentissage
- Normal

Le mode Normal

Pour chaque octet reçu, le programme va chercher l'intitulé de cet octet correspondant dans le fichier. Si il n'existe pas, le programme le signale : « commande inconnue »

Le mode Apprentissage

Pour chaque octet reçu, le programme va chercher l'intitulé de cet octet correspondant dans le fichier. Si il n'existe pas, le programme demande à l'utilisateur de saisir l'intitulé associé, et le sauve dans le fichier.

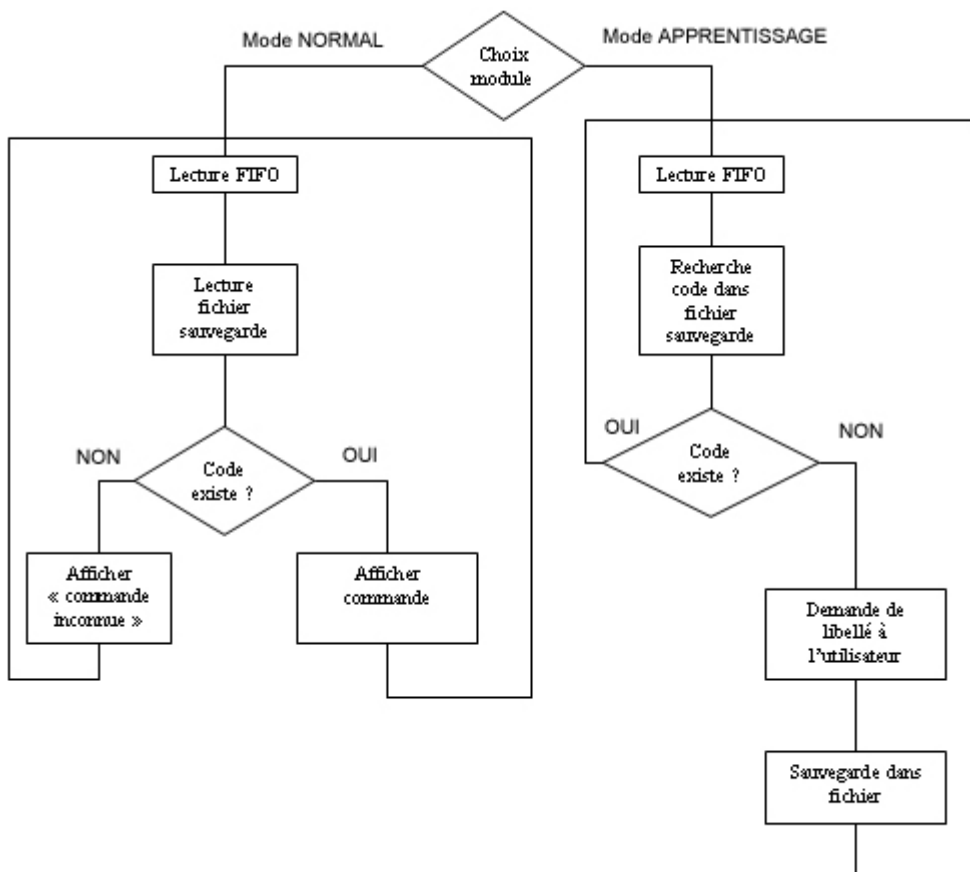


Figure 12 : diagramme processus utilisateur

7 Les divers problèmes rencontrés

Outre les divers problèmes inhérent au développement tel que des problèmes de syntaxe, d'allocation de mémoire... Nous avons rencontrés deux principales difficultés :

- définir ce qui devait être temps réel.
- comprendre la trame reçue .

La partie temps réel est le module d'acquisition uniquement. Au départ, il contenait la traduction complète de la trame, mais pour des raisons de temps d'exécution, il nous est apparu plus judicieux de le faire dans le processus utilisateur.

Le décodage de la trame a occupé une grande partie de notre temps. En effet, nous n'avons pas trouvé de documentation sur la télécommande JVC ni sur leur protocole. Il nous a fallu déduire la fréquence d'échantillonnage et le standard par observation s successives.

