ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

Харківський національний університет радіоелектроніки
Кафедра ЕОМ

Кваліфікаційна робота
Перший (бакалаврський) рівень

Мікроконтролерний пристрій для синтезу музики

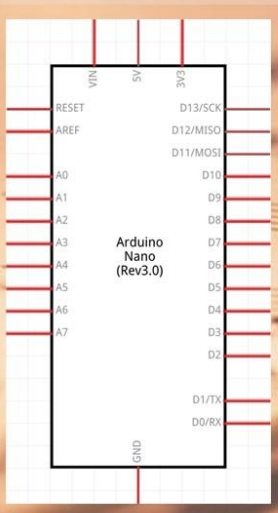
Автор:
Данило Хижняк,
ст.гр. КІУКІ-21-3

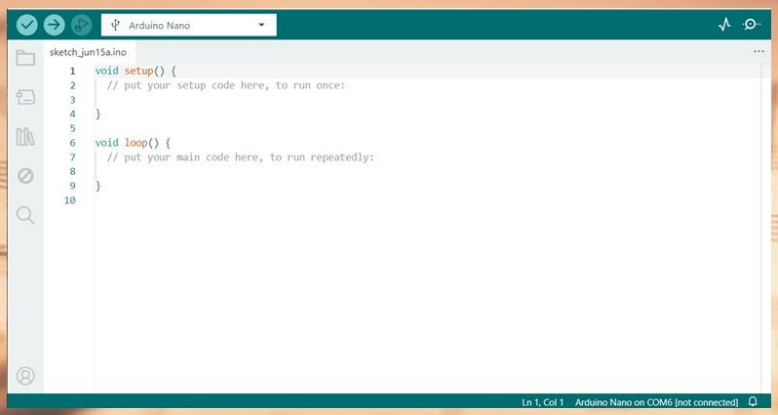
Керівник:
Дмитро Росінський,
ст. викл. каф. ЕОМ

Ключові недоліки наявних рішень

- Відсутність підтримки поліфонії
- Обмежена автономність
- Відсутність зручної індикації
- Тривіальність

Апаратна база та програмні інструменти розробки





3

Компоненти розробки

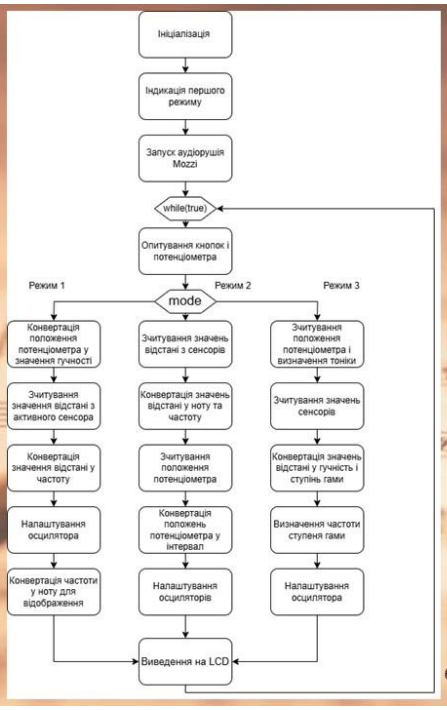
4

Етапи виконання роботи

- Аналіз наявних на ринку рішень з метою сформулювати технічне завдання
- Висвітлення теоретичних відомостей про оцифрування та синтез звуку для вибору програмних засобів (бібліотек)
- Формулювання технічного завдання
- Проєктування схеми пристрою
- Розробка алгоритму керуючої програми та його реалізація

5

Алгоритм керуючої програми МК



6

Загальна архітектура програми

7

Процедура перетворення звуку у частоту

```
float getFreq(int distance) { //Конвертація частоти у відстань
  unsigned long now = mozziMicros();
  if (now - lastUpdate < UPD_INT) {
    return lastFreq;
  }
  lastUpdate = now;
  if (distance <= 0 || distance > MAX_DIST) {
    return lastFreq;
  }
  float clampedDist = constrain((float)distance, MIN_DIST,
MAX_DIST);
  float newFreq = MIN_FREQ + ((clampedDist - MIN_DIST) /
(MAX_DIST - MIN_DIST)) * (MAX_FREQ - MIN_FREQ);
  if (abs(newFreq - lastFreq) < 5.0) {
    return lastFreq;
  }
  float smoothFreq = lastFreq * 0.9 + newFreq * 0.1;
  lastFreq = smoothFreq;
  return smoothFreq;
}
```

