

Systemy Wbudowane

Arduino C

Wersja 2022

dr inż. Marek Wilkus
Wydział Inżynierii Metali i Informatyki Przemysłowej
AGH Kraków

<http://home.agh.edu.pl/~mwilkus>

1

Arduino C - stałe

- Unikać redefiniowania istniejących stałych.
- Stosowane dla polepszenia zrozumiałości kodu.
 - Lepiej `HIGH` niż `0x01`
 - Lepiej `INPUT_PULLUP` niż `0x2`
- Uwzględniają wielkość liter.

2

Arduino C - Stałe

- true, false
- HIGH
 - Stan wysoki (wyjście)
 - Włączenie 20K rezystorów podciągających (wejście).
- LOW
 - Stan niski (wyjście)
 - Wyłączenie 20K rezystorów podciągających (wejście).
- INPUT, OUTPUT, INPUT_PULLUP

3

Arduino C - Stałe

- INPUT_PULLUP:


```
pinMode(12, INPUT);
digitalWrite(12, HIGH);
```
- LED_BUILTIN – wyprowadzenie, do którego podłączony jest LED diagnostyczny, w Arduino Uno pin 13.

4

Arduino C - Stałe

- Matematyczne:
 - `#define PI` 3.1415926535897932384626433832795
 - `#define HALF_PI` 1.5707963267948966192313216916398
 - `#define TWO_PI` 6.283185307179586476925286766559
 - `#define DEG_TO_RAD` 0.017453292519943295769236907684886
 - `#define RAD_TO_DEG` 57.295779513082320876798154814105
 - `#define EULER` 2.718281828459045235360287471352

5

Funkcje matematyczne

- `min(x,y)`, `max(x,y)`
- `abs(x)`
- `constrain(x,a,b)`
 - x – gdy $x \in [a..b]$
 - a – gdy $x < a$
 - b – gdy $x > b$
- `map(liczba,fromLow,fromHigh,toLow,toHigh)`
 - Mapuje wartość z jednego przedziału na inny
 - **Nie obsługuje liczb zmiennoprzecinkowych!**
- `pow(podstawa, wykładnik)`, `sqrt(x)`
- `radians(deg)`, `degrees(rad)`
- `sq(x) = x*x`

6

Funkcja delay i jej konsekwencje

- Istnieją dwie metody projektowania programu wykonywanego okresowo:
 - Korzystając z delay – w przyszłości trudny do rozbudowy,
 - Korzystając z licznika – wymaga odpowiedniego zaprojektowania (jako maszyna stanowa), lecz umożliwia łatwą rozbudowę,

```
void loop()
{
  digitalWrite(ledPin, HIGH);
  delay(500);
  digitalWrite(ledPin, LOW);
  delay(500);
}

unsigned long previousMillis=0;
void loop()
{
  if (millis() - previousMillis >= 500)
  {
    previousMillis = millis();

    digitalWrite(ledPin, !digitalRead(ledPin));
  }
  //!!!
}
```

Wersja z delay:
- W trakcie 500ms nie możemy nic zrobić (poza przerwaniem)

Wersja z millis – kod otwarty na rozbudowę ⁷

Operacje na bitach

- bitSet(x,n) – zapisuje 1 na bit n zmiennej x.
- bitClear(x,n) – zapisuje 0.

```
byte k=0;
```

```
bitSet(k,2);
bitSet(k,3);
```

```
bitClear(k,2);
```

```
0 0 0 0 0 0 0 0
Bit 7                               Bit 0
```

```
0 0 0 0 1 1 0 0
Bit 7                               Bit 0
```

```
0 0 0 0 1 0 0 0
Bit 7                               Bit 0
```

8

Operacje na bitach

- bitWrite(x,n,b) – zapisuje n-ty bit zmiennej x wartością b (1/0 lub HIGH/LOW).
- bitRead(x,n) – odczytuje wartość n-tego bitu zmiennej x.

