Laser-wave

2 Av. Augustin Fresnel,
91120 Palaiseau
(33) 01 23 45 67 89

Rapport technique -Annexe

Code Mbed commenté lié à la Conversion MIDI/DMX

```
//on appelle les libraires dont on va avoir besoin
#include "mbed.h"
#include "platform/mbed_thread.h"
//Ce tableau constitue la liste des couleurs prédéfinie que nous allons utiliser pour nos test, elles peuvent être choisir :
//une colonne correspond à une couleur avec ligne 1 = poids du pixel rouge / ligne 2 poids du pixel vert / ligne 3 poids du pixel bleu. chaque LED peut stocker 12 couleurs.
                                        // LED1                                                                 LED2
        LED3                                                                    LED4
const   uint8_t         scriabin_r[48] = {99 , 59 , 233, 241,   197, 245,  68, 255, 170,   0, 255,
0,      99 , 59 , 233, 241,   197, 245,  169, 8  , 218,   11 , 176,   0  ,          255, 255, 255,
127,   37 , 255,  68, 255, 170,   0, 255,   0,              255, 140, 255, 255,   0, 255,  68, 255,
170,   0, 255,   0};
const   uint8_t         scriabin_g[48] = {218, 249, 215, 56 ,   151, 190,  68, 255, 170,   0,
255,   0,         218, 249, 215, 56 ,   151, 190,  234, 86 , 71 ,   189, 229,   251,       15 , 255,
148, 247,   146, 255,  68, 255, 170,   0, 255,   0,              255, 140, 255, 255,   0, 255,  68,
255, 170,   0, 255,   0};
const   uint8_t         scriabin_b[48] = {239, 168, 64 , 56 ,   228, 232,  68, 255, 170,   0, 255,
0,      239, 168, 64 , 56 ,   228, 232,  198, 186, 91 ,   240, 55 ,   251,       255, 000, 40 ,
154,   255, 255,  68, 255, 170,   0, 255,   0,              255, 140, 255, 255,   0, 255,  68, 255,
170,   0, 255,   0};

//définition du code pour détecter le début et la fin d'une sequence d'une sequence pour les pads et les controleurs

#define         MIDI_NOTE_ON        0x90
#define         MIDI_NOTE_OFF       0x80
#define         MIDI_CC             0xB0

#define SAMPLES     512 //définit la taille d'un bus de data en DMX
//définiton des différents branchement pour la carte nucléo entreée et sortie analogique et digitales ainsi que connection serial port.
Serial   debug_pc(USBTX, USBRX);
InterruptIn       my_bp(USER_BUTTON);
```

```
Serial   dmx(A0, A1);
DigitalOut  out_tx(D5);
DigitalOut  start(D4);   //envoie des données
DigitalOut  enableDMX(D6);
AnalogIn        CV_volume(PC_1);
AnalogIn        CV_pitch(PB_0);

AnalogIn        variationR(PC_0);
AnalogIn        variationG(PC_2);
AnalogIn        variationB(PC_3);

Serial   midi(D8, D2);

// intialisation liée au pilotage DMX
char dmx_data[SAMPLES] = {0}; // initialisation d'un bus de donnée, ce bus est traité comme
une chaine de caractère (512 bits)
char nb = 0;
void initDMX(); // fonction d'initialisation de la liaison
void updateDMX(); // traitement des données

// iinitalisation liée à la commande MIDI
char    cpt_midi;
char    new_data_midi, new_note_midi;
char    midi_data[3], channel_data, note_data, velocity_data;
char    control_ch, control_value;

void initMIDI(void);
void ISR_midi_in(void);
bool isNoteMIDIdetected(void);
bool isNoteCCdetected(void);



// Main
int main() {
        debug_pc.baud(115200);
        debug_pc.printf("Essai DMX512\r\n");

        initDMX();
        initMIDI();

        int i;

        while(1) {
        if(isNoteMIDIdetected()){// partie du pilotage slié aux pads
        debug_pc.printf("C=%d,N=%d,V=%d\r\n", channel_data, note_data, velocity_data);
        char note = note_data%12;
        debug_pc.printf("N=%d\r\n", note);
```

```
// Renkforce LPT12 - AD 1


for(i=0;i<4;i++){



dmx_data[1+i*8] = scriabin_r[note+12*i]; // les canaux  1 2 et 3 sont relié aux pixels rouge vert et bleus dans notre exemple, ces donnés se trouvent dans les datasheets constructeurs. On fait varier i pour progresser dans la liste
dmx_data[2+i*8] = scriabin_g[note+12*i];
dmx_data[3+i*8] = scriabin_b[note+12*i];

}

new_note_midi = 0;
}
if(isNoteCCdetected()){ // partie du pilotage lié au controleurs
debug_pc.printf("C=%d,V=%d\r\n", control_ch, control_value);

//pour les contrôles en translation les controleurs utiles sont les 4 premier, le premier gère l'intenité moyenne de la led à l'adresse choisie,
//et les trois controleurs suivants permettent de contrôler le poids rouge vert et bleu dans la couleur -> cela sert à installer une lumière d'ambiance.

