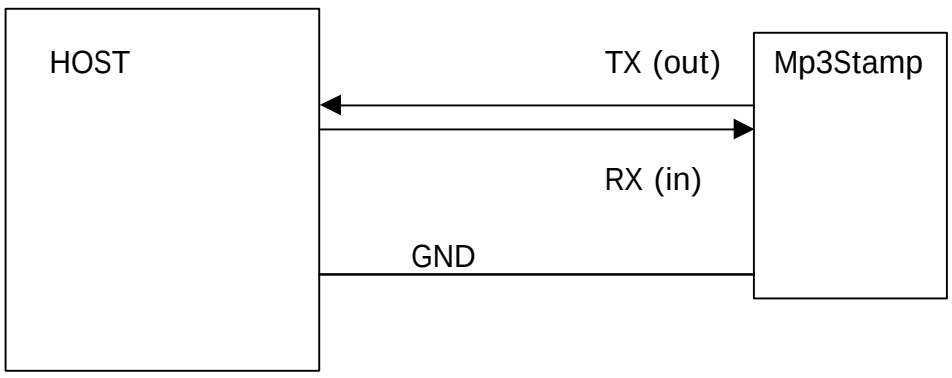
Guide du programmeur

Les modules Mp3Stamp de Reality SYS, ont été conçu pour permettre aux développeurs d'ajouter très facilement voix, effets sonores ou musique de qualité hifi à une vaste gamme d'applications allant des loisirs à la sécurité en passant par l'automobile et la signalisation.
Le choix d'un module intégrant toutes les fonctions de lecture et de décodage simplifie énormément le design hardware et software.
Le module réalise les opérations de lecture sur des cartes standard enregistrée sur PC/Windows ou MAC, il assure le décodage mp3 jusqu'à la sortie analogique prête à être amplifiée ou directement disponible pour une sortie casque.

En termes de ressources hardware, l'application « hote » n'à besoin de rien de plus qu'un port série asynchrone pour piloter le Mp3Stamp.
La communication s'effectuant à basse vitesse, il est même possible d'envisager une liaison série « software » si un UART n'est pas disponible.
Le module décharge le processeur principal de toutes les fonctions en temps réels liées au décodage audio.
L'initialisation est entièrement autonome, et il suffit de transmettre 2 bytes pour démarrer la lecture et le décodage d'un fichier sur la carte.
De plus, le module possède ses propres switches de réglages et une mémoire pour la sauvegarde des paramètres ce qui allège encore la tache du processeur principal.

1 - REALISATION DE L'INTERFACE HARDWARE

La communication est assurée au travers d'une liaison série asynchrone configurée à 9600 bauds, 8 data bits, pas de parité, 1 ou 2 stop bits.
Si le module est connecté directement à un microcontrôleur, il suffit de brancher directement les entrées et sorties RX et TX du module sur le microcontrôleur.
Le module fonctionne en 3.3V mais ses entrées tolèrent des niveaux 5V, il peut être raccordé à un microcontrôleur travaillant en 5V.
Pour connecter le module à un équipement RS232 ou RS485, il faut prévoir un adaptateur de niveaux adéquat.



2 - REALISATION DE L'INTERFACE SOFTWARE

Le contrôle du mp3stamp s'effectue par la transmission de commandes du host vers le module.
Il y a 2 formats de commandes :

FORMAT 1 : <COMMAND BYTE>
FORMAT 2 : <COMMAND BYTE><PARAM BYTE>

Dans le premier cas la commande est constituée d'un unique byte, dans le deuxième le byte de commande est suivi d'un byte de paramètre.
Il n'y à pas de caractère particulier à la fin de la commande (<CR><LF>)

Dans sa version la plus simple, la communication peut être unidirectionnelle, du host vers le module, dans ce cas seul le signal RX(in) est nécessaire.
La majorité des commandes ne produisent pas de réponses de la part du module.

Il peut être nécessaire cependant d'interroger le module et d'utiliser le signal TX(out) Des commandes sont prévues pour fournir l'état du module. Ces commandes sont au format 1, les réponses sont en général formées de 4 ou 8 bytes successifs transmis directement après réception de la commande.

Paramètres de l'interface

MODE BYTE	1BYTE	R/W
NAME	8BYTES	R
TIME	4BYTES	R
CURNT TRACK	1BYTE	R/W

MODE BYTE: bits de statut et de configuration
NAME : contient le nom du fichier courant (mode MSDOS 8 caractères)
TIME : à l'arrêt, contient la durée de la piste courante, en lecture, le temps restant.
CURNT TRACK : le numéro de la piste courante, il est possible de modifier ce pointeur pendant la lecture sans affecter celle ci.