8

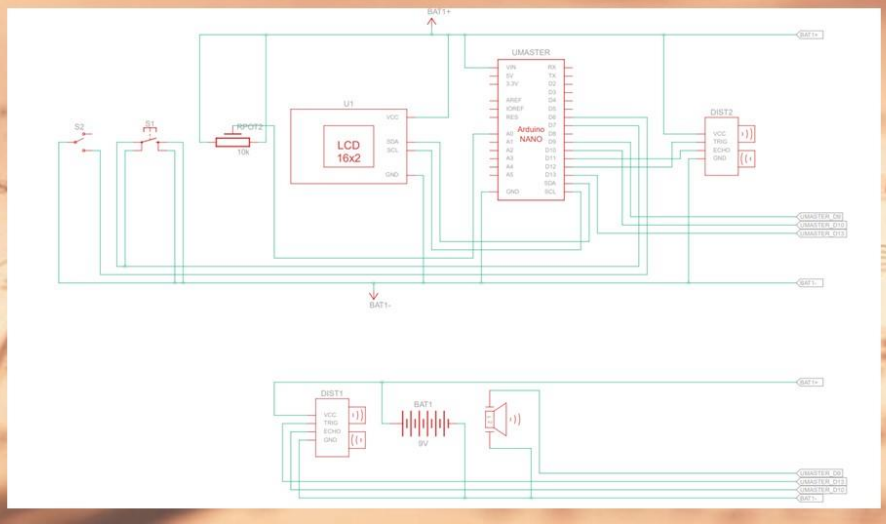
Реалізація функцій конвертації і перемикання режиму

```
byte getScaleDegreeFromDistance(int dist) {
  const byte numDegrees = 7;
  dist = constrain(dist, MIN_DIST, MAX_DIST); //Обмежуємо значення
відстані
  float norm = (float)(dist - MIN_DIST) / MAX_DIST; //Нормалізуємо
значення відстані
  byte degree = (byte)(norm * numDegrees); //Отримуємо інтервал
  if (degree >= numDegrees) degree = numDegrees - 1;
  return degree;
```

```
short switchMode() {
  if (mozziMicros() - lastChangeTime > UPD_INT) {
    byte buttonState = digitalRead(MODE_BUTTON_PIN);
    if (buttonState != lastButtonState) {
      lastChangeTime = mozziMicros();
      lastButtonState = buttonState;
      if (buttonState == LOW) {
        return (mode + 1) % mode_count; //Змінюємо режим лише коли
зміна логічного рівня триває довше ніж брязкіт контактів
      }
    }
  }
  return mode;
}
```

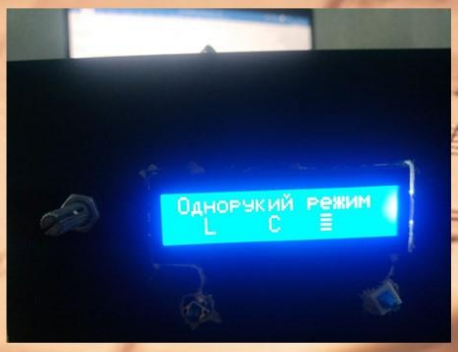
9

Принципова схема пристрою



10

Демонстрація пристрою



11

Висновки до роботи

- Створено оригінальний пристрій для синтезу звуку на базі мікроконтролера ATMega328P.
- Властивості: доступність, простота, зрозумілий інтерфейс, музична виразність.
- Усунено недоліки наявних рішень: забезпечено автономність і підтримку базової поліфонії.
- Потенційні шляхи вдосконалення: додавання другого МК або заміна на потужніший.

