arduino, audio, lecteur-son, em

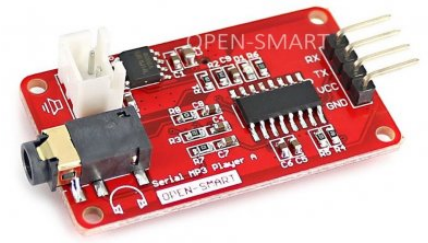
Sonorisation d'un hache lande

/!\ en cours de rédaction 3 mars 2020

Ce projet a été utilisé dans l'exposition «Landes» à l'écomusée des Monts d'Arrée en 2018. La roue du hache lande est utilisée pour lancer la lecture d'une bande sonore : un chant du travail aux champs. Pour cela, un disque en carton avec des aimants intégrés est placé autour de l'axe du hache lande. En tournant la roue d'un demi tour, les aimants passent devant un capteur à effet Hall, quand plusieurs aimants sont détectés, la lecture démarre.

Pour ce projet, nous avons essayé plusieurs modules capables de lire des fichiers sons sur carte micro SD. Plusieurs choses les différencient : sortie amplifiée ou pas, circuit intégré de décodage, coût, bibliothèque associée. Notre choix s'est porté sur le module open-smart serial mp3 player A. La communicaton avec arduino se fait par un port série logiciel, de simples commandes sont envoyées sans utiliser de bibliothèque

Module de lecture MP3 Open-smart serial mp3 player A



doc : open-smart serial MP3 player A manual v1.1 (pdf)

Difficile de dire quel puce est utilisée sur ce circuit, les inscriptions ont été effacée, il s'agit peut-être d'une YX6100-16S ou YX6200-16S de Flyron Technology, ou d'une WT2003-16S de Shenzen Waytronic Electronics. En tout cas, le déclenchement de la lecture se fait en envoyant un message correctement formaté par le port série d'arduino vers le module mp3. Le manuel donne le détail des différents modes de commande possible et des messages que l'on peut envoyer.

Dans notre cas, il y a un seul son à lire quand les conditions sont remplies, cad. qu'un demi tour de la roue a été détecté. La lecture se lance uniquement si elle n'est pas déjà lancée, pour cela on bloque l'arduino en utilisant la fonction delay() ajustée à la durée du son à lire.

Code final

circuit_expo_landes_hache_lande_opensmart.ino (cliquer pour afficher le code)