if(control_ch==1){
for(i=0;i<4;i++){
dmx_data[0+i*8] = 0;
dmx_data[3+i*8] = control_value*2+1;
}
}
if(control_ch==2){
for(i=0;i<4;i++){
dmx_data[0+i*8] = 0;
dmx_data[3+i*8] = 100;

dmx_data[4+i*8] = control_value*2+1;
}
}

if(control_ch==3){
for(i=0;i<4;i++){
dmx_data[0+i*8] = 0;
dmx_data[3+i*8] = 100;

dmx_data[5+i*8] = control_value*2+1;
}
}
```

```
        if(control_ch==4){
        for(i=0;i<4;i++){
        dmx_data[0+i*8] = 0;
        dmx_data[3+i*8] = 100;

        dmx_data[6+i*8] = control_value*2+1;
        }
        }

        }
        // pour les controles en rotation : on fait varier les couleurs en fonction de la
fréquence de la musique, en parcourant une liste de couleur pré choisie.
        while(control_ch==12 && control_value!=0){
        for(j=0;j<12;j++){// on parcours les douze couleurs de la lampes ( chaque ligne du
tableau du haut)
        for(i=0;i<4;i++){// on par cours chaque pixel de chaque couleur (colone du tableau du
haut)

        dmx_data[0+i*8] = 0;
        dmx_data[3+i*8] = 255;

        dmx_data[4+i*8] = scriabin_r[j+12*i];
        dmx_data[5+i*8] = scriabin_g[j+12*i];
        dmx_data[6+i*8] = scriabin_b[j+12*i];

        }
        updateDMX();
        w=int((double(control_value)*(-2182)+302182));// on fait varier le temps que dure une
couleur linéairement par rapport à la valeur du controleur
        wait_us(w);             }
        }



        }
        new_data_midi = 0;
        }
        updateDMX();
        wait_us(10000);

        }
}

/* Fonction d'initialisation de la liaison DMX */
void initDMX(){
        // Initialisation DMX
        dmx.baud(250000);
        dmx.format (8, SerialBase::None, 2);
```

```
	enableDMX = 0;
	// Initialisation canaux DMX
	for(int k = 0; k < SAMPLES; k++){
	dmx_data[k] = 0;
	}
	updateDMX();
}

/* Fonction de mise à jour de la liaison DMX */
void updateDMX(){ // Permet d'envoyer la trame complète sur la série associée qui a été initialisée par la fonction init_dmx
	enableDMX = 1;
	start = 1;	// /start
	out_tx = 0;	// break
	wait_us(88);
	out_tx = 1;	// mb
	wait_us(8);
	out_tx = 0;	// break
	start = 0;
	dmx.putc(0);	// Start
	for(int i = 0; i < SAMPLES; i++){
	dmx.putc(dmx_data[i]);		// data
	}
	wait_us(23000); // time between frame
}

/* Fonction d'initialisation de la liaison MIDI */
void initMIDI(void){
	midi.baud(31250);
	midi.format(8, SerialBase::None, 1);
	midi.attach(&ISR_midi_in, Serial::RxIrq);
}
/* Detection d'une note reçue en MIDI */
bool isNoteMIDIdetected(void){
	if(new_note_midi == 1)
	return true;
	else
	return false;
}
/* Detection d'une note reçue en MIDI */
bool isNoteCCdetected(void){
	if(new_data_midi == 1)
	return true;
	else
	return false;
}
/* Fonction d'interruption sur MIDI */
void ISR_midi_in(void){ // Les données sont reçues et traitées par cette fonction
```

```
        char data = midi.getc(); // cahque bit du bus de data midi est stocké dans data
        if(data >= 128)
        cpt_midi = 0;
        else
        cpt_midi++;
        midi_data[cpt_midi] = data; // chaque data est ensuite sotcjé dans le tableau
midi_data
        if(cpt_midi == 2){
        cpt_midi = 0;
        if(((midi_data[0] & 0xF0) == MIDI_NOTE_ON) || ((midi_data[0] & 0xF0) ==
MIDI_NOTE_OFF)){ // on demande ensuite si le bus detectée vient d'un controleur ou d'un
pad, si c'est un pad c'est la premère partie du code
        new_note_midi = 1;
        channel_data = midi_data[0] & 0x0F;
        note_data = midi_data[1]; // Valeur de la note stockée dans le deuxième indice
        velocity_data = midi_data[2]; // Force d'appuie sur la touche entre 0 et 127 stockée
dans le troisième indice
        }
        else{// si c'est un controleur c'est ici
