NOTICE D'UTILISATION : LÉVITATEUR À DEUX PLAQUES

Contrôle Synchrone Maître-Esclave (Prototype 3)

4 février 2026

Sommaire

1 Étapes clés

1. Prise en main des codes Arduino
2. Prise en main pratique

2 Prise en main des codes Arduino

Ce troisième prototype nécessitant l'utilisation de 2 plaques de lévitation contrôlables de façon indépendantes, il est alors nécessaire d'utiliser 2 cartes Arduino et donc d'être capable de les faire fonctionner de façon synchrone. Pour ce faire, nous avons mis en place un système maître-esclave. Les codes utilisés sont présentés en Annexe.

Le fonctionnement de ce système est assez basique : les phases à émettre sur chacune des deux plaques sont stockées "en dur" (dans le code) dans chacune des deux cartes Arduino, ce qui permet de ne pas avoir à gérer le transfert des octets de l'Arduino maître vers l'Arduino esclave.

La communication entre les 2 deux cartes Arduino se résume donc à leur synchronisation sur le signal créneau du pin 2 de la carte maître permettant de cadencer l'émission des signaux pour les 64 transducteurs. Le fonctionnement schématisé de la communication entre les 2 programmes est repris par la figure suivante.

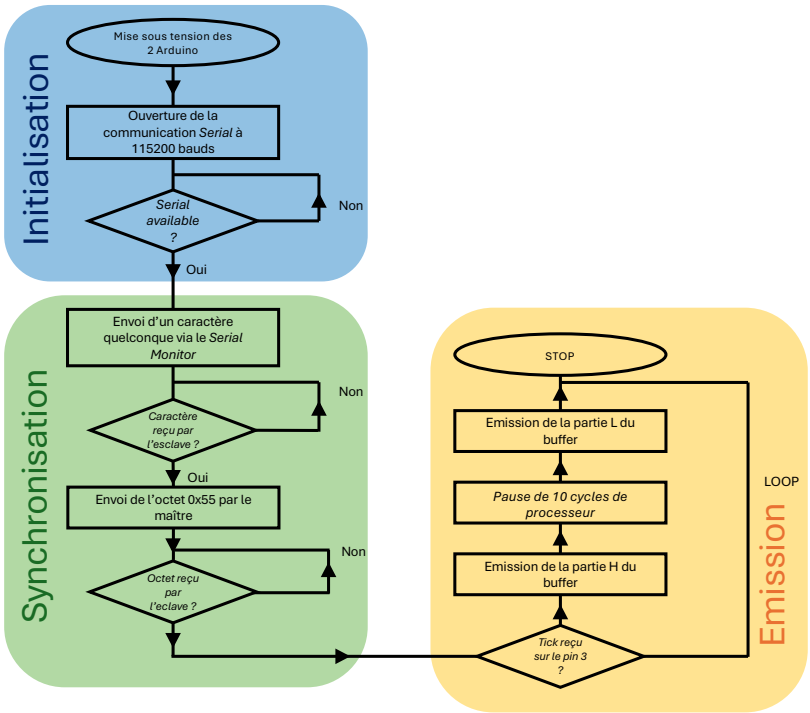

FIGURE 1 – Schéma explicatif de la communication entre les 2 cartes Arduino

2.1 Initialisation

Cette première partie des codes est tout à fait similaire à celle du code NewDriverMega disponible sur GitHub et est commune aux deux cartes Arduino.

Dans cette phase d'initialisation, les deux cartes Arduino configurent les constantes, les différentes variables qui seront utilisées ainsi que l'ensemble des ports parallèles (A, C, L, B, K, F, H, D, G et J) en sortie et les placent à l'état bas afin de garantir un état initial stable pour les transducteurs. Elles effacent ensuite le buffer contenant les données d'émission pour éviter toute valeur résiduelle.

