Sound Interactive Business Card

Ian Costabile

Abstract: This paper documents the design and production of a business card design that contains sound and interactive sensors. The purpose of this card was to investigate creative ways of sharing contact details within networks of musicians, technologists and business representatives. The information contained in the card displays contact information and services offered by an artist (the author). The concept was to produce an affordable design that could easily be produced by a single person, initially for a maximum of 20 pieces. Several prototypes were developed, documented and tested during the first semester of 2018.

In business culture, trade or business cards are important networking materials which have existed for centuries. In the mid-19thcentury, their production increased due to chromolithography, a process of making multi-coloured prints. At this time, cards offered less limitations than magazines for displaying illustrations in colour, and for this reason they became 'popular and heavily collected' (Thomas and Banerjee, 2013, p.60). In the 1990s, Mini CDs presented a more interactive way of displaying contact information, along with multimedia presentations, samples of work or digital portfolios (Piotrowski, 2013, ch. 19). However, with the development of mobile technology and new types of media storage, business card CDs did not offer much advantage and thus are not very common today. Furthermore, in recent years, simple contact information on cards has moved on from basic address and telephone information to digital communication details such as email and website addresses, Facebook, LinkedIn and Twitter links.

With the advent of microcontrollers, it is possible for artists to programme and design electronic cards independently. In this project, I have proposed a sound interactive card, which has been linked to my PhD field of research, concerned with the design of interactive sound spaces. This documentation is divided in 6 sections: Components and Sounds; Composition and Interactivity; Production and Electronic Design; Graphic Design; Code; Results and Further Prototypes.

1. Components and Sounds

Primary considerations in selecting components were affordability and dimension, since business cards are usually produced in large quantities and are compact objects. PCB manufacturing can be costly and it was decided that all the components would be soldered by hand, thus it was necessary to use a minimal number of pieces for quick production.

Sound processing was another fundamental consideration. Sound greeting cards, such as Christmas cards, usually employ chips such as the UM66 (UMC, n.d.), which is a melody generator IC. Several versions of this chip can be found with different programmed melodies. The UM66T01, for instance, generates the tones for Jingle Bells and other two Christmas songs. Although these chips offer a quick possibility for integrating sound to small objects, as a composer I felt it would be more interesting to include original compositions made especially for the card itself, and it would be interesting to add interactivity, which would require modulating sounds. For this reason, an ATtiny85 was selected. This is an 8-bit AVR controlled unit developed by Microchip (Microchip, 2013). This chip is affordable[1] and it can generate sound through PWM techniques (Cook, 2015, p. 303). Programming it can be easily done through the Arduino IDE and a ISP programmer module. Another advantage is its capability of running on low power, thus a 3V coin cell battery could be used as a power source. Therefore, the composition could be written in C++, emitting sound directly from a piezo disc, which touching on the card's paper surface is slightly amplified. The PWM technique offered the production of square wave tones and in addition, white noise sounds were created through a function known as LFSR (Linear-Feedback Shift Register), which generates a fast sequence of random signals. Shaping the envelope of the white noise generator made it possible to create drum timbres, such as kick drum and hi-hat.

Other considerations were to use an FSR sensor for modulation sounds, an SMD pushbutton for changing modes and a slide switch for switching the circuit on and off.

2. Composition and Interactivity

It was fundamental to have simple melodies in order to offer more space for interactivity, yet, I still intended to exhibit samples of my personal ideas as a composer. For this, a minimalist structure was employed and three melodic

[1] It was possible to find a dealer selling 20pcs for £15.00, equivalent to £1.33 per piece.

themes were composed for the card. The first theme is made of a tetrachord arpeggio of C major with a flattened 6th in conjunction with a mixed-meter rhythm, 3+3+2+2. The second theme presents a descending minor third parallel motion of a major triad and regular rhythm. The third theme is another variation of a C major *b*6.

