Kurs Arduino (1)
Język programowania

Rozpoczynamy naukę programowania Arduino. W pierwszej kolejności zajmiemy się specyficznym językiem Arduino, który dostępnymi bibliotekami oraz składnią niewiele różni się od języka C. Dlatego też preferowana jest podstawowa znajomość składni i komend języka C oraz ich użycia. W kolejnych częściach kursu zapoznamy się z zestawem Arduino UNO i jego uruchomieniem w środowisku Arduino IDE. W kolejnych częściach kursu zaprezentujemy sposób programowania w tym systemie na podstawie praktycznych przykładów.

Jak wspomniano, język Arduino IDE jest zbliżony do języka C. Jego komendy umieszczono w **tabeli 1**. Składa się on ze struktur, zmiennych oraz funkcji. W strukturach *Setup()* oraz *Loop()*, wymaganych przez język Arduino, będzie się znajdował program. Pozostałe struktury kontrolne, arytmetyczne, bitowe czy logiczne pokazane w tab. 1 są identyczne, jak dla języka C.

W programowaniu w każdym języku są wykorzystywane zmienne i związane z nimi typy danych oraz stałe. W języku Arduino, oprócz standardowych stałych dla języka C, są dostępne dodatkowe stałe *LOW*, *HIGH*, *INPUT* oraz *OUTPUT* związane z operacjami na liniach portów mikrokontrolera. Natomiast typy zmiennych są identyczne jak dla języka C. Nowością w języku Arduino są dostępne funkcje związane z mikrokontrolerem. Dostępne są funkcje wykonujące operacje na portach mikrokontrolera. Pierwsza z funkcji *pinMode(pin, mode)* umożliwia konfigurowanie poszczególnych wyprowadzeń portów mikrokontrolera, ustalanie czy dana nóżka ma być wejściem czy wyjściem. Pozostałe funkcje *digitalWrite()* oraz *digitalRead()* dotyczą zapisu lub odczytu wartości linii portu.

Kolejnymi funkcjami są funkcje dotyczące obsługi analogowych linii portów mikrokontrolera. Składają się one z funkcji *analogReference()*, *analogRead()* i *analogWrite()* odpowiednio: ustalających napięcie odniesienia dla przetwornika A/C, funkcji odczytu zmierzonej wartości analogowej i zapisu wartości analogowej (sygnał PWM).

Funkcje należące do grupy zaawansowanych służą odpowiednio do generowania tonu audio na dowolnej linii portu mikrokontrolera, generowania strumienia bitów oraz odczytu długości impulsu na linii mikrokontrolera. Prawdopodobnie często będą wykorzystywane funkcje służące do odmierzania czasu. Umożliwiają one wstawienie w programie opóźnień oraz wykonywanie pomiaru czasu. Arduino IDE ma również funkcje matematyczne, trygonometryczne czy funkcje generatorów pseudolosowych. Z mikrokontrolerami związane są nieodzownie operacje na bajtach oraz bitach. Dlatego też Arduino udostępnia funkcje związane z bitami i bajtami. Umożliwiają one zapis bajtów, ich odczyt oraz ustawianie/kasowanie i odczyt dowolnych bitów zmiennych. Są to bardzo pomocne funkcje przydatne w operowania na portach mikrokontrolera.

Kolejne funkcje są przeznaczone do obsługi przerwań. Umożliwiają one przerwanie pracy programu głównego i wykonanie bardziej priorytetowego zadania. Dostępne są funkcje obsługi przerwań zewnętrznych zgłaszanych od linii portów mikrokontrolera oraz wewnętrznych przerwań zgłaszanych przez peryferia mikrokontrolera jak czasomierzy czy interfejsów komunikacyjnych.

Ostania z dostępnych funkcji języka Arduino jest funkcją obsługi transmisji szeregowej zgodnej z RS232. Będzie ona bardzo pomocna podczas komunikacji mikrokontrolera np. z komputerem lub innym urządzeniem zgodnym i wyposażonym w interfejs RS232. Mogą to być np. moduły Bluetooth, GPS czy GSM. Dostępne funkcje języka Arduino jak i inne instrukcje pokazane w **tab. 1** będą dokładniej opisane i pokazane z użyciu podczas praktycznych przykładów ich wykorzystania.

Biblioteki

Oprócz dostępnych instrukcji języka Arduino dostępne są liczne biblioteki funkcji umożliwiających obsługę różnych układów dołączanych do mikrokontrolera. Część z nich wymieniono w **tabeli 2**.