12

ДОДАТОК Б
КЕРУЮЧИЙ КОД ПРОГРАМИ

Б.1 Заголовочний файл програми

Лістинг Б.1.1 – Файл «therduino.h»

```
#ifndef THERDUINO_H
#define THERDUINO_H
#include <Wire.h> //Бібліотека I2C
#include <MozziPing.h>
#include <Mozzi.h>
#include <Oscil.h>
#include <tables/sin2048_int8.h> //Імпорт двигуна синтезу звуку
#include <tables/saw2048_int8.h>
#include <EventDelay.h>
#define MODE_BUTTON_PIN 7 //Основні константи - піни, екстремуми
відстані, частота, інтервал для імітації delay проти брязкоту
кнопок
#define MAJSWITCH 6
#define TONIC_POT_PIN A0
#define TRIG_LEFT 12
#define ECHO_LEFT 11
#define TRIG_RIGHT 13
#define ECHO_RIGHT 10
#define MIN_DIST 5
#define MAX_DIST 60
#define MIN_FREQ 261.63
#define MAX_FREQ 987.77
#define UPD_INT 50000
#define CHANGE_MODE      0x01 // Види змін параметрів звуку для
вибіркового оновлення на екрані, які повертає функція
#define CHANGE_TONIC     0x02
#define CHANGE_MAJOR     0x04
#define CHANGE_VOLUME    0x08
#define CHANGE_INTERVAL  0x10
#define CHANGE_NOTE      0x20
float getFreq(int distance); //Прототипи функцій
void getLevel(byte volume);
byte distanceToLevel(int distance);
int freqToNote(float freq);
byte getScaleDegreeFromDistance(int dist);
float getScaleFreq(byte tonic, bool isMajor, byte degree);
short switchMode();
void handleMajSwitch();
byte tonicSelect();
void drawMode();
void drawMode0(uint8_t changes), drawMode1(uint8_t changes),
```

```
drawMode2(uint8_t changes);
#endif
```