Interaction became possible as the user can modulate up to 3 octaves on the first and second themes, and modulate the tempo on the last theme. In the first set of prototypes, modulation was made possible by FSR (Force Sensitive Resistors) sensors. Thus, the user only needed to press a finger on a specific area where an FSR was positioned.

The composition was called Card Fantasia and was attributed to one mode of the card. Another two modes were employed, a synthesiser mode and an 8-bit techno mode. Modes could be changed by pressing an area of the card that contained a momentary pushbutton switch.

The synthesiser mode was a simple way of offering an understanding of the modulation range provided by the FSR sensor. It only emitted a square wave tone through direct mapping of the sensor to frequency (440 to 1760), offering a frequency shift in 3 octaves.

The 8-bit techno mode was an experiment with white noise generating techniques. It is also a composition made of two themes. The user can control 3 variations in these sections, through a build-up arrangement. At its lowest level, only kick drums can he heard. At the second level, hi-hat joins the mix. At the third and last level, a melody is introduced. The user can freely change levels at any moment.

3. Production and Electronic Design

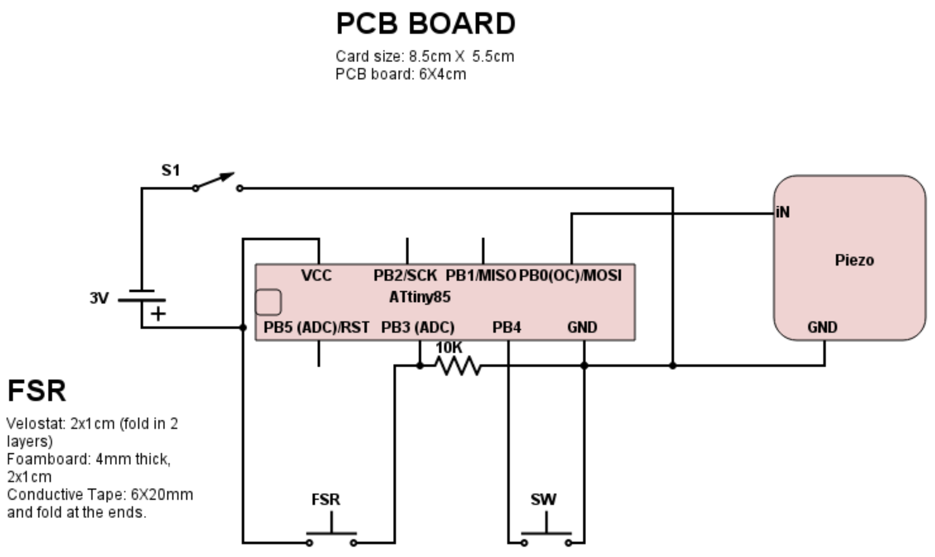

Figure 1: Schematics for Interactive Sound Business Card: Prototype #1 (FSR)

It was important to make the design as simple as possible for quick soldering, thus designing short and direct connections. A slide switch (S1) was used to start the circuit. An ATtiny85 was responsible for receiving sensor data, modulating and generating sounds. A 10K resistor supported the FSR, creating a voltage divider and thus modulating voltage directed to the ADC (PB3) of the microcontroller. A pushbutton switch for changing modes was connected directly through PB4 and GND. The PB4 thus functioned as a digital input and was also connected to an internal pullup resistor (software activated by the microcontroller), eliminating the need for additional external resistors. A piezo disc was connected to a PWM pin of the microcontroller for generating sound. The power source selected was a common 3.3V coin cell battery, the CR2032. The microcontroller was programmed to run at 1MHz, so it could easily run at this voltage level and consume less power. As a business card is not expected to be excessively used, it could be estimated that the battery can last for many years. Once run out, the card would have to be destroyed and battery properly disposed in a recycling point (as a written warning suggests at the back of the card). Initially, the first prototypes were soldered onto PCB boards. After some trials and sensor calibration, further prototypes were soldered directly on the paper card, using copper tapes for connections and a single wire for the piezo disc. A 3D printed part was designed to block the sides of the card and hide the components. This was also used to support the piezo disc and the pushbutton at the correct height, along with securing the coin cell battery in place. The maximum height of components was 3.2mm, which corresponds to the higher component, the CR2032 battery. With the two paper covers surrounding the electronic structure (400gsm and about 0.4mm each), the total height of the product resulted in approximately 4mm.