Są dostępne dwie grupy bibliotek – biblioteki dostępne z systemem Arduino czyli biblioteki standardowe oraz pozostałe niestandardowe utworzone przez innych użytkowników systemu Arduino. Wśród standardowych bibliotek dostępne są biblioteki funkcji obsługi pamięci EEPROM, komunikacji z komputerem, obsługi wyświetlaczy LCD, transmisji sieciowej ETHERNET, obsługi kart pamięci SD, silników krokowych, programowej wersji interfejsu RS232 czy obsługi interfejsów SPI i I²C/TWI, w które został wyposażony w mikrokontroler. Do niektórych bibliotek standardowych wymagane będą elementy sprzętowe, jak choćby wyświetlacz LCD czy kontroler Ethernet. Jak wspomniano, dostępne są również biblioteki niestandardowe, które można ściągnąć z Internetu. Biblioteki niestandardowe można podzielić na kilka grup. W grupie bibliotek komunikacyjnych można znaleźć biblioteki umożliwiające obsługę wiadomości tekstowych, obsługi interfejsu 1Wire, klawiatury z interfejsem PS2, obsługi telefonu komórkowego czy serwera www. Dostępne są również biblioteki umożliwiające komunikacje zestawów Arduino ze sobą. W grupie bibliotek obsługujących czujniki są dostępne biblioteki obsługujące czujniki pojemnościowe oraz przyciski w jakie jest wyposażona większość urządzeń. Dostępna jest również grupa bibliotek obsługujących wyświetlacze graficzne oraz wyświetlacze wielosegmentowe LED również z wykorzystaniem kontrolerów firmy MAXIM. Biblioteki w grupie generatory umożliwiają generowanie sygnału na dowolnym pinie mikrokontrolera lub z wykorzystaniem scalonych generatorów PWM. Dostępna jest również grupa bibliotek dotyczących czasu. Można w niej znaleźć bibliotekę obsługującą zegar oraz kalendarz – bardzo przydatna biblioteka, gdy będzie wymagany zegar i kalendarz i związana z tym np. rejestracja danych ze znacznikiem czasu rejestracji. Pozostałe biblioteki związane są z odmierzaniem czasu. Ostatnia grupa dostępnych bibliotek dotyczy bibliotek do obsługi tekstów czyli łańcuchów znaków przydatnych podczas wyświetlania tekstowych komunikatów na wyświetlaczu LCD lub wysyłanych do komputera. Jak widać dostępna jest pokaźna liczba bibliotek, która cały czas jest rozwijana. W Internecie można znaleźć wiele innych niestandardowych bibliotek dla Arduino umożliwiających obsługę wielu układów dołączanych do mikrokontrolera. Biblioteki niestandardowe zawsze w pierwszej kolejności należy zainstalować. Składają się one z jednego pliku z przedrostkiem .h oraz jednego .cpp. W ramach kursu będą dokładniej opisywane tylko biblioteki wy-

Listing. 1. Szkic programy w Arduino

```
int buttonPin = 3;  //inicjacja zmiennej

void setup()  //struktura inicjalizująca
{
  Serial.begin(9600);
  pinMode(buttonPin, INPUT);
}

void loop()   //nieskończona pętla programu
{
  // ...
}
```

korzystywane w przykładowych projektach i związanymi z dołączanymi do zestawu Arduino UNO modułami AVTDUINO.

Program główny w Arduino

Nieodzownymi elementami programu są zmienne w których przechowuje się dane oraz funkcje od których zależy działanie programu. Program główny systemu Arduino składa się z dwóch nieodzownych struktur *setup()* oraz *loop()*. Wygląd szkicu programu w Arduino pokazano na **listingu. 1**.

W pierwszej kolejności są inicjowane zmienne. Następnie w strukturze *setup()* inicjowane są tryby pracy linii mikrokontrolera, jego peryferia, linie portów mikrokontrolera oraz funkcje zależne od wykorzystywanych bibliotek. Struktura ta jest wykonywana tylko raz podczas włączania zasilania lub zerowania mikrokontrolera. Po strukturze inicjującej wymagana jest struktura *loop()*, która tworzy niekończoną pętle w której wykonywany jest program sterujący pracą CPU. Działanie instrukcji w pętli będzie zależeć od użytkownika i napływających informacji z otoczenia mikrokontrolera. Oczywiście jest możliwe wychodzenie z nieskończonej pętli do obsługiwanych funkcji z bibliotek lub własnych. Dla większej czytelności programu i jego opisu działania, można wprowadzać komentarze które powinny być oddzielone od instrukcji znakami „//”. Jest możliwe również wprowadzenie komentarza w znakach otwierających komentarz „/*” oraz zamykających komentarz „*/”. Wszystko pomiędzy tymi znakami jest przez język Arduino ignorowane. Komentowanie działania programu jest dobrą praktyką gdyż po pewnym czasie umożliwia to szybsze zapoznanie się z działaniem programu. Każdy przygotowany program będzie musiał być poddany kompilacji a mikrokontroler zaprogramowany utworzonym plikiem z programem.

