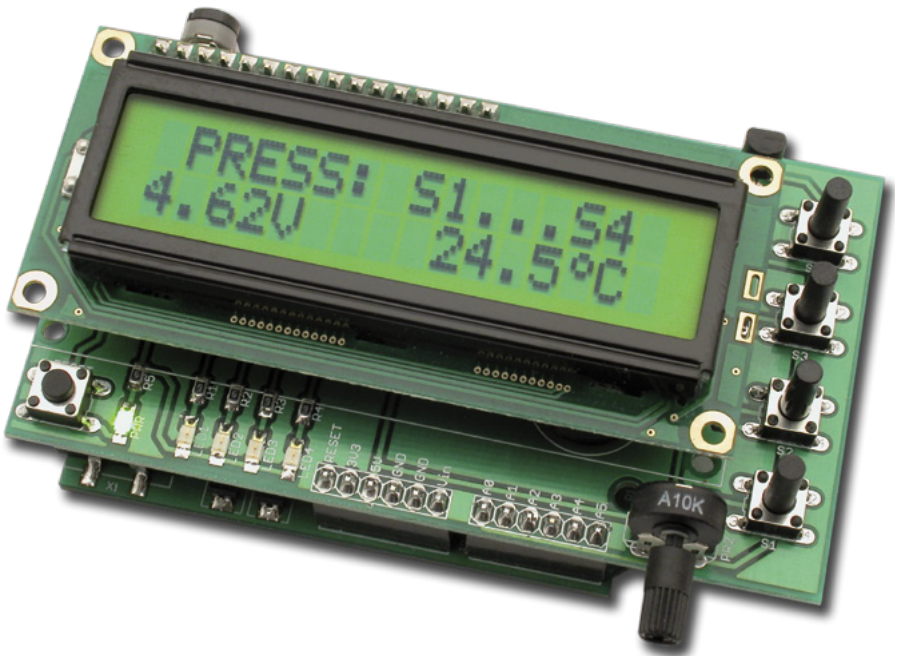
Kurs programowania Arduino (9)

Generator PWM, generowanie dźwięku i obsługa przerwań

Kontynuujemy opis praktycznych przykładów zastosowania oprogramowania i zestawu Arduino. Zaznajomienie się z nimi ułatwi tworzenie programów użytkowych, a dodatkowo, gotowe przykłady można po niewielkich modyfikacjach zastosować we własnych projektach. Przykłady zostały przygotowane z zastosowaniem zestawu Arduino UNO oraz przeznaczonych dla niego modułów AVTDuino LCD i AVTDuino LED.



Generator PWM – pulsująca LED

Sygnał PWM jest przebiegiem okresowym o zmiennym wypełnieniu. Wykorzystując sygnał PWM generowany przez mikrokontroler i uśredniając go za pomocą nieskomplikowanego filtru składającego się z rezystora i kondensatora, można wykonać przetwornik C/A, na wyjściu którego wartość analogowa (napięcie) będzie zależne od wypełnienia generowanego sygnału PWM. Do generowania sygnału PWM dostępna jest funkcja *analogWrite(pin, value)* gdzie pierwszym parametrem jest numer linii cyfrowej PWM a *value* wartością wypełnienia generowanego sygnału PWM w zakresie od 0 do 255. Z wykorzystaniem sygnału PWM można modyfikować np. jasność dołączonej diody LED czy prędkości silnika. Sygnał PWM dla mikrokontrolera ATmega168, który zamontowany jest w Arduino UNO może być generowany na pinach 3, 5, 6, 9, 10 i 11. Działanie generatora pokazane zostanie z wykorzystaniem modułu AVTDuino LCD który posiada diody LED oraz potencjometr. Przykładowy program pokazano na **listingu 5**. Program realizuje pulsującą światłem diodę LED4 która jest zasilana przebiegiem PWM. Również dioda LED3 jest zasilana przebiegiem PWM której jasność zależy od wypełnienia sygnału PWM który zmienia swoja wartość w zależności od ustawienia potencjometru, czyli jasność diody LED3 jest zależna od pozycji potencjometru.