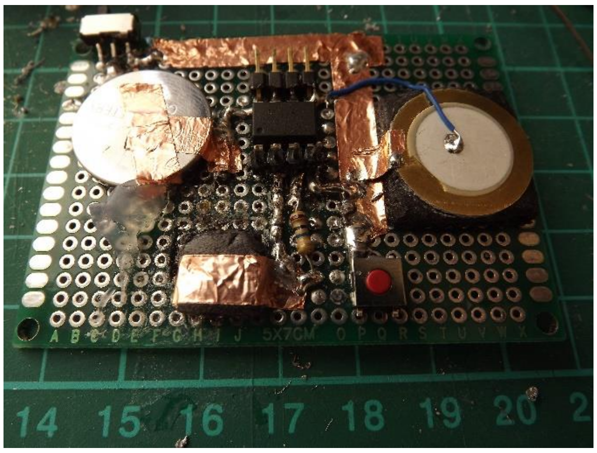
Figure 2: Prototype with components on PCB board.

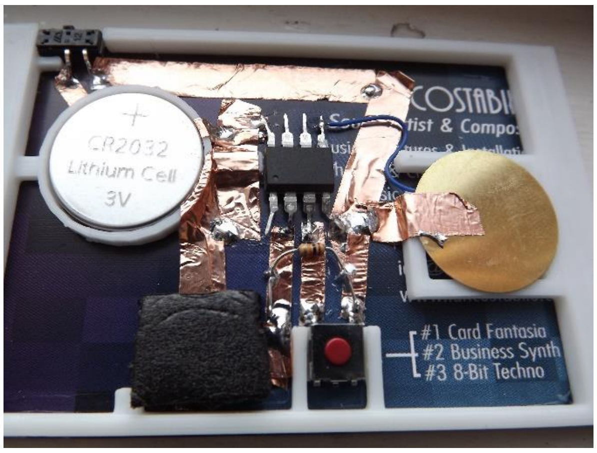

Figure 3: Prototype with components on paper and 3D printed cover.

The FSR sensor had to be hand-made, since commercially available options were not affordable for this project. For this, a Velostat sheet was purchased and cut into slices of 2 x 1cm. It was then glued to a piece of foam board (3mm thick), with a copper tape (20mm thick) in between. The Velostat material is conductive, so the more the user presses against it, the more it will conduct electricity resulting in changes in resistance.

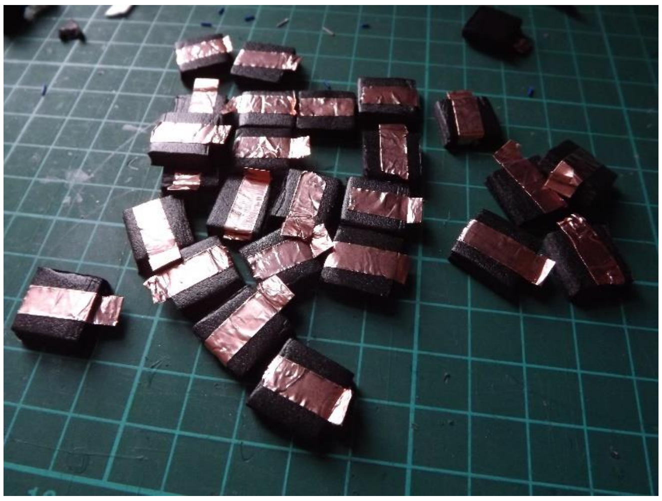
Figure 4: FSR sensors, first stage (copper tape application)