9

Czas

- delay(x), delayMicroseconds(x)
- micros() - liczba μ s od uruchomienia Arduino (lub ostatniego przeładowania tego licznika).
- millis() - jak micros(), w milisekundach.

```
byte ledState = LOW;
long previousMillis = 0;
long interval = 500;

...

void loop()
{
  unsigned long currentMillis = millis();
  if(currentMillis - previousMillis > interval)
  {
    previousMillis = currentMillis;
    if (ledState == LOW)
      ledState = HIGH;
    else
      ledState = LOW;

    digitalWrite(ledPin, ledState);
  }
}
```

10

Jeszcze raz przerwania

- Tymczasowe wyłączenie przerw na wskazanym fragmencie kodu:

```
noInterrupts();
...
interrupts();
```

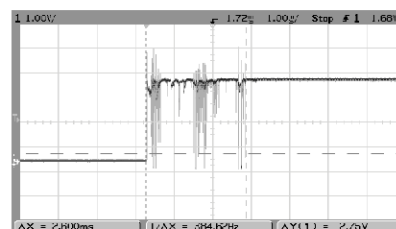
- Wyłączenie obsługi przerwania:

```
detachInterrupt(nrPrzerwania);
```

11

Debouncing

- Wciśnięcie przycisku nie skutkuje z reguły pojedynczą zmianą stanu.
- Styki przycisku uderzają o siebie, odbijając się i przez niewielki czas (0.1-1ms) dają zmienne stany,
- Działanie oparte o ilość wciśnień przycisku działa więc w sposób niekontrolowany.
- Niezbędna jest eliminacja drgań.
- Można zrealizować ją sprzętowo (kondensator), lub programowo.



- Założenie: Sprawdzamy okresowo stan linii.
- Jeżeli linia pozostaje zmieniona przez konkretną ilość sprawdzeń → przycisk rzeczywiście został wciśnięty/puszczony.
- Wymagania:
 - Zapisanie dotychczasowego stanu przycisku,
 - Licznik ilości sprawdzeń.
 - Zewnętrzna zmienna do zapisu i końcowego stanu przycisku
- W implementacji możemy zmieścić się w byte przez użycie pól bitowych!

13

- ALGORYTM:
 - jeżeli (aktualny stan przycisku != zapisany stan)
 - licznik++;
 - jeżeli (licznik >= 4)
 - zapisany stan = aktualny stan przycisku
 - ustaw odpowiednio zewnętrzną zmienną
 - licznik = 0
 - W przeciwnym wypadku
 - licznik = 0

14

- ALGORYTM:
 - jeżeli (aktualny stan przycisku != zapisany stan)
 - licznik++;
 - jeżeli (licznik >= 4)
 - zapisany stan = aktualny stan przycisku
 - ustaw odpowiednio zewnętrzną zmienną**
 - licznik = 0
 - W przeciwnym wypadku
 - licznik = 0

```
state=digitalRead(7);
if (!state)
{
  button1=1;
  count=0;
}
```

Zmieniamy stan na „aktywny” gdy przycisk został wciśnięty. Brak efektu powtarzania.

```
state=digitalRead(7);
button1=state;
count=0;
```

Cały czas „publikujemy” stan. Wciśnięcie powoduje b. częste powtarzanie.

15

```
//Kod dla debouncingu jednego przycisku
//Funkcja wywołuje się co iteracja głównej pętli.
//Wynik sprawdzamy w zmiennej globalnej button1.
void debounce()
{
  static byte count=0; //ilość sprawdzeń
  static byte state=0; //Stan przycisku. Zmienne STATYCZNE zachowują wartości!
  if (digitalRead(7)!=state) //Jeżeli stan na wejściu zmienił się
  {
    count++; //ilość++
    if (count>=4) //przez >=4 sprawdzenia przycisk nie zmienił swojego stanu
    {
      state=digitalRead(7); //zapisz nowy stan jako trwały
      if (!state)
      {
        button1=1; //Proszę nie używać button1=istate, bo go wyzeruje!
        //zwrucujemy my gdy uznamy, że wtkonałismy to co jest
      } //niezbędne przy przyciśniętym przycisku.
      count=0; //zliczanie możliwe od nowa
    }
  }
  else //stan pozostaje cały czas taki sam, nic nie zliczamy
  {
    count=0;
  }
}
```

16

- Użycie:


```
void loop() {
  debounce(); // Wywołanie funkcji debounce, ustalenie stanu przycisku do button1.
  if (button1) // Obsługa zdarzenia w button1
  {
    Serial.println("BUTTON1 DOWN!"); // Skasowanie „znacznika” po obsłużeniu zdarzenia.
    button1=0;
  }
}
```

Reszta kodu.