La configuration de l'interruption INT5 (sur la broche 3) est mise en place pour détecter chaque front descendant du signal de synchronisation envoyé par la carte maître ; cette interruption alimente un compteur sync_cnt utilisé comme horloge externe.

Enfin, les modules inutiles du microcontrôleur (ADC, SPI, TWI, certains timers et ports série) sont désactivés afin de réduire la charge et d'assurer un fonctionnement précis et déterministe, ce qui est essentiel pour la génération synchrone des signaux.

Il est à noter que le code maître possède une étape qui est absente dans le code esclave : l'émission du signal de synchronisation à 80 kHz. En effet, la carte esclave n'a pas besoin d'émettre ce signal puisque sa broche 3 est directement connectée à la broche 2 de l'Arduino maître, ce qui lui permet de se synchroniser sur le signal du maître.

> **Remarque :** il est à noter que nous avons fait le choix d'utiliser un signal de synchronisation à 80 kHz et non plus à 40 kHz comme précédemment. Cela résulte uniquement de contraintes informatiques liées au cadencement du processeur de la carte Arduino et permet d'obtenir, en utilisant la méthode ci-après présentée, un signal à 40 kHz pour les 64 transducteurs

2.2 Synchronisation

La synchronisation des deux cartes débute par un court protocole d'amorçage sur la liaison série. Avant tout échange automatique, la carte maître attend qu'un caractère (quelconque) soit envoyé depuis le moniteur série de l'utilisateur ; ce premier envoi agit comme un déclencheur manuel permettant de signaler que l'opérateur est prêt et que le système peut s'amorcer.

Lorsque ce caractère reçu, le maître transmet alors l'octet 0x55 à l'esclave. De son côté, l'esclave reste bloqué tant qu'il n'a pas reçu cette valeur, ce qui garantit qu'il ne démarre pas prématurément. Une fois 0x55 reconnu, l'esclave confirme implicitement sa disponibilité en allumant sa LED 13 et les deux cartes basculent dans leur boucle synchronisée.

À partir de ce point, l'horloge commune est fournie par le maître sous forme d'un signal carré de 80 kHz généré par le Timer 3. Chaque front descendant de ce signal, détecté sur INT5, incrémente sur les deux cartes le compteur sync_cnt, qui sert alors de référence temporelle partagée. Grâce à ce double mécanisme — déclenchement manuel, handshake maître-esclave via 0x55, puis synchronisation par horloge matérielle — les deux systèmes

démarrent exactement en même temps et exécutent ensuite leurs mises à jour de sorties en parfaite simultanéité.

2.3 Emission

Une fois la synchronisation établie, les deux cartes peuvent commencer la phase d'émission. Chacune dispose d'un tableau buffer contenant les états logiques à appliquer successivement aux dix ports parallèles raccordés aux transducteurs. Ces valeurs représentent les motifs d'excitation ultrasonore en haute résolution temporelle.

Lorsqu'un front descendant du signal de synchronisation est détecté, la fonction wait_ticks() assure que chaque mise à jour survient exactement au bon instant, garantissant un timing strictement identique entre le maître et l'esclave. À chaque tick, les cartes envoient simultanément les tranches successives du motif grâce à la macro OUTPUT_WAVE(), qui écrit en une seule instruction l'ensemble des dix octets sur les ports matériels.

L'organisation en pointeurs *emittingPointerH* et *emittingPointerL* permet de structurer les données en deux blocs correspondant aux étapes hautes et basses du motif. Ce mécanisme d'émission, basé sur l'écriture directe dans les registres et synchronisé par une horloge externe, assure une génération parfaitement alignée des ondes ultrasonores, condition indispensable pour obtenir un champ acoustique cohérent et stable.

3 Prise en main pratique

> L'utilisation des deux Arduino n'étant pas à cette étape du projet optimale, il faut absolument veiller à réaliser les étapes suivantes dans l'ordre et en respectant précisément les recommandations à venir.