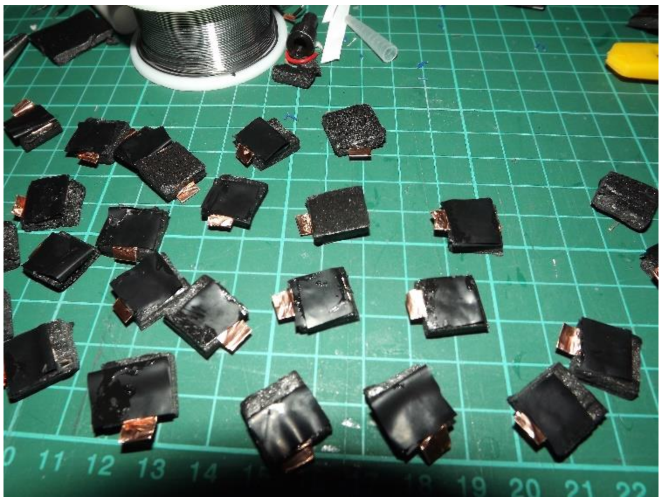
Figure 5: FSR sensors, second stage (Velostat application)

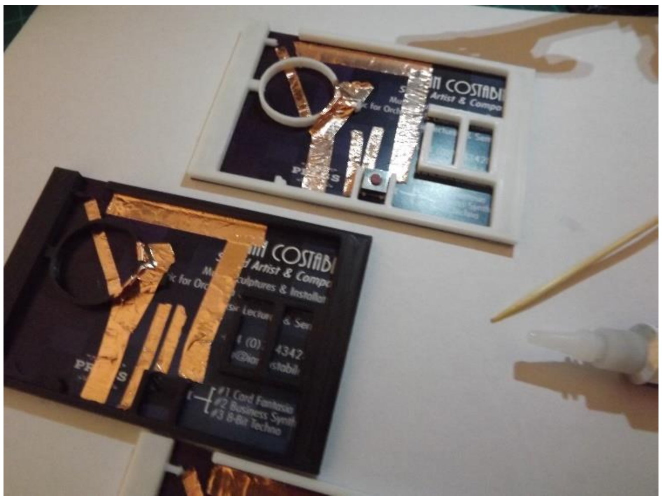
Figure 6: Initial production steps

The final stage of production consisted of gluing the paper card onto the 3D printed part. Positioning it with glue proved to be troublesome, thus double-sided tape was used before the glue, facilitating the process. The total time to produce one card is of about 20 minutes (plus 20 minutes printing the 3D part).

4. Graphic Design

The graphic design was produced by myself in a common graphic editor and printed by an online printing company in 85x55mm (UK standard).