W programie w pierwszej pętli *for* wykonywanej 255 razy z wykorzystaniem komendy *analogWrite(Led4, i)* jest zwiększana wartość wypełnienia sygnału PWM (dioda LED4 rozjaśnia się). Wartość wypełnienia ustala zmienna *i*. Zmianie ona wypełnienie sygnału PWM od 0 do 100 %. Dzięki użyciu instrukcji *delay(1)* wypełnienie zmienia się co 1 milisekundę. W kolejnej pętli jest sytu-

Dodatkowe materiały na CD/FTP:
ftp://ep.com.pl, user: 18453, pass: 5eyp1854
• poprzednie części kursu

Listing 5. Zmiana jasności świecenia diod LED za pomocą PWM

```
const int Led3 = 11;                //przypisanie aliasów do pinów portów
const int Led4 = 10;                //przypisanie aliasów do pinów portów
int wart = 0;                       //zmienna pomocnicza

void setup() {                      //funkcja inicjalizacji
 analogReference(DEFAULT);          //konfigurowanie napięcia odniesienia dla
                                    //przetwornika A/C - domyślnie napięciem
                                    //odniesienia jest VCC (5V).
}

void loop() {                       //pętla główna programu
  for (int i = 0; i < 256; i++)     //pętla wykonywana 255
  {
    analogWrite(Led4, i);           //nastawa wypełnienia PWM
    delay(1);                       //opóźnienie
  }
  for (int i = 255; i > 0; i--)     //pętla wykonywana 255 razy
  {
    analogWrite(Led4, i);           //nastawa wypełnienia PWM
    delay(1);                       //opóźnienie
  }
  wart = analogRead(A0);            //pomiar napięcia z potencjometru
  analogWrite(Led3, wart/4);        //nastawa wypełnienia PWM

}                                   //koniec pętli głównej programu
```

Listing 6. Generowanie przykładowej melodii

```
#define NOTE_B0  31                                          //definicje częstotliwości nut
#define NOTE_C1  33
#define NOTE_CS1 35
#define NOTE_D1  37
#define NOTE_DS1 39
#define NOTE_E1  41
#define NOTE_F1  44
#define NOTE_FS1 46
#define NOTE_G1  49
#define NOTE_GS1 52
#define NOTE_A1  55
#define NOTE_AS1 58
#define NOTE_B1  62
#define NOTE_C2  65
#define NOTE_CS2 69
#define NOTE_D2  73
#define NOTE_DS2 78
#define NOTE_E2  82
#define NOTE_F2  87
#define NOTE_FS2 93
#define NOTE_G2  98
#define NOTE_GS2 104
#define NOTE_A2  110
#define NOTE_AS2 117
#define NOTE_B2  123
#define NOTE_C3  131
#define NOTE_CS3 139
#define NOTE_D3  147
#define NOTE_DS3 156
#define NOTE_E3  165
#define NOTE_F3  175
#define NOTE_FS3 185
#define NOTE_G3  196
#define NOTE_GS3 208
#define NOTE_A3  220
#define NOTE_AS3 233
#define NOTE_B3  247
#define NOTE_C4  262
#define NOTE_CS4 277
#define NOTE_D4  294
#define NOTE_DS4 311
#define NOTE_E4  330
#define NOTE_F4  349
#define NOTE_FS4 370
#define NOTE_G4  392
#define NOTE_GS4 415
#define NOTE_A4  440
#define NOTE_AS4 466
#define NOTE_B4  494
#define NOTE_C5  523
#define NOTE_CS5 554
#define NOTE_D5  587
#define NOTE_DS5 622
#define NOTE_E5  659
#define NOTE_F5  698
#define NOTE_FS5 740
#define NOTE_G5  784
#define NOTE_GS5 831
#define NOTE_A5  880
#define NOTE_AS5 932
#define NOTE_B5  988
#define NOTE_C6  1047
#define NOTE_CS6 1109
#define NOTE_D6  1175
#define NOTE_DS6 1245
#define NOTE_E6  1319
#define NOTE_F6  1397
#define NOTE_FS6 1480
#define NOTE_G6  1568
#define NOTE_GS6 1661
#define NOTE_A6  1760
#define NOTE_AS6 1865
#define NOTE_B6  1976
#define NOTE_C7  2093
#define NOTE_CS7 2217
#define NOTE_D7  2349
#define NOTE_DS7 2489
#define NOTE_E7  2637
#define NOTE_F7  2794
#define NOTE_FS7 2960
#define NOTE_G7  3136
#define NOTE_GS7 3322
#define NOTE_A7  3520
#define NOTE_AS7 3729
#define NOTE_B7  3951
#define NOTE_C8  4186
#define NOTE_CS8 4435
#define NOTE_D8  4699
#define NOTE_DS8 4978

int piezo = 8;                                               //linia do której dołączono głośniczek PIEZO

int melodia[] = {
  NOTE_C4, NOTE_G3,NOTE_G3, NOTE_A3, NOTE_G3,0, NOTE_B3, NOTE_C4};  //tablica nut przykladowej melodii

int czas_trwania[] = {
  4, 8, 8, 4,4,4,4,4 };                                      //tablica czasu trwania nut

void setup() {                                               //procedura konfiguracyjna
  pinMode(piezo, OUTPUT);                                    //linia portu z PIEZO jako wyjściowa
}
```