circuit_expo_landes_hache_lande_opensmart.ino

```
/* Sonorisation du hache lande
 * Quimper, 22 avril 2018, pierre <at> lesporteslogiques.net
 *
 * carte arduino nano robotdyn.com / arduino 1.8.2 @ zibu, debian 7
 *
 * le circuit utilise un capteur à effet Hall 49E (sortie analogique)
 * et un lecteur de MP3 série open smart "Serial MP3 Player A"
 */

boolean DEBUG = true;         // afficher les messages de débugage sur le port série hardware

#include <SoftwareSerial.h>

const byte BROCHE_RX    = 8;
const byte BROCHE_TX    = 9;
const byte BROCHE_HALL  = A6;

byte aimant_compteur  = 0;     // conserve le nombre d'aimants détectés pour mesurer la rotation
```

```
byte aimant_declenche = 4;     // Combien d'aimants pour déclencher la lecture ?
boolean aimant_etat;           // état actuel de l'aimant
boolean aimant_etat_precedent; // état précédent de l'aimant
long last_aimant      = 0;     // Conserve le moment du dernier aimant détecté

SoftwareSerial myMP3(BROCHE_RX, BROCHE_TX);//create a myMP3 object

static int8_t Send_buf[6] = {0} ;
/************Command byte**************************/
/*basic commands*/
#define CMD_PLAY  0X01
#define CMD_PAUSE 0X02
#define CMD_NEXT_SONG 0X03
#define CMD_PREV_SONG 0X04
#define CMD_VOLUME_UP   0X05
#define CMD_VOLUME_DOWN 0X06
#define CMD_FORWARD 0X0A // >>
#define CMD_REWIND  0X0B // <<
#define CMD_STOP 0X0E
#define CMD_STOP_INJECT 0X0F//stop interruptting with a song, just stop the interlude

/*5 bytes commands*/
#define CMD_SEL_DEV 0X35
#define DEV_TF 0X01
#define CMD_IC_MODE 0X35
#define CMD_SLEEP   0X03
#define CMD_WAKE_UP 0X02
#define CMD_RESET   0X05

/*6 bytes commands*/
#define CMD_PLAY_W_INDEX   0X41
#define CMD_PLAY_FILE_NAME 0X42
#define CMD_INJECT_W_INDEX 0X43

/*Special commands*/
#define CMD_SET_VOLUME 0X31
#define CMD_PLAY_W_VOL 0X31

#define CMD_SET_PLAY_MODE 0X33
#define ALL_CYCLE 0X00
#define SINGLE_CYCLE 0X01

#define CMD_PLAY_COMBINE 0X45//can play combination up to 15 songs

void sendCommand(int8_t command, int16_t dat );


void setup() {
  pinMode(BROCHE_HALL, INPUT);

  if (DEBUG) {
    Serial.begin(9600);
    while (!Serial) ;                     // wait for Arduino Serial Monitor
  }
  Serial.println("hello");

  myMP3.begin(9600);
  delay(500);                           // Attendre l'initialisation complète de la puce
  sendCommand(CMD_SEL_DEV, DEV_TF);   // Sélectionner la carte microSD
  delay(200);                           //wait for 200ms

}
void loop() {
    boolean START = false;                 // Va t'il falloir déclencher la lecture du son ?

  aimant_etat_precedent = aimant_etat;   // Mémoire de l'état précédent

/*
  int ttt = analogRead(BROCHE_HALL);
  Serial.println(ttt);
  delay(50);
*/

  // Détecte t'on quelque chose ?

  if (analogRead(BROCHE_HALL) > 500) {   // aimant détecté
    last_aimant = millis();
    if (DEBUG) Serial.println("aimant détecté!");
    aimant_etat = true;
  } else {
    //if (DEBUG) Serial.println(analogRead(BROCHE_HALL));
    aimant_etat = false;
    //delay(100);
  }

  // Selon l'état actuel et l'état précédent de l'aimant on ajoute un au compteur

  if (aimant_etat && !aimant_etat_precedent && (millis() - last_aimant < 3000)) {
    aimant_compteur ++;
    if (DEBUG) Serial.println("passage d'aimant détecté!");
    if (DEBUG) Serial.println(aimant_compteur);
  }

  // Si c'est trop long entre 2 passages, on remet le compteur à zéro
```

```
  if ((millis() - last_aimant) > 3000) aimant_compteur = 0;

  // A t'on atteint le nombre de passage pour déclencher le son ?

  if (aimant_compteur >= aimant_declenche) {
    START = true;
    aimant_compteur = 0;
   }


  if (START) {
    playWithVolume(0X1E01);   // play the first song with volume 30(0x1E)
    delay(147308);            // Oh! Ca, c'est tricher, correspond à la durée du son...
  }

}

void setVolume(int8_t vol)
{
  mp3_5bytes(CMD_SET_VOLUME, vol);
}
void playWithVolume(int16_t dat)
{
  mp3_6bytes(CMD_PLAY_W_VOL, dat);
}

/*cycle play with an index*/
void cyclePlay(int16_t index)
{
  mp3_6bytes(CMD_SET_PLAY_MODE, index);
}

void setCyleMode(int8_t AllSingle)
{
  mp3_5bytes(CMD_SET_PLAY_MODE, AllSingle);
}


void playCombine(int8_t song[][2], int8_t number)
{
  if (number > 15) return; //number of songs combined can not be more than 15
  uint8_t nbytes;//the number of bytes of the command with starting byte and ending byte
  nbytes = 2 * number + 4;
  int8_t Send_buf[nbytes];
  Send_buf[0] = 0x7e; //starting byte
  Send_buf[1] = nbytes - 2; //the number of bytes of the command without starting byte and ending byte
  Send_buf[2] = CMD_PLAY_COMBINE;
  for (uint8_t i = 0; i < number; i++) //
  {
    Send_buf[i * 2 + 3] = song[i][0];
    Send_buf[i * 2 + 4] = song[i][1];
  }
  Send_buf[nbytes - 1] = 0xef;
  sendBytes(nbytes);
}


void sendCommand(int8_t command, int16_t dat = 0)
{
  delay(20);
  if ((command == CMD_PLAY_W_VOL) || (command == CMD_SET_PLAY_MODE) || (command == CMD_PLAY_COMBINE))
    return;
  else if (command < 0x10)
  {
    mp3Basic(command);
  }
  else if (command < 0x40)
  {
    mp3_5bytes(command, dat);
  }
  else if (command < 0x50)
  {
    mp3_6bytes(command, dat);
  }
  else return;

}

void mp3Basic(int8_t command)
{
  Send_buf[0] = 0x7e; //starting byte
  Send_buf[1] = 0x02; //the number of bytes of the command without starting byte and ending byte
  Send_buf[2] = command;
  Send_buf[3] = 0xef; //
  sendBytes(4);
}
void mp3_5bytes(int8_t command, uint8_t dat)
{
  Send_buf[0] = 0x7e; //starting byte
  Send_buf[1] = 0x03; //the number of bytes of the command without starting byte and ending byte
  Send_buf[2] = command;
  Send_buf[3] = dat; //
  Send_buf[4] = 0xef; //
```

```
  sendBytes(5);
}
void mp3_6bytes(int8_t command, int16_t dat)
{
  Send_buf[0] = 0x7e; //starting byte
  Send_buf[1] = 0x04; //the number of bytes of the command without starting byte and ending byte
  Send_buf[2] = command;
  Send_buf[3] = (int8_t)(dat >> 8);//datah
  Send_buf[4] = (int8_t)(dat); //datal
  Send_buf[5] = 0xef; //
  sendBytes(6);
}
void sendBytes(uint8_t nbytes)
{
  for (uint8_t i = 0; i < nbytes; i++) //
  {
    myMP3.write(Send_buf[i]) ;
  }
}
```