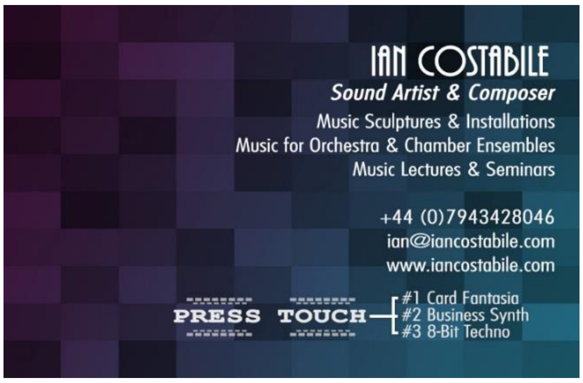

Figure 7: Front of the card

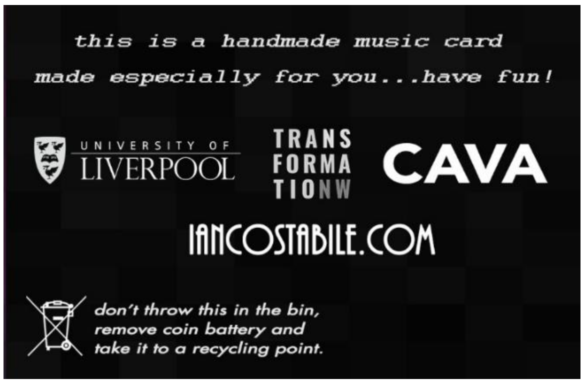
this is a handmade music card
made especially for you...have fun!
UNIVERSITY OF LIVERPOOL
TRANS FORMA TIONW
CAVA
IANCOSTABILE.COM
don't throw this in the bin, remove coin battery and take it to a recycling point.