acja odwrotna. Wartość *i* zmienia się od 255 do 0, co daje zmianę wypełnienia od 100 do 0% i dioda LED przygasa. Cykliczne, naprzemienne wykonywanie obu pętli powoduje efekt pulsującego światła. Komenda *wart = analogRead(A0)* odczytuje wartość z potencjometru do zmiennej *wart*. W kolejnej instrukcji *analogWrite(Led3, wart / 4)* wartość odczytana z A/C jest dzielona przez 4 i używana jako nastawa wypełnienia. Daje to zależność jasności diody Led3 od ustawienia potencjometru. Warto wspomnieć, że z użyciem sygnału PWM można w łatwy sposób wykonać przetwornik C/A.

Generator melodii

W systemie Arduino dostępne są instrukcje umożliwiające generowanie dźwięku za pomocą dołączonego głośniczka. Do jego generowania śluzy komenda *tone(pin, freq, tim)*, której parametrami są numer portu wyjściowego sygnału audio, częstotliwość sygnału oraz czas. Trzeci parametr jest opcjonalny. Uruchomienie programu z **listingu 6**

Listing 7. Przykład funkcji obsługi przerwania Timera 1

```
#include "TimerOne.h"                                //biblioteka funkcji Timera1

const int Led1 =  13;                                //aliasy wyprowadzeń portów
const int Led2 =  12;
const int Led3 =  11;
const int SW4 =  0;

byte f_led1 = 0;                                     //flaga diody Led1
byte f_led2 = 0;                                     //flaga diody Led2
byte f_led3 = 0;                                     //flaga diody Led3

void setup() {                                       //funkcja inicjalizacji
  pinMode(Led1, OUTPUT);                             //Konfigurowanie linii sterujących
                                                     //LED
  pinMode(Led2, OUTPUT);
  pinMode(Led3, OUTPUT);
  pinMode(SW4, INPUT);                               //konfigurowanie linii z przyciskiem
                                                     //SW4
  digitalWrite(SW4, HIGH);                           //dołączenie do SW4 rezystora
                                                     //podciągającego
  digitalWrite(Led1, HIGH);                          //wyłączenie diod LED
  digitalWrite(Led2, HIGH);
  digitalWrite(Led3, HIGH);

                                                     //konfigurowanie przerwania
                                                     //zewnętrznego od SW4
                                                     //wywołującego procedurę on_off
                                                     //przy opadającym zboczu
  attachInterrupt(SW4, on_off, FALLING);
                                                     //inicjalizacja Timera 1 -
                                                     //przerwanie co 500 ms
  Timer1.initialize(500000);
                                                     //uruchomienie przerwania od Timera
                                                     //1 w procedurze int_led3
  Timer1.attachInterrupt(int_led3);
}

void loop() {                                        //pętla główna programu
  digitalWrite(Led1, f_led1);                        //zapis stanu Led1
  digitalWrite(Led2, f_led2);                        //zapis stanu Led2
  f_led2 = !f_led2;                                  //zmiana stanu na przeciwny flagi
                                                     //led2
  delay(100);                                        //opóźnienie 100 ms
}                                                    //koniec pętli głównej programu

                                                     //**** procedura obsługi przerwania
                                                     //zewnętrznego ****
void on_off()
{
  f_led1 = !f_led1;                                  //zmiana na przeciwny stanu flagi
                                                     //dla Led1
}

                                                     //**** procedura obsługi przerwania
                                                     //Timera 1 ****
void int_led3() {
  digitalWrite(Led3, f_led3);                        //zapis stanu Led3
  f_led3 = !f_led3;                                  //zmiana na przeciwną stanu flagi
                                                     //dla Led3
};
```

powoduje wygenerowanie prostej, przykładowej melodii. Do generowania dźwięku wykorzystano komendę *Tone()* oraz *noTone()*, która wyłącza generator dźwięku.