Б.2 Основний код програми

Лістинг Б.2.1 – Файл “master.ino”

```
#include "therduino.h"
#define _LCD_TYPE 1 //Необхідне для бібліотеки для роботи з
екраном визначення типу підключення через I2C
#include <LCD_1602_RUS_ALL.h> //Бібліотека для роботи з екраном,
код не компілюється, якщо винести підключення у заголовочний
файл
#include <Wire.h>
// Масив режимів та змінна поточного режиму
const char* modes[3] = {"Однорукий режим", "Дворукий режим",
"Тональний режим"};
const short mode_count = sizeof(modes) / sizeof(modes[0]);
short mode = 0;

// Массив нот
const char* notes[12] = {"A", "A#", "B", "C", "C#", "D", "D#",
"E", "F", "F#", "G", "G#"};
float intervalRatios[] = {
  1.0,    // унісон
  1.189,  // мала терція
  1.26,   // великаерція
  1.33,   // кварта
  1.5,    // квинта
  2.0     // октава
};
//Масиви ступеней мажорного та мінорного звукорядів, а також
значення гучності, що відповідають рівням
const byte majorScale[7] = {0, 2, 4, 5, 7, 9, 11};
const byte minorScale[7] = {0, 2, 3, 5, 7, 8, 10};
const byte volumeLevels[5] = {50, 100, 150, 200, 250};

//Два табличних осцилятори - синусоїдальний та пилоподібний із
частотою дискретизації 2048 Гц
Oscil <SIN2048_NUM_CELLS, MOZZI_AUDIO_RATE>
sin_osc(SIN2048_DATA);
Oscil <SAW2048_NUM_CELLS, MOZZI_AUDIO_RATE>
saw_osc(SAW2048_DATA);

byte volume = 0; //Основні змінні
int volumeLevel;
byte tonicIndex = 9;
int intervalIndex = 0;
byte note = 9;
int dist = 0;

unsigned short lastButtonState = LOW; //Змінні debounce
```

```
unsigned long lastChangeTime = 0;
EventDelay debounceDelay;

bool isMajor = true;
bool inactiveSensorRight = true;
bool intervalChange = false;

float freq = 261.63; //Нота "До" 1 октави за замовчуванням
float lastFreq = 261.63; //Змінна для визначення факту зміни
частоти порівнянням
unsigned long lastUpdate = 0; // Час останнього оновлення

MozziPing sonarLeft(TRIG_LEFT, ECHO_LEFT, 60); //Оголошуємо
об'єкти ультразвукових сенсорів
MozziPing sonarRight(TRIG_RIGHT, ECHO_RIGHT, 60);

LCD_1602_RUS lcd(0x27,16,2); // Оголошуємо екран
void setup() {
  pinMode(MODE_BUTTON_PIN, INPUT_PULLUP);
  pinMode(MAJSWITCH, INPUT_PULLUP);
  lcd.init();
  lcd.backlight(); //Ініціалізація екрану
  Serial.begin(9600); //З'єднання по віртуальному COM для дебагу
  drawMode0(CHANGE_MODE); //Імітуємо переключення на перший
режим кнопкою для коректного стану змінних
  startMozzi(); // Запуск аудіодвигуна
}
void updateControl(){ //Основна керуюча функція
  short newMode = switchMode();
  if (newMode != mode) {
    mode = newMode; //Обробка кнопки зміни режимів із захистом
від нескінченного переключення в разі залипання
  }
  handleMajSwitch(); //Обробка перемикача
  switch(mode) {
  case 0:
    volume = mozziAnalogRead<8>(TONIC_POT_PIN); //Зчитування
"сирого" значення гучності, яке переводиться у рівень функцією
getLevel()
    getLevel(volume);
    if(inactiveSensorRight){
    dist = sonarLeft.ping_cm(); //Лівий сенсор надає значення
дистанції, якщо правий неактивний
    freq = getFreq(dist); //Отримуємо частоту зі значення
дистанції
    sin_osc.setFreq(freq);
    note = freqToNote(freq); //Індекс ноти у масиві для
відображення
  }
    else {
    dist = sonarRight.ping_cm();
    freq = getFreq(dist);
    sin_osc.setFreq(freq);
```

```
    note = freqToNote(freq);
  }
  case 1:
    if(inactiveSensorRight){
    dist = sonarLeft.ping_cm();
    volume = sonarRight.ping_cm(); //У цьому режимі гучністю
керує вже другий сенсор
  }
    else {
    dist = sonarRight.ping_cm();
    volume = sonarLeft.ping_cm();
  }
    distanceToLevel(volume);
    freq = getFreq(dist);
    note = freqToNote(freq);
    intervalIndex = constrain(mozziAnalogRead(TONIC_POT_PIN) /
(1024 / 6), 0, 5); //Зчитуємо положення потенціометра як
інтервал зі списку і обмежуємо значення.
    float harmonyFreq = freq * intervalRatios[intervalIndex];
    sin_osc.setFreq(freq);
    saw_osc.setFreq(harmonyFreq); //Налаштування додаткового
осцилятора
    case 2:
     tonicIndex = tonicSelect();
    dist = sonarLeft.ping_cm();
    volume = sonarRight.ping_cm();
    distanceToLevel(volume);
    byte degree = getScaleDegreeFromDistance(dist); //Отримання
ступеню гами в залежності від відстані
    float scaleFreq = getScaleFreq(tonicIndex, isMajor, degree);
//Отримання частоти ступеню, враховуючи тип звукоряду та тоніку
    sin_osc.setFreq(scaleFreq);
  }
  drawMode();
}
AudioOutput updateAudio(){
  int16_t voice1 = sin_osc.next();
  int16_t voice2 = saw_osc.next();
  int16_t mixed = (voice1 + voice2) / 2;
  if(mode == 0){
  return MonoOutput::from16Bit(voice1 *
volumeLevels[volumeLevel-1]); //Передаємо асинхронній функції
команду на подальше обчислення хвилі
  } //із заданими параметрами звуку
  else if(mode == 1){
  return MonoOutput::from16Bit(mixed * volumeLevels[volumeLevel-
1]);
  }
  else if(mode == 2){
    return MonoOutput::from16Bit(voice1 *
volumeLevels[volumeLevel-1]);
  }
}
```

```
void loop() {
  audioHook();
}
```