Figure 8: Back of the card

5. Code

```
#include <Wire.h>
//#include <avr/interrupt.h>
//noise
unsigned long int reg;
unsigned long int newr;
unsigned char lobit;
unsigned char b31, b29, b25, b24;


//FSR sensor
unsigned int  frequency;
int sensorValue;
int sensorMin = 1023; //calibrate at the setup (it needs a high value here,
otherwise calibration will fail)
int sensorMax; //calibrate at the setup

//Tone and Melodies
#include "pitches.h"

unsigned int melodyAbase[] = { NOTE_C4, NOTE_E4, NOTE_G4,
NOTE_C4, NOTE_E4, NOTE_GS4,  //C(b6) arpeggio - 123 123
                     NOTE_E4, NOTE_GS4, NOTE_E4, NOTE_C5 };
//12 12

unsigned int  melodyBbase[] = { NOTE_E4, NOTE_C4, NOTE_G3,
NOTE_E3,
                     NOTE_CS4, NOTE_A3, NOTE_E3, NOTE_CS3,
                      NOTE_AS3, NOTE_FS3, NOTE_CS3,
                      NOTE_AS3, NOTE_G3, NOTE_DS3, NOTE_AS2,
NOTE_C2 }; //

unsigned int  melodyCbase[] = { NOTE_E5, NOTE_C5, NOTE_GS4,
NOTE_G4, NOTE_E4, NOTE_G4, NOTE_GS4, NOTE_C5,
                      NOTE_D5, NOTE_C5, NOTE_GS4, NOTE_G4,
NOTE_E4, NOTE_G4, NOTE_GS4, NOTE_C5};

/*unsigned int  pentatonic[] = { NOTE_A3, NOTE_C4, NOTE_D4,
NOTE_E4, NOTE_G4,
                     NOTE_A4, NOTE_C5, NOTE_D5, NOTE_E5,
NOTE_G5,
                     NOTE_A5, NOTE_C6, NOTE_D6, NOTE_E6,
NOTE_G6,
                     NOTE_A6, NOTE_C7, NOTE_D7, NOTE_E7,
NOTE_G7};
*/

unsigned int  bpm = 130;
uint8_t  thisNote;
uint8_t  speakerPin = 0;
uint8_t  octave = 2; //0 to 2, higher octaves won't play properly
uint8_t  technoPress = 0;

unsigned int bpmTOms;
uint8_t crotchet;
uint8_t quaver;
uint8_t semiquaver;
uint8_t demisemiquaver;
uint8_t hemidemisemiquaver;
uint8_t semihemidemisemiquaver;
uint8_t subdivision;


//Counters
volatile int pressCounter = 0;


void setup() {
  cli(); //close interrupts
  GIMSK |= _BV(PCIE);                   // Enable Pin Change Interrupts
  PCMSK |= _BV(PCINT4);                 // Use PB4 as interrupt pin
  sei(); //open interrupts, last thing in the setup
  pinMode(speakerPin, OUTPUT);
  pinMode(A3, INPUT);
  pinMode(4, INPUT_PULLUP); //PUSHBUTTON - INTERRUPT
  reg = 0x55aa55aaL; //For the Noise Generator: The seed for the
bitstream. It can be anything except 0.
 //calibrate minimum value for 500ms
 while (millis() < 500) {
  sensorValue = analogRead(A3);
    if (sensorValue < sensorMin) {
      sensorMin = sensorValue;
      sensorMax = sensorMin + 250; //CALIBRATE HERE
    }
  }
```

```
}

void loop() {
  if (pressCounter == 0){
    //composition mode
    bpm = 210;
    arpeggioA();
    arpeggioB();
    arpeggioC();
    }
  //generateHighNoise (1000);
  else if (pressCounter == 1){
    //synth mode
  FSRsynth();
  }
    else if (pressCounter == 2){
      //techno mode
       bpm = 130;
       bpmTOms = 60000 / bpm;
       crotchet = bpmTOms;
       quaver = bpmTOms / 2;
       semiquaver = bpmTOms / 4;
       demisemiquaver = bpmTOms / 8;
       hemidemisemiquaver = bpmTOms / 16;
       semihemidemisemiquaver = bpmTOms / 32;
       subdivision = semiquaver;
        technoA();
        technoB();
    }
}

 ISR(PCINT0_vect) {
 static unsigned long last_interrupt_time = 0;
 unsigned long interrupt_time = millis();
 // If interrupts come faster than 200ms, assume it's a bounce and ignore

 if (interrupt_time - last_interrupt_time > 200)
 {
    if (digitalRead(4) == LOW){ //still needs this as the interrupt is
triggered by both states
  pressCounter++; //this has to be a volatile variable to work inside the
ISR
    if (pressCounter >= 3){
    pressCounter = 0; //reset counter
    }
  }
 }

 last_interrupt_time = interrupt_time;
}
void sensorReading(){
  sensorValue = analogRead(A3);
}

void FSRsynth(){
  sensorReading();
  frequency = map(sensorValue, sensorMin, sensorMax, 440, 1760);
  frequency = constrain(frequency, 440, 1760);
  tone (speakerPin, frequency, 100);
}

/*
void FSRsynth2(){
  sensorReading();
  thisNote = map(sensorValue, sensorMin, sensorMax, 0, 19);
  if (thisNote > 19){
  tone (speakerPin, pentatonic[19], 10);
  }
  else if (thisNote < 0) {
  tone (speakerPin, pentatonic[0], 10);
  }
  else if (thisNote >= 0 && thisNote <= 19) {
  tone (speakerPin, pentatonic[thisNote], 10);
  }
  delay (10);
}
*/


void FSRoctave() {
  sensorReading();
  octave = map(sensorValue, sensorMin, sensorMax, 0, 2);
  octave = constrain(octave, 0, 2);
}

void FSRbpm() {
  sensorReading();
  bpm = map(sensorValue, sensorMin, sensorMax, 300, 500);
  bpm = constrain(bpm, 300, 500);
}

void FSRtechno() {
  sensorReading();
  technoPress = map(sensorValue, sensorMin, sensorMax, 0, 2);
  technoPress = constrain(technoPress, 0, 2);
}

void arpeggioA (){
for (unsigned int i = 0; i <= 7; i++){
  for (thisNote = 0; thisNote < sizeof(melodyAbase)/sizeof(int);
thisNote++) {
    if (pressCounter == 0){
    FSRoctave();
    unsigned int  bpmTOms = 60000 / bpm * 4;
    unsigned int  noteDuration = bpmTOms / 8;
    unsigned int  melodyA = melodyAbase[thisNote]*pow(2, octave);
//calculate octave from the sensor