        if(midi_data[0] == MIDI_CC){
                new_data_midi = 1;
                control_ch = midi_data[1];
                control_value = midi_data[2];
        }
        }
        }
}
```

Code Mbed commenté lié à l'interface graphique

```
/*****************************************************************************/
/*  Commande par ordi                                                        */
/*****************************************************************************/
/*  PROTS  / Oscar BOUCHER */

/*****************************************************************************/
/* Inspiré de  LEnsE / Julien VILLEMEJANE           /   Institut d'Optique Graduate School */
/*****************************************************************************/
/*  Brochage                                                                 */
/*         TO COMPLETE                                                       */
/*****************************************************************************/
/*  Test réalisé sur Nucléo-L476RG                                           */
/*****************************************************************************/

#include "mbed.h"
#include "platform/mbed_thread.h"

#define SAMPLES     512
Serial   debug_pc(USBTX, USBRX);
InterruptIn       my_bp(USER_BUTTON);


Serial   dmx(A0, A1);
DigitalOut  out_tx(D5);
DigitalOut  start(D4);   //envoie des données
DigitalOut  enableDMX(D6);
AnalogIn         CV_volume(PC_1);
AnalogIn         CV_pitch(PB_0);

AnalogIn         variationR(PC_0);
AnalogIn         variationG(PC_2);
AnalogIn         variationB(PC_3);
DigitalOut  myled(LED1);
//Serial           midi(D8, D2);



// DMX
char dmx_data[SAMPLES] = {0};
char nb = 0;

void initDMX();
void updateDMX();


//interface
char ad[3];
```

```
char val[3];
char test;
int adresse;
int valeur;

Ticker DMX;

void upDMX(void);




int w;
// Main
int main() {
	debug_pc.baud(115200); // configuration de la liaison

	test = 0;
	initDMX(); //permet d'initialiser la liaison DMX
	DMX.attach(&upDMX,0.3); // on renvoie la trame DMX toute les 0,3s




	while(1) {
	test = debug_pc.getc(); //caractère de début
	if (test == 97){ //a en ASCII

	ad[0]=debug_pc.getc(); // récupération caractère par caractère
	ad[1]=debug_pc.getc();
	ad[2]=debug_pc.getc();

	val[0]=debug_pc.getc();
	val[1]=debug_pc.getc();
	val[2]=debug_pc.getc();


	adresse = ((ad[0]-48)*100+(ad[1]-48)*10+(ad[2]-48)); //conversion en entier
	valeur = ((val[0]-48)*100+(val[1]-48)*10+(val[2]-48));
	test =0; //remise à 0 de la valeur test

	dmx_data[adresse-1]=char(valeur); // le canal 1 correspond à l'indice 0

	updateDMX();

	}
```

```
        wait_us(10000);
        updateDMX();

        }
}



/* Fonction de mise à jour lié au ticker*/
void upDMX(){
        updateDMX();
        }


/* Fonction d'initialisation de la liaison DMX */
void initDMX(){
        // Initialisation DMX
        dmx.baud(250000);
        dmx.format (8, SerialBase::None, 2);
        enableDMX = 0;
        // Initialisation canaux DMX
        for(int k = 0; k < SAMPLES; k++){
        dmx_data[k] = 0;
        }
        updateDMX();
}

/* Fonction de mise à jour de la liaison DMX */
void updateDMX(){ // Permet d'envoyer la trame complète sur la série associée qui a été
initialisée par la fonction init_dmx
        enableDMX = 1;
        start = 1;          // /start
        out_tx = 0;         // break
        wait_us(88);
        out_tx = 1;         // mb
        wait_us(8);
        out_tx = 0;         // break
        start = 0;
        dmx.putc(0);    // Start
        for(int i = 0; i < SAMPLES; i++){
        dmx.putc(dmx_data[i]);          // data
        }
        wait_us(23000); // time between frame
}
```