Лістинг Б.2.2 – Файл «therduino_conversions.h»

```
#include "therduino.h"
float getFreq(int distance) { //Конвертація частоти у відстань
  unsigned long now = mozziMicros();
  if (now - lastUpdate < UPD_INT) {
    return lastFreq;
  }
  lastUpdate = now;
  if (distance <= 0 || distance > MAX_DIST) {
    return lastFreq;
  }
  float clampedDist = constrain((float)distance, MIN_DIST,
MAX_DIST);
  float newFreq = MIN_FREQ + ((clampedDist - MIN_DIST) /
(MAX_DIST - MIN_DIST)) * (MAX_FREQ - MIN_FREQ);
  if (abs(newFreq - lastFreq) < 5.0) {
    return lastFreq;
  }
  float smoothFreq = lastFreq * 0.9 + newFreq * 0.1;
  lastFreq = smoothFreq;
  return smoothFreq;
}
void getLevel(byte volume){ //Встановлення рівня гучності в
залежності від показів потенціометра
  if(volume <= 49) volumeLevel = 5;
  if(volume > 49 && volume <= 98) volumeLevel = 4;
  if(volume > 98 && volume <= 147) volumeLevel = 3;
  if(volume > 147 && volume <= 196) volumeLevel = 2;
  if(volume > 196) volumeLevel = 1;
}
byte distanceToLevel(int distance) { //Встановлення рівня
гучності в залежності від відстані, яку показує сенсор
  if(distance <= 10) volumeLevel = 5;
  if(distance > 10 && distance <= 20) volumeLevel = 4;
  if(distance > 20 && distance <= 30) volumeLevel = 3;
  if(distance > 30 && distance <= 40) volumeLevel = 2;
  if(distance > 40 or distance == 0) volumeLevel = 1;
}
int freqToNote(float freq) {
  int n = round(12.0 * log(freq / 440.0)) + 9; //Обчислення
індексу ноти у масиві
  int noteIndex = (n % 12 + 12) % 12;
  return noteIndex;
}
byte getScaleDegreeFromDistance(int dist) { //Конвертація
відстані у ступінь
  const byte numDegrees = 7;
  dist = constrain(dist, 5, 60);
  float norm = (float)(dist - 5) / 60.0;
```

```
  byte degree = (byte)(norm * numDegrees);
  if (degree >= numDegrees) degree = numDegrees - 1;
  return degree;
}
float getScaleFreq(byte tonic, bool isMajor, byte degree) {
//Отримання частоти ступеню гами з урахуванням типу звукоряду
  const byte* scale = isMajor ? majorScale : minorScale;
  byte semitoneOffset = scale[degree];
  byte noteIndex = (tonic + semitoneOffset) % 12;
  int octave = (tonic + semitoneOffset) / 12;
  return 440.0 * pow(2.0, ((noteIndex - 9) + 12 * octave) /
12.0);
}
```