8 Codes Sources

8.1 Module d'acquisition temps réel

```
00049 #define MODULE
00050 #include <rtl.h>
00051 #include <time.h>
00052 #include <pthread.h>
00053 #include <rtl_fifo.h>
00054 #include <asm/io.h>
00055
00056 #define FIFO_NUM 0
00057 #define FIFO_SIZE 2048
00058 #define PERIODE 112000
00059 #define ENTETESIZE 32
00060 #define ENTETEZEROSIZE 16
00061 #define VIDESIZE 100
00062 #define MINECHANTILLON 2
00063
00064 pthread_t thread;
00065
00066
00067
00074 void rec bin(char c)
00075 {
00076     static char un = '1';
00077     static char zero = '0';
00078     static int nbZero = 0;
00079
00080     if ( c == '1' )
00081     {
00082         if ( nbZero > 2 && nbZero < ENTETEZEROSIZE )
00083             rtf_put( FIFO_NUM , &un , sizeof( char ) );
00084         else if ( nbZero > 0 && nbZero < ENTETEZEROSIZE )
00085             rtf_put( FIFO_NUM , &zero , sizeof( char ) );
00086         nbZero = 0;
00087     }
00088     if ( c == '0' ) nbZero++;
00089 }
00090
00091
00092
```

```

00098 void reception(char c)
00099 {
00100     static int nbUn    = 0;
00101     static int nbZero = 0;
00102     static int enregistrement = 0;
00103     static char finDeLigne = '\n';
00104
00105     if ( c == '1' )
00106     {
00107         nbZero = 0;
00108         nbUn++;
00109     }
00110     else
00111     {
00112         nbUn = 0;
00113         nbZero++;
00114     }
00115
00116     /* Si une grande succession de zeros, on reinitialise les variables */
00117     if ( nbZero > VIDESIZE )
00118     {
00119         enregistrement = 0;
00120         rtf_put( FIFO_NUM , &finDeLigne , sizeof( char ) );
00121     }
00122
00123     /* On detecte le debut d'un entete */
00124     if ( nbUn > ENTETESIZE ) enregistrement = 1;
00125
00126     if ( enregistrement == 1 ) rec bin(c);
00127 }
00128
00129
00130
00136 void * routine (void *arg)
00137 {
00138     struct sched_param p;
00139     char valeur;
00140     int nbUn = 0;
00141     int nbZero = 0;
00142
00143     /* Parametre du thread */
00144     p.sched_priority = 1;
00145     pthread_setschedparam ( pthread_self(), SCHED_FIFO , &p );
00146     pthread_make_periodic_np ( pthread_self() , gethrtime() , PERIODE );
00147
00148     /* Boucle en lecture */
00149     while( 1 )
00150     {
00151         pthread_wait_np();
00152         valeur = inb(0x379);
00153
00154         /* le bit est a 0 si la valeur lue est egale a 120, 1 sinon */
00155         if ( valeur == 120 ) nbZero++;
00156         else nbUn++;
00157
00158         /* si nombre de 0 lu superieur a echantillonnage, la valeur lue est 0 */
00159         if( nbZero >= MINECHANTILLON )
00160         {
00161             //rtf_put( FIFO_NUM , &zero , sizeof( char ) );
00162             reception( '0' );
00163             nbZero = 0;

```

```

00164         nbUn = 0;
00165     }
00166     /* si nombre de 1 lu superieur a echantillonage, la valeur lue est 1 */
00167     else if ( nbUn >= MINECHANTILLON )
00168     {
00169         //rtf_put( FIFO_NUM , &un , sizeof( char ) );
00170         reception( '1' );
00171         nbZero = 0;
00172         nbUn = 0;
00173     }
00174 }
00175 }
00176
00177
00178
00179
00184 int init module(void)
00185 {
00186     int res;
00187     res = rtf_create( FIFO_NUM , FIFO_SIZE );
00188     if ( res < 0 ) rtl_printf( "erreur de creation du FIFO\n" );
00189
00190     res = pthread_create( &thread , NULL , routine , 0 );
00191     if ( res < 0 ) rtl_printf( "erreur de creation du trhead\n" );
00192
00193     rtl_printf("Driver Recepteur IR demarre... copyright Brice Carbou & Fabrice
Berna");
00194     return 0;
00195 }
00196
00197
00198
00199
00204 void cleanup module(void)
00205 {
00206     rtf_destroy( FIFO_NUM );
00207     pthread_delete_np( thread );
00208     rtl_printf( "Fin du programme...\n" );
00209 }