3.1 Téléversement des codes sur les 2 Arduino

La première étape consiste à téléverser les codes sur les 2 Arduino. Pour ce faire, il faut tout d'abord avoir repéré quel Arduino est connecté à quel port COM de l'ordinateur. Pour obtenir cette information, il suffit de connecter séparément chaque Arduino afin d'identifier à quel port il est associé. Une fois cette information obtenue, vous pouvez téléverser le code Arduino Maître sur l'Arduino contrôlant la plaque du bas et le code Esclave sur l'Arduino contrôlant la plaque du haut.

3.2 Connexion des deux Arduino

Il est maintenant temps de connecter l'Arduino maître (contrôlant la plaque du bas) à l'Arduino esclave (contrôlant la plaque du haut). Pour ce faire, il suffit de faire correspondre le connecteur mâle à 6 pins de la carte maître sur le connecteur femelle à 6 pins de la carte esclave en s'assurant que l'on connecte bien le port TX de la carte maître au port RX de la carte esclave (cf. gravures à l'arrière du PCB). Il est également possible d'utiliser un câble en nappe disposant de 6 connecteurs mâle et 6 connecteur femelle, ce qui permet d'augmenter la flexibilité de l'installation en réduisant les contraintes liées à la connexion des deux cartes. Cette solution est présentée en figure suivante.

> Remarque : il est impossible de téléverser le code sur l'Arduino esclave lorsque ce dernier a été connecté à l'Arduino maître via son port RX car l'utilisation de ce dernier bloque l'accès à l'ordinateur au port série UART.

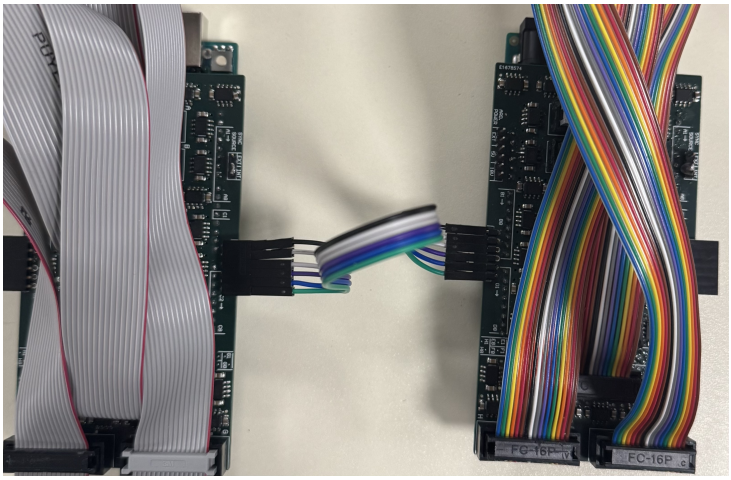

FIGURE 2 – Connexion physique des deux cartes Arduino

3.3 Lancement de l'émission sur les deux matrices

Une fois les connexions réalisées, il ne reste qu'à lancer l'émission des signaux acoustiques sur les 2 matrices de 64 transducteurs. Pour cela, il suffit d'envoyer via le moniteur série de l'Arduino maître un caractère quelconque (lettre ou chiffre) à l'Arduino esclave. L'envoi de ce caractère permet de synchroniser les 2 Arduino et d'initier l'émission des signaux.

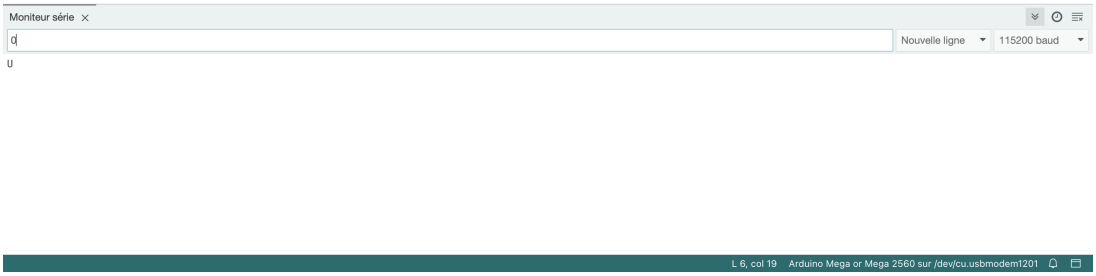

FIGURE 3 – Interface du moniteur série pour l'initialisation

3.4 Alimentation du dispositif

L'étape finale consiste à alimenter le dispositif avec une tension continue comprise entre 25 et 35 Vpp.