W pierwszej kolejności, w programie zdefiniowano częstotliwości nut. Następnie do linii 8 przypisano nazwę (alias) *piezo*, zdefiniowano tablice *melodia* z nutami wygrywanej melodii oraz *czas_trwania* z wartościami czasu trwania nut. W funkcji konfiguracyjnej linia, do której dołączono głośniczek jest skonfigurowana jako wyjściowa. Odtwarzanie melodii odbywa się w pętli *for* co 300 ms. Do zmiennej *czas* jest wstawiana obliczona wartość trwania nuty. Generowanie dźwięku odbywa się z użyciem funkcji *tone()*. Obliczana jest również pauza pomiędzy nutami, która z użyciem funkcji *delay()* wprowadza opóźnienie pomiędzy nutami. Funkcja *noTone()*, której parametrem jest numer wyprowadzenia z głośniczkiem, powoduje wyłączenie generatora dźwięku.

Przerwania wewnętrzne oraz zewnętrzne

Mikrokontroler umożliwia wykonywanie procedur obsługi przerwań czyli podprogramów, których wykonanie musi odbyć się natychmiast po zaistnieniu określonego zdarzenia. Wtedy wykonywanie programu głównego jest przerywane na czas realizacji podprogramu obsługi przerwania.

Przykład programu zamieszczony na **listingu 7** ilustruje sposób obsługi przerwań zewnętrznych zgłaszanych za pomocą przycisku S4 oraz wewnętrznych wywoływanych co 500 ms przez Timer1. Przerwanie zewnętrzne powoduje zaświecenie się lub zgaszenie diody Led1, natomiast przerwanie Timera 1 powoduje miganie diody Led3. Program główny steruje diodą Led2. Do obsługi przerwania zewnętrznego wykorzystywane są funkcje włączające przerwania *attachInterrupt(interrupt, function, mode)* oraz wyłączające przerwanie *detachInterrupt(interrupt)*. Parametr *interrupt* jest numerem portu, od którego jest zgłaszane przerwanie. Parametr *function* to nazwa funkcji, która będzie wykonywana po zaistnieniu przerwania, a *mode* określa moment zgłoszenia przerwania. Parametr *mode* ma następujące wartości:

- *LOW* – wywoływane gdy pin posiada stan niski,
- *CHANGE* – wywoływane gdy pin zmieni stan,
- *RISING* – wywoływane przy narastającym zboczu sygnału,
- *FALLING* – wywoływane przy opadającym zboczu sygnału.

Wewnętrzne przerwanie zgłaszane przez Timer 1 jest obsługiwane z użyciem biblioteki *TimerOne*. Umożliwia ona konfigurowanie przerwania za pomocą komendy *Timer1.initialize(czas)*, gdzie *czas* jest podawany w mikrosekundach i określa, co ile będzie wywoływane przerwanie od Timera 1. Komenda *Timer1.attachInterrupt(funkcja)* jest wykorzystywana do wskazania funkcji wywoływanej jako procedura obsługi przerwania.

W programie przykładowym z list. 7 w pierwszej kolejności są konfigurowane linie portów. Następnie przerwanie zewnętrzne jest konfigurowane za pomocą funkcji *attachInterrupt(SW4, on_off, FALLING)*. Dzięki temu naciśnięcie przycisku SW4 powoduje wywołanie funkcji *on_off* przy opadającym zboczu sygnału. W procedurze *on_off* następuje zmiana na przeciwną zmiennej *f_led1*. W zależności od tej zmiennej, w programie głównym będzie ustawiane lub zerowane wyprowadzenie sterujące diodą Led1.

Komenda *Timer1.initialize(500000)* konfiguruje Timer1, tak aby zgłaszał przerwanie co 500 ms. Procedura *Timer1.attachInterrupt(int_led3)* ustala, która funkcja będzie wywoływana podczas przerwania. W tym wypadku jest to *int_led3*, w której jest zmieniany stan zmiennej *f_led3* na przeciwny i w zależności od niego jest zaświecana lub gaszona dioda Led3.

W programie głównym co 100 ms zmienia się stan *f_led2*, co steruje migotaniem diody Led2. W ten sposób można zauważyć, że mimo wykonywania przez CPU programu głównego powodującego miganie Led2, miga również Led3 (przerwanie Timera 1) i jest możliwość zaświecenia lub zgaszenia przyciskiem SW4 diody Led1 (przerwanie zewnętrzne).

