فرشید نوشی ۹۸۲۱۰۲۸

کاربردهای پیزوالکتریک ← گیتار، سنسورها، منابع ولتاژ در توان بالا

نحوه‌ی کارکرد اسپیکر پیزوالکتریک ← کریستال داخلی پیزو با اعمال ولتاژ تغییر نوسان می‌دهد پس می‌توانیم با اعمال ولتاژ متغیر نوسان در کریستال ایجاد کرده و در نتیجه صوت تولید کنیم. [illegible] با وصل کردن به پس متغیر بورد آردوینو، می‌توان نت‌های مختلف را پخش کرد.

تایمر ← [illegible] استفاده از tone با تابع PWM برای پین‌های 3, 11 در بوردهای غیر Mega باعث ایجاد اختلال در عملکرد می‌شود.

اسیلوسکوپ ← پیک‌های درست شده در اسیلوسکوپ متناسب با نوت در حال اجرا هستند. هرچه صدا بزرگتر باشد فرکانس نوت پایین‌تر است و بالعکس. هرچه صدا زیرتر باشد طول موج در حالت فعال کمتر است و بالعکس.

lab9 > src > main.cpp > getNoteDuration(int)

```cpp
#include <Arduino.h>
#include <Keypad.h>

#include "pitches.h"
#include "themes1.h"
#include "themes2.h"
#include "themes3.h"
#include "themes4.h"
#include "themes5.h"

const int POT_PIN = A0;
#define SOUND_PIN 8

// create 4*3 keypad
const byte ROWS = 4; // number of rows
const byte COLS = 3; // number of columns
char keys[ROWS][COLS] = {{'1', '2', '3'},
                         {'4', '5', '6'},
                         {'7', '8', '9'},
                         {'*', '0', '#'}};
byte rowPins[ROWS] = {50, 51, 52, 53};
byte colPins[COLS] = {49, 48, 47};
Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS);
float scale;

float *getNoteDuration(int selectedSound);
float *getMelody(int selectedSound);
int getMelodySize(int selectedSound);
```

```
30  void setup()
31  {
32  }
33
34  int selectedSound = 1;
35  void loop()
36  {
37    char keyPressed = keypad.getKey();
38    if (keyPressed)
39    {
40      if (keyPressed == '*' || keyPressed == '#' || keyPressed == '0') // the play the selected sound
41      {
42        float *melody = getMelody(selectedSound);
43        float *noteDurations = getNoteDuration(selectedSound);
44        for (int note = 0; note < getMelodySize(selectedSound); note++)
45        {
46          scale = analogRead(POT_PIN) / 512.0;
47          int duration = 800.0 / (noteDurations[note]);
48          tone(SOUND_PIN, (int)(melody[note] * scale), duration);
49          delay((int)(1.2*duration));
50        }
51      }
52      else
53      {
54        selectedSound = int(keyPressed - '0');
55      }
56    }
57  }
```

```
59  float *getNoteDuration(int selectedSound)
60  {
61    switch (selectedSound)
62    {
63    case 1:
64      return noteDurations1;
65      break;
66    case 2:
67      return noteDurations2;
68      break;
69    case 3:
70      return noteDurations3;
71      break;
72    case 4:
73      return noteDurations4;
74      break;
75    case 5:
76      return noteDurations5;
77      break;
78
79    default:
80      return noteDurations1;
81      break;
82    }
83  }
```

```
84  float *getMelody(int selectedSound)
85  {
86      switch (selectedSound)
87      {
88      case 1:
89          return melody1;
90          break;
91      case 2:
92          return melody2;
93          break;
94      case 3:
95          return melody3;
96          break;
97      case 4:
98          return melody4;
99          break;
100     case 5:
101         return melody5;
102         break;
103     default:
104         return melody5;
105         break;
106     }
107 }
```

```
int getMelodySize(int selectedSound)
{
  switch (selectedSound)
  {
  case 1:
    return sizeof(melody1) / sizeof(int);
    break;
  case 2:
    return sizeof(melody2) / sizeof(int);
    break;
  case 3:
    return sizeof(melody3) / sizeof(int);
    break;
  case 4:
    return sizeof(melody4) / sizeof(int);
    break;
  case 5:
    return sizeof(melody5) / sizeof(int);
    break;
  default:
    return sizeof(melody1) / sizeof(int);
    break;
  }
}
```