    tone(speakerPin, melodyA, noteDuration);
    unsigned int  pauseBetweenNotes = noteDuration * 1.30;
   // buttonPress(); //so it can exit

    delay(pauseBetweenNotes);
      }
    }
  }
}
void arpeggioB (){
for (unsigned int i = 0; i <= 7; i++){
  for (thisNote = 0; thisNote < sizeof(melodyBbase)/sizeof(int);
thisNote++) {
    if (pressCounter == 0){
    FSRoctave();
    unsigned int  bpmTOms = 60000 / bpm * 4;
    unsigned int  noteDuration = bpmTOms / 8;
    unsigned int  melodyB = melodyBbase[thisNote]*pow(2, octave);
//calculate octave from the sensor
    tone(speakerPin, melodyB, noteDuration);
    unsigned int  pauseBetweenNotes = noteDuration * 1.30;
 //   buttonPress(); //so it can exit
    delay(pauseBetweenNotes);
      }
    }
  }
}
void arpeggioC (){
for (unsigned int i = 0; i <= 7; i++){
  for (thisNote = 0; thisNote < sizeof(melodyCbase)/sizeof(int);
thisNote++) {
    if (pressCounter == 0){
    FSRbpm();
    unsigned int  bpmTOms = 60000 / bpm * 4;
    unsigned int  noteDuration = bpmTOms / 8;
    unsigned int  melodyC = melodyCbase[thisNote]*pow(2, octave);
//calculate octave from the sensor
    tone(speakerPin, melodyC, noteDuration);
    unsigned int pauseBetweenNotes = noteDuration * 1.30;
   // buttonPress(); //so it can exit
    delay(pauseBetweenNotes);
      }
    }
  }
}
void technoA (){  // ANOTHER OCTAVE HIGHER AND LESS
DRUMS
for (unsigned int i = 0; i <= 31; i++){
   // BEAT 1
  FSRtechno();
    if (pressCounter == 2){
    generateLowNoise (semihemidemisemiquaver); //1.1
    delay (subdivision - semihemidemisemiquaver);
    //no_sound//1.2
    delay (subdivision);
    if (technoPress == 2){
    generateHighNoise (semiquaver); //1.3
    delay (subdivision - semiquaver);
    }
        else {
        delay (subdivision); //1.3
        }
    if (technoPress == 0){
    }
        else {
```

```
      tone (speakerPin, 524, demisemiquaver); //1.4
      }
    delay (subdivision);
     // BEAT 2
      FSRtechno();
    if (technoPress == 0){
    }
    else if (technoPress == 1 || technoPress == 2){
    tone (speakerPin, 1048, demisemiquaver); //1.1
    }
    generateLowNoise (semihemidemisemiquaver); //1.1
    delay (subdivision - semihemidemisemiquaver);
    //no_sound//1.2
    delay (subdivision);
    if (technoPress == 0){
    //no_sound////1.3
    delay (subdivision);
    }
    else if (technoPress == 1){
    tone (speakerPin, 524, semiquaver); //1.3
    delay (subdivision);
    }
    else if (technoPress == 2){
    tone (speakerPin, 524, semiquaver); //1.3
    generateHighNoise (semiquaver); //1.3
    delay (subdivision - semiquaver);
    }
    //no_sound//1.4
    delay (subdivision);
    }
  }
}
void technoB (){  // THEME 2
for (unsigned int i = 0; i <= 31; i++){
    if (pressCounter == 2){
    //BEAT 1
    FSRtechno();
    generateLowNoise (semihemidemisemiquaver); //1.1
    delay (subdivision - semihemidemisemiquaver);
    if (technoPress == 1 || technoPress == 2){
    tone (speakerPin, 264, demisemiquaver);//1.2
    }
    delay (subdivision);
    if (technoPress == 0 || technoPress == 1){
    delay (subdivision);
    }
    else if (technoPress == 2){
    generateHighNoise (semiquaver); //1.3
    delay (subdivision - semiquaver);
    }
    if (technoPress == 1 || technoPress == 2){
    tone (speakerPin, 311, demisemiquaver); //1.4
    }
    delay (subdivision);
   //BEAT 2
    FSRtechno();
    if (technoPress == 1 || technoPress == 2){
    tone (speakerPin, 370, demisemiquaver); //1.1
    }
    generateLowNoise (semihemidemisemiquaver); //1.1
    delay (subdivision - semihemidemisemiquaver);
    //no_sound//1.2
    delay (subdivision);
    if (technoPress == 0){
    delay (subdivision);
    }
    else if (technoPress == 1){
    tone (speakerPin, 264, 30); //1.3
    delay (subdivision);
    }
    else if (technoPress == 2){
    generateHighNoise (semiquaver); //1.3
    delay (subdivision - semiquaver);
    }
    if (technoPress == 1 || technoPress == 2){
    tone (speakerPin, 264, demisemiquaver); //1.4
    }
    delay (subdivision);
    }
  }
}

void generateHighNoise(uint32_t period){
  for( uint32_t tStart = millis();  (millis()-tStart) < period;  ){ //the loop
runs over the period duration
   b31 = (reg & (1L << 31)) >> 31;
   b29 = (reg & (1L << 29)) >> 29;
   b25 = (reg & (1L << 25)) >> 25;
   b24 = (reg & (1L << 24)) >> 24;
   lobit = b31 ^ b29 ^ b25 ^ b24;
   newr = (reg << 1) | lobit;
   reg = newr;
   digitalWrite (speakerPin, reg & 1);
   delayMicroseconds (30);} // Changing this value changes the frequency.
}
void generateLowNoise(uint32_t period){
  for( uint32_t tStart = millis();  (millis()-tStart) < period;  ){ //the loop
runs over the period duration
   b31 = (reg & (1L << 31)) >> 31;
   b29 = (reg & (1L << 29)) >> 29;
   b25 = (reg & (1L << 25)) >> 25;
   b24 = (reg & (1L << 24)) >> 24;
   lobit = b31 ^ b29 ^ b25 ^ b24;
   newr = (reg << 1) | lobit;
   reg = newr;
   digitalWrite (speakerPin, reg & 1);
   delayMicroseconds (200);} // Changing this value changes the
frequency.
}
```