Listing 6. c.d.

```
void loop() {                                                    //pętla główna programu
  for (int nuta = 0; nuta < 8; nuta++) {                         //pętla wygrywania melodii
    int czas = 1000/czas_trwania[nuta];                          //obliczenie czasu trwania nuty
    tone(piezo, melodia[nuta],czas);                             //generowanie dźwięku
    int pausa = czas * 1.30;                                     //obliczenie czasu pauzy
    delay(pausa);                                                //czas pauzy
    noTone(piezo);                                               //wyłączenie dźwięku
  }
  delay(300);                                                    //opóźnienie 300 ms
}
```

Listing 8. Przykład obsługi pamięci EEPROM wbudowanej w mikrokontroler

```
#include <EEPROM.h>                        //biblioteka obsługi pamięci EEPROM
#include <LiquidCrystal.h>                 //biblioteka obsługi LCD

LiquidCrystal lcd(8, 9, 4, 5, 6, 7);       //konfigurowanie I/O dla LCD

byte wart;                                 //zmienna dla wartości odczytanej z EEPROM

void setup() {                             //funkcja inicjalizacji
  lcd.begin(16, 2);                        //rozdzielczość wyświetlacza LCD
}

void loop() {                              //pętla główna programu
  for (int i = 0; i < 512; i++)            //pętla czyszcząca zawartość EEPROM
  {
    EEPROM.write(i, 0);                    //zerowanie komórki wskazywanej przez zmienną i
    lcd.setCursor(0, 0);                   //kursor na początek ekranu LCD
    lcd.print("Clear EEPROM");             //wyświetlenie komunikatu
    delay(1);                              //opóźnienie 1 ms;
  }
  for (int i = 0; i < 512; i++)            //pętla zapisująca dane do EEPROM
  {
    EEPROM.write(i, i);                    //zapamiętanie i pod adresem wskazywanym przez i
    lcd.setCursor(0, 0);                   //kursor na początek ekranu LCD
    lcd.print("Write EEPROM");             //wyświetlenie komunikatu
    lcd.setCursor(0, 1);                   //kursor w 1 kolumnie 2 wiersza
    lcd.print(i,DEC);                      //wyświetlenie zapamiętywanej liczby
    delay(50);                             //opóźnienie 50 ms
  }

  for (int i = 0; i < 512; i++)            //pętla odczytująca dane z EEPROM
  {
    wart = EEPROM.read(i);                 //odczyt komórki pamięci do zmiennej wart o adresie wskazywanym przez zmienna i
    lcd.clear();                           //czyszczenie LCD
    lcd.setCursor(0, 0);                   //kursor na początek ekranu LCD
    lcd.print("Read EEPROM");              //komunikat wyświetlany w 1 linii LCD
    lcd.setCursor(0, 1);                   //ustawienie kursora w 1 kolumnie 2 wiersza
    lcd.print(wart,DEC);                   //wyświetlenie na LCD liczby z EEPROM
    delay(100);                            //opóźnienie 100 ms
  }
  lcd.clear();                             //czyszczenie LCD
  lcd.setCursor(0, 0);                     //ustawienie kursora na początek LCD
  lcd.print("END");                        //wyświetlenie napisu END
  while(1);                                //nieskończona pętla
}                                          //koniec pętli głównej programu
```

Obsługa pamięci EEPROM

Dość często, gdy będzie potrzebne nieulotne zapamiętanie ważnych danych, jest wykorzystywana pamięć EEPROM wbudowana w mikrokontroler. Do obsługi pamięci EEPROM jest przeznaczona biblioteka *EEPROM*, która ma dwie funkcje: zapisu – *EEPROM.write(address, value)* oraz odczytu – *EEPROM.read(address)*.

Argumentami wywołania funkcji zapisu są adres komórki oraz zapisywana wartość, natomiast funkcji odczytu jedynie adres, a jej ciało zwraca wartość komórki o podanym adresie. Mikrokontroler zastosowany w Arduino UNO ma pamięć EEPROM o wielkości 512 bajtów. Przykładowy program obsługujący pamięć EEPROM pokazano na **listingu 8**. Program czyści pamięć EEPROM, a następnie zapisuje do niej kolejno wartości od 0 do 255. Następnie odczytuje wartości z całej pamięci EEPROM i wyświetla je na wyświetlaczu LCD modułu *AVTDuino LCD*.

Pamięć EEPROM jest czyszczona przez zapisanie do każdej jej komórki wartości 0 (instrukcja *EEPROM.write(i, 0)*). Zmienna *i* zawiera adres zapisywanej komórki w pamięci EEPROM. W kolejnej pętli *for* zapisywane są do pamięci wartości od 0 do 255 (instrukcja *EEPROM.write(i, i)*). Jednocześnie na wyświetlaczu LCD jest pokazywana informacja o numerze zapisywanej komórki. W ostatniej pętli *for* następuje odczyt komórek pamięci EEPROM z wykorzystaniem komendy *wart = EEPROM.read(i)*, dzięki której odczytana wartość jest zapisywana do zmiennej *wart*, a następnie wyświetlana na wyświetlaczu LCD.

Marcin Wiązania, EP