Pour cela, connecter d'abord les fiches banane noire et rouge du câble d'alimentation respectivement sur la borne et la borne + de l'alimentation continue.

Ensuite, connecter la fiche jack du câble d'alimentation à l'un des deux shield d'amplification (maître ou esclave, les deux étant interconnectés). Pour cette étape, se référer au paragraphe "Alimentation du Driver Board" de la notice d'utilisation du prototype à

une plaque disponible sur notre GitHub. Enfin, allumer l'alimentation, régler l'intensité environ au tiers du maximum et enfin régler la tension à environ 25 V.

> Remarque : par expérience, il est possible qu'un court-circuit apparaisse à la suite d'une mauvaise manipulation du matériel. Ce dernier est souvent repérable à l'aide de l'alimentation continue : si le voyant "CC" de cette dernière est allumé ou l'on entend les calibres de cette dernière passer les uns à la suite des autres, l'éteindre immédiatement afin de préserver le matériel.

Il ne reste plus qu'à faire léviter !

3.5 Procédure d'arrêt

Pour éteindre le système, il suffit de commencer par éteindre l'alimentation continue, puis de déconnecter les Arduino de l'ordinateur.

A Annexes techniques

A.1 Communication à deux cartes Arduino

A.1.1 Carte *maître*

Voici le code Arduino utilisé pour le contrôle de la carte *maître* :

```
#include <avr/sleep.h>
#include <avr/power.h>

#define N_PATTERNS 32
#define N_TRANSDUCTEUR 80
#define N_PORTS 10
#define N_DIVS 10
#define N_T 8

//ports A C L B K F H D G J

#define WAIT() __asm__ __volatile__("nop\n\t""nop\n\t""nop\n\t""nop\n\t""nop\n\t""nop\n\t""nop\n\t""nop\n\t""nop\n\t""nop\n\t")
#define OUTPUT_WAVE(pointer, d)  PORTA = pointer[d*N_PORTS + 0]; PORTC = pointer[d*N_PORTS + 1]; PORTL = pointer[d*N_PORTS + 2]; PORTB = pointer[d*N_PORTS + 3]; PORTK = pointer[d*N_PORTS + 4]; PORTF = pointer[d*N_PORTS + 5]; PORTH = pointer[d*N_PORTS + 6];  PORTD = pointer[d*N_PORTS + 7]; PORTG = pointer[d*N_PORTS + 8]; PORTJ = pointer[d*N_PORTS + 9]

static byte buffer[ N_DIVS * N_PORTS];
volatile uint8_t sync_cnt = 0;  // compteur de fronts sur pin 3 (INT5)

// Attente robuste de N ticks (gere le wrap 0..255 et les ticks manques)
static inline void wait_ticks(uint8_t n, volatile uint8_t* last_seen) {
  uint8_t start = *last_seen;
  while ( (uint8_t)(sync_cnt - start) < n ) { /* spin, interruptions actives */ }
  *last_seen = sync_cnt;
}

static inline void wait_substep(volatile uint16_t* tref, uint16_t ticks)
{
  uint16_t start = *tref;
  while ((uint16_t)(TCNT4 - start) < ticks) {
    // attente active, base de temps = Timer4
  }
  *tref = start + ticks;
}

void setup()
{
  //set as output ports A C L B K F H D G J