Initialisation

L'initialisation est automatique et ne demande pas d'intervention de la part du host.
A la mise sous tension, le firmware du module détecte la présence d'une carte MMC, parcours le répertoire racine et initialise ses bits d'états à partir des présets sauvés en mémoire flash.
La led est allumée pendant ce test.
Cette opération dure de 1 à 2 sec, ce temps dépend du fait que la carte est présente ou non et du nombre de fichiers sur la carte
Si la carte n'est pas détectée, le circuit émet un beep sur la sortie audio et la led reste allumée. L'erreur est reflétée dans le byte de statut.

Si une carte est détectée, une chaîne de caractère décrivant le système de fichier est directement transmise, si la carte contient au moins un fichier la led s'éteint.

Exemple :

```
Partition type :6
Partition start:32
MSWIN4.1
Sect/cluster  :4
Root :279
```

La chaîne est terminée par le caractère &FF (255).

A partir de ce moment, le circuit est prêt à recevoir des commandes.
Si une carte est présente, le pointeur de piste courante est positionné sur la piste 1 ou la piste définie par une commande BOOT

Si le bit CONTINUE de MODEBYTE le spécifie, la lecture démarre.

A la mise en route ou après une commande RESET, le processeur host doit donc attendre la fin de l'initialisation automatique avant d'envoyer des commandes, pour cela il peut attendre un délai fixe de quelques secondes ou bien attendre la réception du caractère 255 qui suit le descriptif de la partition.

La première commande devrait être (mais ce n'est pas nécessaire) une demande de statut pour s'assurer que la carte est détectée et vérifier le nombre de fichier.
Videz le buffer de réception du host avant d'envoyer cette première commande.

Jouer une piste

Plusieurs commandes permettent de jouer une piste.
La plus simple est la commande PLAY (caractère « O » comme open), elle ne nécessite pas de paramètre, la piste jouée est la piste courante.
La commande « E » exécute la piste qui est fournie comme parametre,
Par contre la commande « D » affecte la valeur du paramètre à CRNT TRACK sans exécuter la lecture.
Rem : la première piste est la piste 1, il n'y a pas de piste 0.

Synchroniser le host avec la lecture

Le processeur host peu interroger périodiquement le MODEBYTE (bit 7) pour déterminer si la lecture est terminée. Il peut également demander le temps restant avant la fin (commande « T »)
Enfin, un byte est automatiquement transmit à la fin de la lecture d'une piste ce byte représente MODEBYTE.

LISTE DES COMMANDES

Les codes des commandes sont donnés en ASCII et en base 10.

Commandes sans paramètres, 1 BYTE de commande pas de réponse	
PLAY : « O» 79	démarre instantanément la piste courante. si un play est envoyé alors que la carte joue, on redémarre au début. annule une pause en cours.
STOP : « H » 72	arrêt instantané et retour au début. annule une pause en cours.
PAUSE : « ! » 33	met la lecture en pause, pas de retour au début. si déjà en pause, reprends la lecture (toggle) pas d'effet si en STOP
ZAP : « Z » 90	passe à la piste suivante et démarre sa lecture. annule une pause en cours.
BOOT : « B » 66	Enregistre la piste courante comme piste de démarrage

Commandes avec 1 paramètre, 1 BYTE de commande, 1 BYTE paramètre, pas de réponse		
GOTO N : « D »,BYTE 68,1-255	positionne la piste courante sur N (1-255) si la piste N n'existe pas, se repositionne sur la piste 1. ne démarre pas la lecture mais après cette commande, SNDTIME peut être utilisé et retourne la durée totale.	
PLAY N : « E »,BYTE 69,1-255	positionne la piste courante sur N (1-255) démarre la lecture. si la piste N n'existe pas, se repositionne sur la piste 1 et ne démarre pas la lecture.	
SETVOL N : « V », BYTE 86,0-255	Affecte la valeur N (0-255) au niveau de sortie audio la valeur est DIRECTEMENT sauvée en flash 0 est le volume max. 255, le minimum. à partir de 80 la sortie est pratiquement inaudible	
SETMODE B : « M »,BYTE 77,0-63	Modifie le mode de fonctionnement, chaque bit de B a une signification	
	B0 : Continue	0: la lecture s'arrête à la fin d'une piste, rien n'est joué à la mise sous tension 1: la lecture continue à la fin d'une piste, repart à la première après la dernière la piste X est jouée à la mise sous tension.
	B1 : Loop	0: les piste s'enchaînent. 1: une seule piste est répétée.
	B2 : Vol LOCAL	0: le volume est contrôlé au niveau du module 1: le volume ne peut pas être réglé au niveau du module
	B3 : Trig LOCK	1: le redéclenchement d'une séquence est impossible 0: mode normal
	B4 : Led ! écriture seule	0: Led OFF 1: Led ON (utiliser cette fonction à l'arret)
	B5: Tri	0: Normal 1:Indexé (A0000)
	B6 ! Lect. seule	1: ERR CARTE MMC
	B7 ! Lect. seule	0 : STOP 1 : PLAYING
	La valeur du byte de mode est DIRECTEMENT sauvée en flash	