17

- Zastosowanie pól bitowych:
 - -2B z zajętego RAMu (mamy go 2kB)
 - +15-17B zajętej pamięci programu (mamy jej jednak aż 32kB)
- ```
struct button_db
{
 unsigned int count : 4; // Licznik: 4 bity, 0..15
 unsigned int state : 1; // Zapamiętany stan: 1 bit, 0..1
 unsigned int pressed : 1; // Ostateczny stan przycisku: 1 bit, 0..1
 debounced;
}
```

18

- Wbudowane pozwalają na obsługę:
  - EEPROMu,
  - Zaawansowanych protokołów szeregowych,
  - Wyświetlacza LCD,
  - Kart SD,
  - Interfejsu SPI,
  - Portu szeregowego na dowolnym wyprowadzeniu,
  - Specjalistycznych urządzeń zewnętrznych.

```
#include <EEPROM.h>

void setup()
{
 Serial.begin(9600);
}

int a=0;
int v;
int poprzednie=0;

void serialEvent()
{
 if (Serial.available(>0)
 {
 int k=Serial.parseInt();
 EEPROM.write(a,k);
 }
}
```

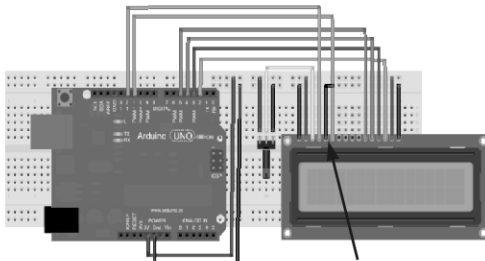
Adres w EEPROM (0)

Adres w EEPROM (0)

Wartość (bajt!)

```
void loop()
{
 v=EEPROM.read(a);
 if (poprzednie!=v)
 {
 Serial.println(v);
 poprzednie=v;
 }
}
```

- Wymagania:
  - +5V, GND
  - Zasilanie LED podświetlenia (opcjonalne)
  - Regulacja kontrastu (potencjometr)
  - Enable
  - Register Select
  - R/W (LOW)
  - 4 bity szyny danych



R/W do masy! 21

- Zapis wartości do rejestru danych + zapis rozkazu do rejestru rozkazów (linia RS)
- Dla wielu wyświetlaczy zgodnych z Hitachi
  - wbudowana biblioteka.

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup() {
 lcd.begin(16, 2);
 lcd.print("hello, world!");
 lcd.setCursor(0, 1);
 lcd.print("Dolna linia");
}
```

```
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup() {
 lcd.begin(16, 2);
 lcd.print("hello, world!");
 lcd.setCursor(0, 1);
 lcd.print("Dolna linia");
}
```

RS Enable D4 D5 D6 D7

Ilość kolumn Ilość wierszy

Kolumna wiersz

```
int i;
...
i=2;
...
lcd.print("LICZBA TO ");
lcd.print(i);
...

double k=32.23;
lcd.print(k,1);
lcd.setCursor(0,1);
lcd.print(k,2);
```



- Software'owy RS232: Biblioteka SoftwareSerial.
  - Brak buforowania.
- Użycie:



```
SoftwareSerial mySerial(2,3);
```

//w Setup:

```
mySerial.begin(4800);
```

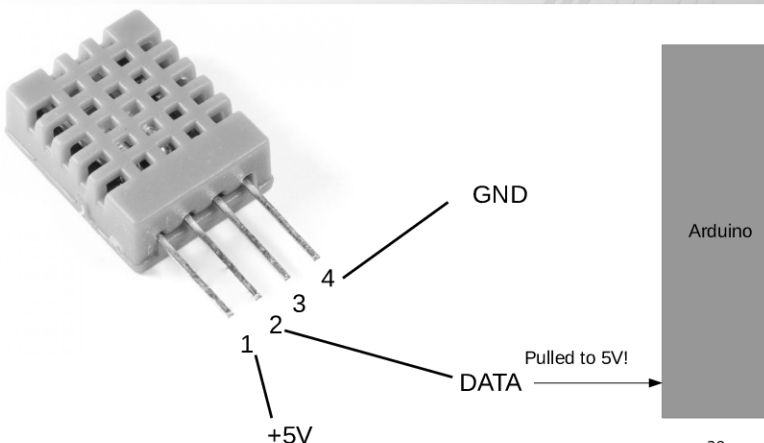
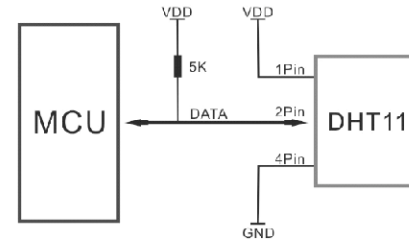
A dalej jak z typowym portem szeregowym.

```
SoftwareSerial mySerial(10, 11);
void setup()
{
 Serial.begin(9600); //inicjalizacja sprzętowego
 mySerial.begin(4800); //inicjalizacja programowego
}