  DDRA = DDRC = DDRL = DDRB = DDRK = DDRF = DDRH = DDRD = DDRG = DDRJ = 0xFF;
  //low signal on all of them
  PORTA = PORTC = PORTL = PORTB = PORTK = PORTF = PORTH = PORTD = PORTG = PORTJ = 0x00;

  //clear the buffers
  for (int i = 0; i < (N_PATTERNS * N_DIVS * N_PORTS); ++i) {
    buffer[i] = 0;
  }

  // INT5 (PE5 = pin 3 sur Mega) sur FRONT DESCENDANT
  EICRB |= (1 << ISC51);   // ISC51:50 = 10 => front descendant
  EICRB &= ~(1 << ISC50);
  EIFR   = (1 << INTF5);   // clear tout pending
  EIMSK |= (1 << INT5);    // enable INT5

  // generate a sync signal of 40khz in pin 2
  pinMode (2, OUTPUT);
  noInterrupts();           // disable all interrupts
  TCCR3A = bit (WGM10) | bit (WGM11) | bit (COM1B1); // fast PWM, clear OC1B on compare
  TCCR3B = bit (WGM12) | bit (WGM13) | bit (CS10);   // fast PWM, no prescaler
  OCR3A =  (F_CPU / 80000L) - 1;
  OCR3B = (F_CPU / 80000L) / 2;
```

```
  TCCR4A=0;
  TCCR4B=(1<<CS40);
  TCNT4=0;

  interrupts();             // enable all interrupts

  //sync in signal at pin 3
  pinMode(3, INPUT_PULLUP); //please connect pin3 to pin 2


  // disable everything that we do not need
  ADCSRA = 0;  // ADC
  power_adc_disable ();
  power_spi_disable();
  power_twi_disable();
  power_timer0_disable();
  power_usart1_disable();
  power_usart2_disable();
  power_usart3_disable();

//creation d'une liste de valeur binaire
  byte buffer[N_DIVS * N_PORTS] = {
  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x7B, 0x8F, 0x07, 0x03,
  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x7B, 0x8F, 0x07, 0x03,
  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x7B, 0x8F, 0x07, 0x03,
  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x7B, 0x8F, 0x07, 0x03,
  0x42, 0x32, 0xCB, 0x4D, 0x5A, 0xBD, 0x10, 0x82, 0x00, 0x01,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0xBD, 0xCD, 0x34, 0xB2, 0xA5, 0x42, 0x6B, 0x0D, 0x07, 0x02
};


//initialisation des variables
  byte* emittingPointerH = & buffer[0];
  byte* emittingPointerL = & buffer[N_PORTS * N_DIVS / 2];

  Serial.begin(115200);
  while(!Serial.available());
  Serial.write(0x55);


//------boucle-------------------------

LOOP:
{
  uint8_t last = sync_cnt;

  wait_ticks(1,&last);
  OUTPUT_WAVE(emittingPointerH, 0); WAIT();
  OUTPUT_WAVE(emittingPointerH, 1); WAIT();
  OUTPUT_WAVE(emittingPointerH, 2); WAIT();
  OUTPUT_WAVE(emittingPointerH, 3); WAIT();
  OUTPUT_WAVE(emittingPointerH, 4); WAIT();
  OUTPUT_WAVE(emittingPointerL, 0); WAIT();
  OUTPUT_WAVE(emittingPointerL, 1); WAIT();
  OUTPUT_WAVE(emittingPointerL, 2); WAIT();
  OUTPUT_WAVE(emittingPointerL, 3); WAIT();
  OUTPUT_WAVE(emittingPointerL, 4); WAIT();


  goto LOOP;
}
}

void loop() {}

ISR(INT5_vect) {
  sync_cnt++;
}
```