Rem : pour que le mode LOOP soit effectif, il faut que le mode CONTINUE soit également activé.

Commandes impliquant une réponse, 1 BYTE de commande, 4 ou 8 BYTES de réponse (nb bytes indéterminé pour commande List)	
SNDSTAT : « S » 83	Demande de statut, la réponse retourne l'état du player 4 BYTES renvoyés <mode><volume><piste courante><nbfiles> <mode> voir SETMODE <volume> 0=max à 255=mute <piste> 1 à nbfiles (<=255) <nbfiles> nombre de fichiers sur la carte
SNDNAME :« I » 73	Demande le titre de la piste courante 8 BYTES renvoyés (ASCII)
SNDTIME : « T » 84	Retourne le compteur temporel en 1/100 de secondes. - à l'arret suite à une commande D, durée totale. - en PLAY ou PAUSE, durée restante. 4 BYTES renvoyés, le byte de poids le plus faible en premier Il est possible de ne lire que les 2 premiers bytes. (si < 10 min)
LIST : « L » 76	Enumère la liste de tous les fichiers présents sur la carte ainsi que leur taille

ANNEXE : Exemple de code source utilisant le protocole.

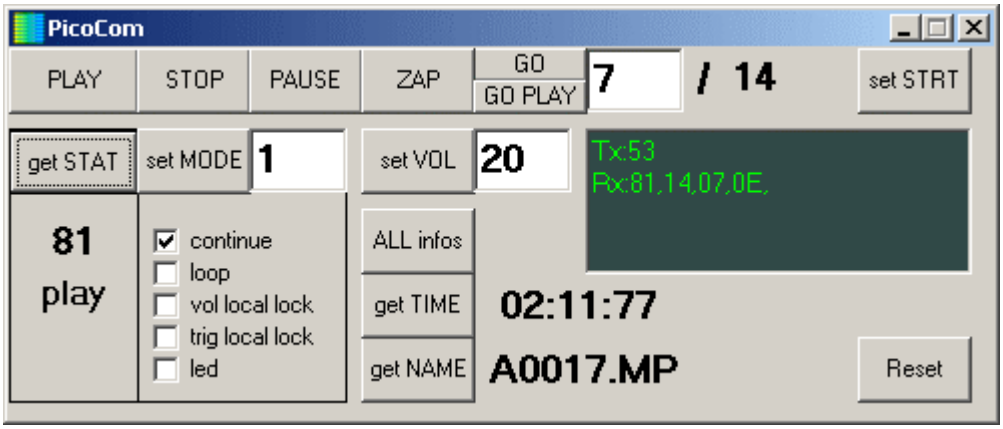
```
//---------------------------------------------------------
//---------------------------------------------------------
// PLAY
void __fastcall TForm1::Button1Click(TObject *Sender)
{
        Memo1->Text="Tx:" + IntToHex(byte('O'),2);
        ComPort1->Send("O");
}
//---------------------------------------------------------
// PAUSE
void __fastcall TForm1::Button2Click(TObject *Sender)
{
        Memo1->Text="Tx:" + IntToHex(byte('!'),2);
        ComPort1->Send("!");
}
//---------------------------------------------------------
// STOP
void __fastcall TForm1::Button4Click(TObject *Sender)
{
        Memo1->Text="Tx:" + IntToHex(byte('H'),2);
        ComPort1->Send("H");
}
//---------------------------------------------------------
// ZAP
void __fastcall TForm1::Button10Click(TObject *Sender)
{
        Memo1->Text="Tx:" + IntToHex(byte('Z'),2);
        ComPort1->Send("Z");
}
//---------------------------------------------------------
// GOTO N
void __fastcall TForm1::Button5Click(TObject *Sender)
{    char n;
            n=Edit1->Text.ToInt();
            Memo1->Text="Tx:" + IntToHex(byte('D'),2)+ "," +IntToHex(n,2);
            ComPort1->Send("D");
            ComPort1->Send(n);
}
//---------------------------------------------------------
// GOTO N et PLAY
void __fastcall TForm1::Button12Click(TObject *Sender)
{
   char n;
            n=Edit1->Text.ToInt();
            Memo1->Text="Tx:" + IntToHex(byte('E'),2)+ "," +IntToHex(n,2);
            ComPort1->Send("E");
            ComPort1->Send(n);
}
//---------------------------------------------------------
// SET VOLUME N
void __fastcall TForm1::Button3Click(TObject *Sender)
{    char n;
            n=Edit2->Text.ToInt();
            Memo1->Text="Tx:" + intToHex(byte('V'),2)+","+IntToHex(n,2);
            ComPort1->Send("V");
            ComPort1->Send(n);
```

```
}
//---------------------------------------------------------
// SET MODE N
void __fastcall TForm1::Button9Click(TObject *Sender)
{    char n;
            n=Edit3->Text.ToInt();
            ComPort1->Send("M");
            ComPort1->Send(n);
}
//---------------------------------------------------------
// SET STARTING TRACK
void __fastcall TForm1::Button13Click(TObject *Sender)
{
            Memo1->Text="Tx:" + IntToHex(byte('B'),2);
            ComPort1->Send("B");

}
//---------------------------------------------------------
// SOFT RESET
void __fastcall TForm1::Button14Click(TObject *Sender)
{
        Memo1->Text="Tx:" + IntToHex(byte('X'),2);
        ComPort1->Send("X");

}
//---------------------------------------------------------
// REQUETE STATUS
void __fastcall TForm1::Button6Click(TObject *Sender)
{        last_cmnd=1;
         Memo1->Text="Tx:" + IntToHex(byte('S'),2);
         ComPort1->Send("S");
}
//---------------------------------------------------------
// REQUETE TITRE
void __fastcall TForm1::Button7Click(TObject *Sender)
{        last_cmnd=3;
         Memo1->Text="Tx:" + IntToHex(byte('I'),2);
         ComPort1->Send("I");
}
//---------------------------------------------------------
// REQUETE TEMPS
void __fastcall TForm1::Button8Click(TObject *Sender)
{       last_cmnd=2;
        Memo1->Text="Tx:" + IntToHex(byte('T'),2);
        ComPort1->Send("T");
}
```