Лістинг Б.2.3 – Файл «therduino_graphics.ino»

```
#include "therduino.h"
  byte lastMode = 255;
  byte lastTonic = 255;
  bool lastMajor = !isMajor;
  byte lastVolume = 0;
  int lastInterval = 0;
  byte lastNote = 9;
  int lastLevel = 1;
uint8_t checkDisplayChanges() { //Повернення конкретного типу
зміни звуку
  uint8_t changes = 0;
  if (mode != lastMode)             changes |= CHANGE_MODE;
  if (tonicIndex != lastTonic)      changes |= CHANGE_TONIC;
  if (isMajor != lastMajor)         changes |= CHANGE_MAJOR;
  if (volumeLevel != lastLevel)     changes |= CHANGE_VOLUME;
  if (intervalIndex != lastInterval)changes |= CHANGE_INTERVAL;
  if (note != lastNote)             changes |= CHANGE_NOTE;
  return changes;
}

void updateLastValues(uint8_t changes) { //Синхронізація
попередніх значень за умови їхньої зміни
  if (changes & CHANGE_MODE) lastMode = mode;
  if (changes & CHANGE_MODE) lastTonic = tonicIndex;
  if (changes & CHANGE_MAJOR) lastMajor = isMajor;
  if (changes & CHANGE_VOLUME) lastLevel = volumeLevel;
  if (changes & CHANGE_INTERVAL) lastInterval = intervalIndex;
  if (changes & CHANGE_NOTE) lastNote = note;
}

// Отрисовка каждого режима
void drawMode0(uint8_t changes) { //Малювання 1 режиму
  if (changes & CHANGE_MODE) {
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print(modes[mode]);
```

```
    lcd.setCursor(2, 1);
    lcd.print(inactiveSensorRight ? "R" : "L");

    lcd.setCursor(7, 1);
    lcd.print(notes[freqToNote(freq)]);

    lcd.setCursor(11, 1);
    for(int i = 0; i< volumeLevel; i++){
      lcd.write(7);
    }
    return;
  }
  lcd.setCursor(2, 1);
  lcd.print(inactiveSensorRight ? "R" : "L");
  if (changes & CHANGE_NOTE) {
    lcd.setCursor(7, 1);
    lcd.print("  "); // очистить 2 символа
    lcd.setCursor(7, 1);
    lcd.print(notes[freqToNote(freq)]);
  }

  if (changes & CHANGE_VOLUME) {
    lcd.setCursor(11, 1);
    lcd.print("     "); // очищуємо рівні гучності на екрані
    lcd.setCursor(11,1);
    for(int i = 0; i<volumeLevel; i++){
      lcd.write(7);
    }
  }
}
void drawMode1(uint8_t changes) { //Малювання 2 режиму
  if (changes & CHANGE_MODE) {
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print(modes[mode]);
    lcd.setCursor(2, 1);
    lcd.print(inactiveSensorRight ? "R" : "L");
    lcd.setCursor(7, 1);
    lcd.print(notes[freqToNote(freq)]);
    lcd.setCursor(11, 1);
    for(int i = 0; i<volumeLevel; i++){
      lcd.write(7);
    }
    return;
  }
  lcd.setCursor(2, 1);
  lcd.print(inactiveSensorRight ? "R" : "L");
  if (changes & CHANGE_NOTE) {
    lcd.setCursor(7, 1);
    lcd.print("  "); // очистить 2 символа
    lcd.setCursor(7, 1);
    lcd.print(notes[freqToNote(freq)]);
  }
```

```
  if (changes & CHANGE_VOLUME) {
    lcd.setCursor(11, 1);
    lcd.print("     "); // очищуємо рівні гучності на екрані
    lcd.setCursor(11,1);
    for(int i = 0; i<volumeLevel; i++){
      lcd.write(7);
    }
  }
}

void drawMode2(uint8_t changes){ //Малювання 3 режиму
  if (changes & CHANGE_MODE) {
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print(modes[mode]);
    lcd.setCursor(2, 1);
    lcd.print(isMajor ? "Maj" : "Min");
    lcd.setCursor(7, 1);
    lcd.print(notes[tonicIndex]);
    lcd.setCursor(11, 1);
    for(int i = 0; i<volumeLevel; i++){
      lcd.write(7);
    }
    return;
  }
  if (changes & CHANGE_TONIC) {
    lcd.setCursor(7, 1);
    lcd.print("  "); // очистить 2 символа
    lcd.setCursor(7, 1);
    lcd.print(notes[tonicIndex]);
  }

  if (changes & CHANGE_MAJOR){
    lcd.setCursor(2, 1);
    lcd.print("   ");
    lcd.setCursor(2, 1);
    lcd.print(isMajor ? "Maj" : "Min");
  }

  if (changes & CHANGE_VOLUME) {
    lcd.setCursor(11, 1);
    lcd.print("     "); // очищуємо рівні гучності на екрані
    lcd.setCursor(11,1);
    for(int i = 0; i<volumeLevel; i++){
      lcd.write(7);
    }
  }
}

void drawMode() {
  uint8_t changes = checkDisplayChanges(); //Перевірка зміни
значень
```

```
  if (changes == 0) return;

  switch (mode) {
    case 0:
      drawMode0(changes);
      break;
    case 1:
      drawMode1(changes);
      break;
    case 2:
      drawMode2(changes);
      break;
  }
  updateLastValues(changes);
}
```