A.1.2 Carte *esclave*

De même, voici le code Arduino utilisé pour le contrôle de la carte *esclave* :

```
#include <avr/sleep.h>
#include <avr/power.h>

#define N_PATTERNS 32
#define N_TRANSDUCTEUR 80
#define N_PORTS 10
#define N_DIVS 10
#define N_T 8

#define WAIT() __asm__ __volatile__("nop\n\t""nop\n\t""nop\n\t""nop\n\t""nop\n\t""nop\n\t""nop\n\t""nop\n\t""nop\n\t""nop\n\t")
#define OUTPUT_WAVE(pointer, d)  PORTA = pointer[d*N_PORTS + 0]; PORTC = pointer[d*N_PORTS + 1]; PORTL = pointer[d*N_PORTS + 2]; PORTB = pointer[d*N_PORTS + 3]; PORTK = pointer[d*N_PORTS + 4]; PORTF = pointer[d*N_PORTS + 5]; PORTH = pointer[d*N_PORTS + 6];  PORTD = pointer[d*N_PORTS + 7]; PORTG = pointer[d*N_PORTS + 8]; PORTJ = pointer[d*N_PORTS + 9]

static byte buffer[ N_DIVS * N_PORTS];
volatile uint8_t sync_cnt = 0;

static inline void wait_ticks(uint8_t n, volatile uint8_t* last_seen) {
  uint8_t start = *last_seen;
  while ( (uint8_t)(sync_cnt - start) < n ) { }
  *last_seen = sync_cnt;
}

void setup()
{
  DDRA = DDRC = DDRL = DDRB = DDRK = DDRF = DDRH = DDRD = DDRG = DDRJ = 0xFF;
  PORTA = PORTC = PORTL = PORTB = PORTK = PORTF = PORTH = PORTD = PORTG = PORTJ = 0x00;

  for (int i = 0; i < (N_PATTERNS * N_DIVS * N_PORTS); ++i) { buffer[i] = 0; }

  EICRB |= (1 << ISC51); EICRB &= ~(1 << ISC50);
  EIFR   = (1 << INTF5); EIMSK |= (1 << INT5);
  pinMode(3, INPUT_PULLUP);

  ADCSRA = 0; power_adc_disable (); power_spi_disable(); power_twi_disable();
  power_timer0_disable(); power_usart1_disable(); power_usart2_disable();
    power_usart3_disable();

  byte buffer[N_DIVS * N_PORTS] = {
  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x7B, 0x8F, 0x07, 0x03,
  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x7B, 0x8F, 0x07, 0x03,
  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x7B, 0x8F, 0x07, 0x03,
  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x7B, 0x8F, 0x07, 0x03,
  0xBD, 0xCD, 0x34, 0xB2, 0xA5, 0x42, 0x6B, 0x0D, 0x07, 0x02,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x42, 0x32, 0xCB, 0x4D, 0x5A, 0xBD, 0x10, 0x82, 0x00, 0x01
};

  byte* emittingPointerH = & buffer[0];
  byte* emittingPointerL = & buffer[N_PORTS * N_DIVS / 2];
  pinMode(13,OUTPUT);
  Serial.begin(115200);
  while(!Serial.available());
  while(Serial.read()!=0x55);
  digitalWrite(13,HIGH);

LOOP:
{
  uint8_t last = sync_cnt;
  wait_ticks(1,&last);
  OUTPUT_WAVE(emittingPointerH, 0); WAIT();
  OUTPUT_WAVE(emittingPointerH, 1); WAIT();
  OUTPUT_WAVE(emittingPointerH, 2); WAIT();
```

```
  OUTPUT_WAVE(emittingPointerH, 3); WAIT();
  OUTPUT_WAVE(emittingPointerH, 4); WAIT();
  OUTPUT_WAVE(emittingPointerL, 0); WAIT();
  OUTPUT_WAVE(emittingPointerL, 1); WAIT();
  OUTPUT_WAVE(emittingPointerL, 2); WAIT();
  OUTPUT_WAVE(emittingPointerL, 3); WAIT();
  OUTPUT_WAVE(emittingPointerL, 4); WAIT();
  goto LOOP;
}
}
void loop() {}
ISR(INT5_vect) { sync_cnt++; }
```