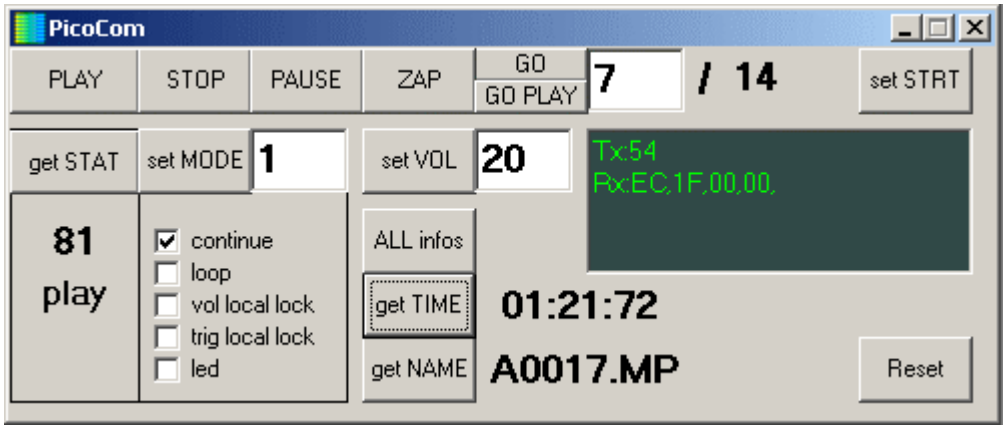
```
//---------------------------------------------------------------
//---------------------------------------------------------------
// Interruption réception série
void __fastcall TForm1::ComPort1ReceiveCallBack(AnsiString Data)
{       int temp;
        int sec;
        int ms;
        int min;
        AnsiString s;
        char* cp=new char[16]; // buffer de réception

        if(last_cmnd!=0){

        cp=Data.c_str();
        Memo1->Lines->Add("Rx:");

        for(temp=0;temp<4;temp++) Memo1->Text=
            Memo1->Text+IntToHex(byte(cp[temp]),2)+",";
// STATUS

        if(last_cmnd==1) {
                Label1->Caption =  IntToHex(byte(cp[0]),2);     // MODE B
                Edit2->Text =  byte(cp[1]);                     // VOLUME
                Edit1->Text =  byte(cp[2]);                     // TRACK nb
                Label2->Caption =  byte(cp[3]);                 // NBFILES

// Mise à jour des bits de mode
                CheckBox1->Checked=((cp[0] & 1)==1);
                CheckBox2->Checked=((cp[0] & 2)==2);
                CheckBox3->Checked=((cp[0] & 4)==4);
                CheckBox4->Checked=((cp[0] & 8)==8);
                CheckBox5->Checked=((cp[0] & 16)==16);

                if((cp[0] & 128)==128)
                                {Label6->Caption="play";}
                        else {Label6->Caption="stop";};
                if((cp[0] & 64)==64) {Label6->Caption="ERR";};

                };

// TIME, seul les 2 premiers bytes utilisés

      if(last_cmnd==2){             temp=byte(cp[0])+256*byte(cp[1]);
                sec=temp/100;
                ms=temp-sec*100;
                min=sec / 60;
                sec=sec % 60;
                s.sprintf ("%2.2u:%2.2u:%2.2u",min,sec,ms);
                Label4->Caption=s;
                };

// TITRE
          if(last_cmnd==3){                 for(temp=4;temp<8;temp++)
          Memo1->Text=Memo1-Text+IntToHex(byte(cp[temp]),2)+",";
                Label5->Caption =Data;};
        };

        last_cmnd=0;
        wait_reponse=false;
        delete[] cp;
}
```

```
//---------------------------------------------------------------
//---------------------------------------------------------------
// Changement des check_box des bits de MODE