Obsługa cyfrowych linii mikrokontrolera

Cyfrowe linie portów mikrokontrolerów mogą być skonfigurowane jako wejścia lub wyjścia. Dotyczy to również linii analogowych. W zestawie Arduino z mikrokontrolerem ATmega domyślnie linie portów są skonfigurowane jako wejścia z wyłączonym rezystorem podciągającym. Czyli domyślnie są to wejścia prawie nie pobierające prądu i bardzo często są wykorzystywane do odczytu sygnałów z czujników. W mikrokontrolerach ATmega jest możliwość programowego włączenia rezystora podciągającego, który domyślnie na linii wejściowej będzie ustalał stanu wysoki. Rozwiązanie z rezystorem podciągającym jest bardzo często wykorzystywane podczas odczytu stanu z przycisków. Jego naciśniecie, na linii wejściowej ustawi stan niski a domyślnie po jego puszczeniu będzie panował stan wysoki wymuszany przez rezystor podciągający. Cyfrowe linie mogą również być wyjściami na których stan może się zmieniać na niski lub wysoki co odpowiada napięciu 0 V i +5 V. Wydajność prądowa wyjść mikrokontrolerów ATmega umożliwia zasilanie dołączonych do nich diod LED. W przypadku większych obciążeń wymagane są dodatkowe wzmacniacze chociażby w postaci tranzystorów. Do obsługi cyfrowych linii w Arduino dostępne są trzy funkcje *pinMode()*, *digitalWrite()* i *digitalRead()*. Za pomocą funkcji *pinMode(pin, mode)* jest możliwość skonfigurowania typu linii cyfrowej – czy ma być wejściem czy wyjściem. Pierwszy parametr określa numer pinu mikrokontrolera zgodnie z opisem linii na płytce zestawu Arduino UNO. Drugi parametr *mode* może posiadać stałe parametry *INPUT* lub *OUTPUT* co wskazuje czy linia ma być wejściem, czy wyjściem.

Instrukcja:

pinMode(13, HIGH) oznacza że linia 13 mikrokontrolera będzie linią wyjściową. Funkcja *digitalWrite(pin, value)* służy do ustawiania stanu linii mikrokontrolera. Pierwszy parametr pin określa numer pinu, natomiast drugi parametr określa jaki ma być jej stan (niski czy wysoki – stałe parametry LOW lub HIGH). Instrukcja: *digitalWrite(13, LOW)* ustawia na linii 13 stan niski czyli napięcie 0V. Instrukcja *digitalWrite()* umożliwia również załączenie rezystora podciągającego na linii będącej wejściem. Aby do linii wejściowej dołączyć wewnętrzny rezystor podciągający należy z wykorzystaniem funkcji *digitalWrite()* wpisać do linii wejściowej wartość *HIGH* co pokazano na poniższym przykładzie:

pinMode(12, LOW); //Konfiguracja linii 12 jako wejściowa
digitalWrite(12, HIGH); //Włączenie rezystora podciągającego do linii 12

Powyższe dwie instrukcje powodują że linia 12 będzie linią wejściową z włączonym rezystorem podciągającym. Instrukcja *digitalRead(pin)* służy do odczytu stanu linii będącej wejściem. Parametr *pin* określa numer pinu mikrokontrolera który jest odczytywany. Funkcja zwraca stan odczytywanego stanu pinu zgodnie z przykładem: *val = digitalRead(12);*

Do zmiennej *val* zostanie zapisany stan 12 linii portu mikrokontrolera. Liczba dostępnych portów będzie zależeć od zastosowanego mikrokontrolera, choć jest również możliwość ich zwiększenia poprzez dołączenie do niego odpowiednich ekspanderów. Na płytce Arduino UNO dostępne są cyfrowe linie portów oznaczone numerami od 0 do 13. Dlatego tez dla ułatwienia właśnie tymi aliasami można się posługiwać podczas konfigurowania portów I/O.