6. Results and Further Prototypes

Some samples were given in meetings and events for academics and corporate representatives, especially for purposes of collaboration in projects through the Transformation North West doctoral programme[2]. Clients received them with enthusiasm and curiosity and I believe the product has supported trust in my skills in design, music and electronics, increasing my prospects of collaboration with industry partners.

Although the custom-made FSR sensors worked for some samples, they failed in others, resulting in different sensor readings. The interface also did not seem to be the most straightforward as some users found complicated to operate it. A third

[2] https://transformationnorthwest.org/

observation was the thickness of the card. Although 4mm is thin enough to fit in a pocket, its thickness distances from the design characteristics of a traditional business card. For these reasons, I decided to develop a second set of prototypes. To solve the sensor issues, the FSR sensor was replaced by an accelerometer, thus sound modulation could be controlled by tilting the card to different angles. In order to reduce its thickness, it was possible to use smaller components, a CR1216 battery (1.25mm height), an SMD slide switch (6x2x4mm — 2mm height) and a SOIC Attiny85 (ATTINY85-20SU — 2.16mm height). The resulting thickness was of 2.3mm, nearly half of the thickness of the first prototype.

References

Cook, M. (2015) Arduino Music and Audio Projects. Apress.

Microchip (2013) Atmel 8-bit AVR Microcontroller with 2/4/8K Bytes In-System Programmable Flash: ATtiny25/V / ATtiny45/V / ATtiny85/V. Available at: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2586-AVR-8-bit-Microcontroller-ATtiny25-ATtiny45-ATtiny85_Datasheet.pdf (Accessed: 10 May 2018)

Piotrowski, C. M. (2008) Professional Practice for Interior Designers. John Wiley & Sons.

Thomas, A. M. K. and Banerjee, A. K. (2013) The History of Radiology. Oxford University Press.

UMC (n.d.) UM66T Series. Simple Melody Generator. Available at: http://www.edutek.ltd.uk/Binaries/Datasheets/Linear/UM66.PDF (Accessed: 10 May 2018)