```

8.2 Processus utilisateur

```

00041 #include <stdio.h>
00042 #include <unistd.h>
00043 #include <fcntl.h>
00044 #include <sys/stat.h>
00045 #include <stdlib.h>
00046
00047 #define FICHER "commandes.jvc"
00048 #define TYPEAPPAREIL 1
00049 #define TYPECOMMANDE 2
00050 #define MODEAPPRENTISSAGE '1'
00051 #define MODENORMAL '2'
00052 #define END '\n'
00053
00054 struct touche
00055 {

```

```

00056 int type; // TYPEAPPAREIL ou TYPECOMMANDE
00057 int code; // Numero de la touche
00058 char libelle[20]; // Libelle de la touche
00059 };
00060
00061 int nbCommandes;
00062
00067 int chargementFichier( struct touche* commande )
00068 {
00069     static FILE* fdesc;
00070     fdesc = fopen( FICHIER , "r" );
00071     if ( fdesc == NULL )
00072     {
00073         system("touch commandes.jvc;");
00074         fdesc = fopen( FICHIER , "r" );
00075         if ( fdesc == NULL )
00076         {
00077             printf( "Impossible d'ouvrir le fichier : %s\n", FICHIER );
00078             return 0;
00079         }
00080     }
00081     nbCommandes = fgetc( fdesc );
00082     fread( commande , sizeof( struct touche ) , nbCommandes , fdesc );
00083
00084     fclose( fdesc );
00085     return 1;
00086 }
00087
00088
00089
00090
00095 int enregistrementFichier( struct touche nvelleTouche[1] , struct touche*
commande )
00096 {
00097     static FILE* fdesc;
00098     static int i,j;
00099     fdesc = fopen( FICHIER , "w+" );
00100     if ( fdesc == NULL )
00101     {
00102         printf( "Impossible d'ouvrir le fichier : %s\n" , FICHIER );
00103         return 0;
00104     }
00105
00106     if ( nvelleTouche == NULL )
00107     {
00108         fputc( nbCommandes , fdesc );
00109         fwrite( commande , sizeof( struct touche ) , nbCommandes , fdesc );
00110         printf("Coucou %d\n",nbCommandes);
00111     }
00112     else
00113     {
00114         fputc( nbCommandes+1 , fdesc );
00115         fwrite( commande , sizeof( struct touche ) , nbCommandes , fdesc );
00116         fwrite( nvelleTouche , sizeof( struct touche ) , 1 , fdesc );
00117     }
00118
00119     fclose( fdesc );
00120     return 1;
00121 }
00122
00123

```

```

00124
00125
00126
00131 int afficheSelonType( int code , int type , struct touche* commande )
00132 {
00133     static int i;
00134     for( i = 0 ; i < nbCommandes ; i ++ )
00135     {
00136         if(( commande[i].type == type ) && ( commande[i].code == code ))
00137         {
00138             printf("%s  ",commande[i].libelle);
00139             return 1;
00140         }
00141     }
00142     return 0;
00143 }
00144
00145
00146
00147
00152 int afficheAppareil( int code , struct touche* commande )
00153 {
00154     printf( " Appareil : " );
00155     return afficheSelonType( code , TYPEAPPAREIL , commande );
00156 }
00157
00158
00159
00160
00165 int afficheCommande( int code , struct touche* commande )
00166 {
00167     printf( " Commande : " );
00168     if( afficheSelonType( code , TYPECOMMANDE , commande ) ) printf("\n");
00169     else return 0;
00170     return 1;
00171 }
00172
00173
00174
00175
00180 int nouvelleEntree( int code , int type , struct touche* commande )
00181 {
00182     char libelle[20];
00183     static struct touche nvlleTouche[1];
00184
00185     /* intialisation du libelle */
00186     sprintf(libelle,"");
00187
00188     printf( "Veuillez saisir son libelle : " );
00189     gets(libelle);
00190
00191     nvlleTouche[0].code = code;
00192     nvlleTouche[0].type = type;
00193     sprintf( nvlleTouche[0].libelle , "%s" , libelle );
00194
00195     return enregistrementFichier( nvlleTouche , commande );
00196 }
00197
00198
00203 int nouvelAppareil( int code , struct touche* commande )
00204 {