void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{if (CheckBox1->Checked ) {mode_byte=mode_byte | 1;}
 else {mode_byte=mode_byte & 254;}  ;
Edit3->Text=int(mode_byte);
}
//---------------------------------------------------------------
void __fastcall TForm1::CheckBox2Click(TObject *Sender)
{if (CheckBox2->Checked ) {mode_byte=mode_byte | 2;}
 else {mode_byte=mode_byte & 253;}  ;
Edit3->Text=int(mode_byte);
}
//---------------------------------------------------------------
void __fastcall TForm1::CheckBox3Click(TObject *Sender)
{if (CheckBox3->Checked ) {mode_byte=mode_byte | 4;}
 else {mode_byte=mode_byte & 251;}  ;
Edit3->Text=int(mode_byte);
}
//---------------------------------------------------------------
void __fastcall TForm1::CheckBox4Click(TObject *Sender)
{if (CheckBox4->Checked ) {mode_byte=mode_byte | 8;}
 else {mode_byte=mode_byte & 247;}  ;
Edit3->Text=int(mode_byte);
}
//---------------------------------------------------------------
void __fastcall TForm1::CheckBox5Click(TObject *Sender)
{if (CheckBox5->Checked ) {mode_byte=mode_byte | 16;}
 else {mode_byte=mode_byte & 239;}  ;
Edit3->Text=int(mode_byte);
}
//---------------------------------------------------------------
//---------------------------------------------------------------
// Demande toutes les infos; status,nom,durée
// utilise un timer pour enchainer les 3 commandes
void __fastcall TForm1::Button11Click(TObject *Sender)
{
  Button11->Enabled=false;
  com_step=1;
  wait_reponse=true;
  Button6Click(Form1); // STAT
  Timer1->Interval =10;
  Timer1->Enabled =true;
}
// timer pour la commande all infos
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{  if (wait_reponse==false){
          wait_reponse=true;
          if (com_step==1)  Button8Click(Form1); // TIME
          if (com_step==2)  Button7Click(Form1);// NAME
          com_step++;
        };

   if (com_step==3) {Timer1->Enabled =false;
   Button11->Enabled=true;};

}
//--FIN-----------------------------------------------------------
```

Envoi d'une commande SNDSTAT ('S' ou 83 ou 53h)
Réponse : 4 bytes MODE = 81h
VOL = 14h (20)
TRACK = 07h (7)
nbTRACKS = 0Eh (14)



Envoi d'une commande SNDTIME ('T' ou 84 ou 54h)
Réponse : 4 bytes 00 00 1F EC = 8172 * 0.01s



Envoi d'une commande PLAY 9 ('E' ou 69 ou 45h)