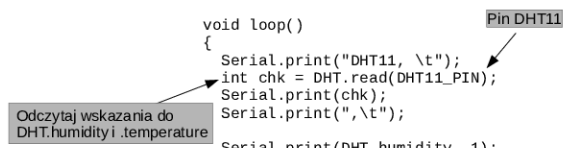
void loop()
{
 if (mySerial.available()) //Przebieg polewny z zębami
 {
 byte k=mySerial.read(); //ZMIENIY
 if (k==1) //ZMIENIY
 {
 mySerial.print(2);
 }
 if (k==R) //REO
 {
 mySerial.print(temp);
 }
 }
 if (Serial.available()) //Przebieg polewny z zębami
 {
 if (Serial.read()==0) //Polecenie: odczytaj wszystkie wyniki
 {
 Serial.print(temp); //Dobry temperatur
 Serial.print(" Lat i rowoco "); //Data nr aradokos numer...
 mySerial.print(1); //na szybko odczytaj numer odczytanej płytki
 int q=0;
 while ((mySerial.available())&&(q<400)) Serial.print(" "); //dajemy czas na budowanie odpowiedzi
 Serial.print(mySerial.read()); //...Przejrzyj pierwszy numer
 Serial.print(" it is ");
 mySerial.print(C); //odczytaj temperaturę z zewnętrznego urządzenia
 q++;
 while ((mySerial.available())&&(q<400)) Serial.print(" "); //dajemy czas na budowanie odpowiedzi
 mySerial.write(0);
 }
 Serial.print((char)mySerial.read()); //Drukujemy kolejne znaki
 delay(50); //dajemy czas między znakami
 Serial.println(); //koniec odczytania dla użytkownika
 }
}
```

- Użytkowanie do celów debuggowania?
  - Nie bez modyfikacji – W Arduino Uno połączenie USB jest podłączone na stałe do pinów 0 i 1.
- Łączenie dwóch płytek: Jak?
  - Połączyć masy
  - Rx → Tx
  - Tx → Rx
- Maksymalna prędkość?
  - Zależna od „obciążenia” procesora, bezpiecznie jest do ok. 19200bps.

- Sensor temperatury i wilgotności działający na jednoprzewodowej magistrali.
- Zasilanie: 5V
- Podłączenie: --->
- Komunikacja dwustronna
- 0..50 \*C, max 80RH
- Uwaga na pin 3/4!



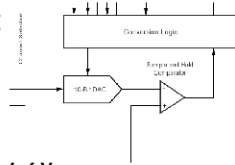
```
#include <dht11.h>
dht11 DHT;
#define DHT11_PIN 2
#define F_PIN 4
#define TEMP_MAX 29
void setup()
{
 Serial.begin(9600);
 Serial.println("Termostat ");
 Serial.println();
 pinMode(F_PIN, OUTPUT);
 digitalWrite(F_PIN, LOW);
}
void loop()
{
 Serial.print("DHT11, \t");
 int chk = DHT.read(DHT11_PIN);
 Serial.print(chk);
 Serial.print(", \t");
 Serial.print(DHT.humidity, 1);
 Serial.print(", \t");
 double temp = DHT.temperature;
 Serial.println(temp, 1);
 if (temp >= TEMP_MAX)
 {
 digitalWrite(F_PIN, HIGH);
 }
 else
 {
 digitalWrite(F_PIN, LOW);
 }
 delay(2000);
}
```





## Jak ADC mierzy (i dlaczego tak długo?)

- Mierzona próbka przechowywana jest w kondensatorze sample-and-hold.
- DAC generuje kolejne napięcia.
- Komparator wychwytuje czy wygenerowane napięcie jest większe, czy mniejsze od mierzonego.
- Końcowy wynik transmitowany jest do pamięci.
- Generowanie kolejnych napięć zajmuje (w zależności od procesora) **12-18 cykli**.



37

## Korzystanie z ADC

- Co mierzyć? Względem czego? - ADMUX

| Bit           | 7     | 6     | 5     | 4 | 3    | 2    | 1    | 0    |       |
|---------------|-------|-------|-------|---|------|------|------|------|-------|
|               | REFS1 | REFS0 | ADLAR | — | MUX3 | MUX2 | MUX1 | MUX0 | ADMUX |
| Read/Write    | R/W   | R/W   | R/W   | R | R/W  | R/W  | R/W  | R/W  |       |
| Initial Value | 0     | 0     | 0     | 0 | 0    | 0    | 0    | 0    |       |

Shift results to the left.

| REFS1 | REFS0 | Voltage Reference Selection                                         |
|-------|-------|---------------------------------------------------------------------|
| 0     | 0     | AREF, internal 1.1V reference                                       |
| 0     | 1     | AVCC with external capacitor at AREF pin                            |
| 1     | 0     | Reserved                                                            |
| 1     | 1     | Internal 1.1V voltage reference with external capacitor at AREF pin |

| MUX3:0 | Single Ended Input    |
|--------|-----------------------|
| 0000   | ADC0                  |
| 0001   | ADC1                  |
| 0010   | ADC2                  |
| 0011   | ADC3                  |
| 0100   | ADC4                  |
| 0101   | ADC5                  |
| 0110   | ADC6                  |
| 0111   | ADC7                  |
| 1000   | ADC8 <sup>(1)</sup>   |
| 1001   | (Reserved)            |
| 1010   | (Reserved)            |
| 1011   | (Reserved)            |
| 1100   | (Reserved)            |
| 1101   | (Reserved)            |
| 1110   | 1.1V D <sub>REF</sub> |
| 1111   | DV (DRE1)             |

## Korzystanie z ADC

- Czy już mierzyć? - ADCSRA

| Bit           | 7    | 6    | 5     | 4    | 3    | 2     | 1     | 0     |        |
|---------------|------|------|-------|------|------|-------|-------|-------|--------|
|               | ADEN | ADSC | ADCFE | ADIF | ADIE | ADPS2 | ADPS1 | ADPS0 | ADCSRA |
| Read/Write    | R/W  | R/W  | R/W   | R/W  | R/W  | R/W   | R/W   | R/W   |        |
| Initial Value | 0    | 0    | 0     | 0    | 0    | 0     | 0     | 0     |        |

Włączyć ADC → Start pomiaru → Pomiar na wyzwalcach → Koniec pomiaru wyzwala przerwanie → Preskaler zegara systemowego dla ADC

39

## PWM – modulacja szerokości impulsu

- Często zachodzi potrzeba stopniowej regulacji jakiegoś urządzenia:
  - Jasność LED,
  - Obroty silnika,
  - Dostarczenie konkretnej wartości napięcia
- W tym celu stosuje się piny obsługujące **PWM** - Pulse Width Modulation.
- W Arduino Uno piny obsługujące PWM zaznaczone są na płytce symbolem tyldy (~).

40

## PWM

- PWM wykorzystujemy ustawiając pin jako wyjście, a następnie określając współczynnik wypełnienia (0..255) funkcją **analogWrite**:

```
analogWrite(pin, współczynnik);
```

41

## PWM

- `analogWrite(PIN,20):`



- `analogWrite(PIN,127):`



- `analogWrite(PIN,255):`



42

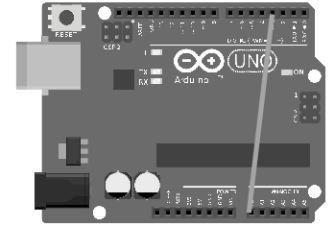
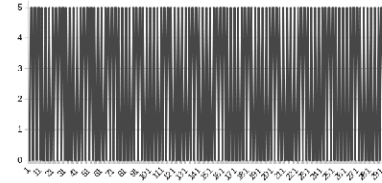


- PWM wykorzystujemy tam, gdzie możemy zapewnić **bezwładność** sterowanego układu:
  - Bezwładność wzroku przy szybko migającej LED,
  - Bezwładność wału silnika,
  - Bezwładność wskazówki mechanizmu wychyłowego,
- Tam, gdzie nie możemy tego zapewnić, stosujemy odpowiednio dobrany **układ całkujący**.  
Dokładne parametry dobieramy w zależności od planowanego poboru prądu, czasu reakcji oraz akceptowalnych szumów.

- Połączenie A0 z pinem 3 i próba z PWM:

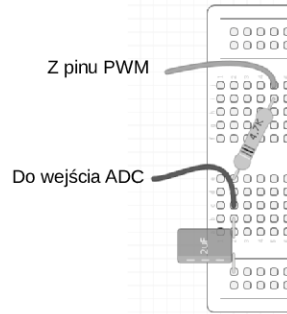
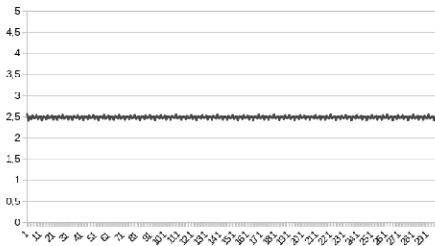
```
void wwl_p() {
 pinMode(A0, INPUT);
 pinMode(3, OUTPUT);
 Serial.begin(5690);
}

void loop() {
 analogWrite(3, 129);
 Serial.println("analogRead(A0)="+analogRead(A0));
 delay(500);
}
```



To bardzo nie wygląda jak 2.5V

- Znacznie lepszy efekt:

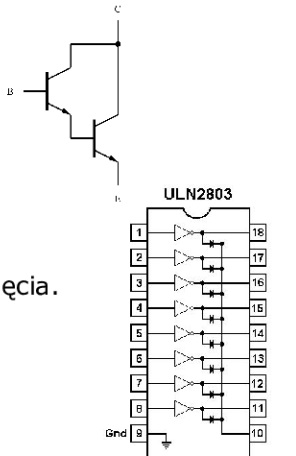


Nieco zaszumione, ale znacznie lepiej

Jak dobrać tranzystor?

- PNP/NPN,
- Si,
- Dopuszczalny prąd  $I_C (P_{tot})$ ,
- Dopuszczalne napięcie  $V_{CEO}$
- Parametry termiczne.

- ULN2803
  - 8x sterownik na tranzystorach w układzie Darlingtona.
  - Po podaniu 1 na wejście - na wyjściu 0.
  - Max 500mA/wyjście (moc !)
  - Max 50V
  - Pin 10 - do przełączanego napięcia.



- Następny wykład:
  - DHT11 - jak to w zasadzie działa?,
  - Dodatkowe interfejsy i urządzenia.
  - Podłączanie urządzeń o wysokim poborze mocy.