```

```

00205 printf( "\n!!! Appareil Inconnu !!! (code %d) \n" , code );
00206 if( !nouvelleEntree( code , TYPEAPPAREIL , commande ) ) return 0;
00207 return chargementFichier( commande );
00208 }
00209
00210
00211
00212
00217 int nouvelleCommande( int code , struct touche* commande )
00218 {
00219     printf( "\n!!! Commande Inconnue !!! (code %d) \n" , code );
00220     if( !nouvelleEntree( code , TYPECOMMANDE , commande ) ) return 0;
00221     return chargementFichier( commande );
00222 }
00223
00224
00225
00226
00227
00228
00234 void bin_code( char c , int reset , struct touche* commande , int mode )
00235 {
00236     static unsigned char masque          = 0x01;
00237     static unsigned char codeCommande   = 0x00;
00238     static unsigned char codeAppareil   = 0x00;
00239     static int    nbBitsRecus    = 0;
00240
00241     /* Reinitialisation des variables */
00242     if ( reset == 1 )
00243     {
00244         codeCommande = 0x00;
00245         codeAppareil = 0x00;
00246         nbBitsRecus  = 0;
00247         return;
00248     }
00249
00250     /* code de l'appareil */
00251     if ( nbBitsRecus < 8 )
00252     {
00253         codeAppareil <<= 1;
00254         if ( c == '1' ) codeAppareil |= masque;
00255     }
00256
00257     /* code de la commande */
00258     else
00259     {
00260         codeCommande <<= 1;
00261         if ( c == '1' ) codeCommande |= masque;
00262     }
00263
00264     nbBitsRecus++;
00265     if ( nbBitsRecus > 15 ) {
00266         nbBitsRecus = 0;
00267         if(!afficheAppareil( (int)codeAppareil , commande ))
00268         {
00269             if ( mode == MODENORMAL ) printf(" Appareil Inconnu ");
00270             else nouvelAppareil( (int)codeAppareil , commande );
00271         }
00272         if(!afficheCommande( (int)codeCommande , commande ))
00273         {
00274             if ( mode == MODEAPPRENTISSAGE )

```

```

                                nouvelleCommande( (int)codeCommande , commande );
00275     else printf("Commande Inconnue\n");
00276     }
00277 }
00278 }
00279
00280
00281
00282
00283 int main () {
00284     static int fd;
00285     static char bit;
00286     static struct touche commande[256];
00287     static char mode;
00288
00289     if (!chargementFichier( commande ))
00290     {
00291         printf("Erreur Chargement Fichier de commandes\n");
00292         return 0;
00293     }
00294
00295     /* Choix du mode d'utilisation */
00296     printf( "Mode d'utilisation ?\n  1 -> Mode Apprentissage\n  2 -> Mode
Normal\n" );
00297     while( (mode!=MODENORMAL) && (mode!=MODEAPPRENTISSAGE) ) gets( &mode );
00298     system( "clear;" );
00299
00300     /* Ouverture de la FIFO et se place en fin de fichier */
00301     fd=open("/dev/rtf0",O_RDONLY);
00302     while ( bit != '\n' ) read ( fd , &bit , sizeof( char ) );
00303
00304     while(1) {
00305         read( fd , &bit , sizeof( char ) );
00306         if ( bit != '\n' )
00307         {
00308             printf( "%c" , bit );
00309             bin_code( bit , 0 , commande , mode );
00310         }
00311         else
00312         {
00313             bin_code( bit , 1 , commande , mode );
00314         }
00315         fflush( stdout );
00316     }
00317     return 0;
00318 }

```