Obsługa analogowych linii mikrokontrolera

W Arduino dostępnych jest kilka linii analogowych z wykorzystaniem których można mierzyć analogowe sygnały np. z czujników (temperatury, ciśnienia) w przedziale napięcia od 0 V do +5 V i z rozdzielczością 10 bitów. 10-bitowa rozdzielczość oznacza, że mierzone napięcie od 0 V do 5 V będzie odczytywane wartościami od 0 do 1023. Dla 5 V daje to rozdzielczość (5 V/1024) 0,0049 V (4,9 mV). Zakres rozdzielczości przetwornika można zmienić za pomocą funkcji *analogReference()*. Pomiar wartości analogowej trwa około 100 mikrosekund. Analogowe linie mikrokontrolera oznaczone w Arduino UNO są jako A0 do A5 i mogą być wykorzystane również jako linie cyfrowe. Konfiguruje się je identycznie za pomocą funkcji *pinMode()*, *digitalWrite()* i *digitalRead()* z tym że parametr *pin* jest oznaczany za pomocą aliasów A0 do A5. Na przykład aby skonfigurować linie analogowa A0 jako wyjście wystarczy komenda *pinMode (A0, OUTPUT);*.

Analogowe linie również posiadają cyfrowo załączane rezystory podciągające które można włączyć z wykorzystaniem funkcji *digitalWrite()*. Aby działało wejście analogowe mikrokontrolera musi ono być wcześniej ustawione jako wejście z wykorzystaniem funkcji *pinMode()*. Należy również wyłączyć rezystor podciągający. Do odczytu napięcia z linii analogowej mikrokontrolera służy funkcja *analogRead(pin)*. Parametrem *pin* jest linia analogowa. Na przykład komenda *val = analogRead(A2); //odczyt wartości sygnału z linii A2* powoduje odczyt wartości analogowej z linii *A2* i zapis jej do zmiennej *val*. Dostępna jest również funkcja *analogReference(type)* za pomocą której można zmienić parametry pracy przetwornika analogowo-cyfrowego mikrokontrolera. Parametr *type* określa napięcie odniesienia dla przetwornika. Dostępne są następujące opcje:

- DEFAULT: napięcie odniesienia dla przetwornika jest napięciem zasilającym mikrokontroler czyli 5 V lub 3,3 V.
- INTERNAL: wbudowane napięcie odniesienia równie 1,1 V dla ATmega168,
- INTERNAL1V1: wbudowane napięcie odniesienia równie 1,1 V dla Arduino Mega,
- INTERNAL2V56: wbudowane napięcie odniesienia równie 2,56 V dla Arduino Mega,

– EXTERNAL: zewnętrzne napięcie odniesienia dołączone do linii AREF mieszczące się w przedziale od 0 V do 5 V.

Możliwość zmiany napięcia odniesienia dla przetwornika A/C mikrokontrolera daje możliwość dostosowania się do wartości mierzonego sygnału analogowego z wymaganą rozdzielczością pomiaru.

Obsługa generatora PWM

Sygnał PWM jest to sygnał prostokątny o modyfikowanym wypełnieniu. Z wykorzystaniem sygnału PWM i jego późniejszym uśrednieniu z wykorzystaniem prostego filtra składającego się z rezystora i kondensatora można uzyskać prosty przetwornik cyfrowo-analogowy na wyjściu którego wartość analogowa (napięcie) będzie zależne od wypełnienia generowanego sygnału PWM.

Częstotliwość sygnału PWM w Arduino jest około 500 Hz. 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. Wartość 255 daje stałe napięcie 5 V, wartość 127 da wypełnienie 50%, czyli napięcie wyjściowe po uśrednieniu 2,5 V, natomiast wartość 0 da wypełnienie 0% i napięcie 0 V.

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. Na przykład funkcja *analogWrite(5, 127); //Sygnał PWM o wypełnienia 127 g*eneruje sygnał PWM na pinie 5 o wypełnieniu 50 %. Nie trzeba również ustawiać linii PWM jako wyjścia przez wywołaniem funkcji *analogWrite()* ale dla czytelności programu zalecane jest ustawienie linii PWM jako wyjście z wykorzystaniem funkcji *pinMode()*.

Tabela 1. Typy struktur, zmienne, funkcje języka Arduino

Struktury	>> (bitshift right)	Cyfrowe I/O
setup()	**Pozostałe operatory**	pinMode()
loop()	++ (increment)	digitalWrite()
Struktury kontrolne	-- (decrement)	digitalRead()
if	+= (compound addition)	Analogowe I/O
if...else	-= (compound subtraction)	analogReference()
for	*= (compound multiplication)	analogRead()
switch case	/= (compound division)	analogWrite() - PWM
while	&= (compound bitwise and)	**Zaawansowane I/O**
do... while	|= (compound bitwise or)	tone()
break	Zmienne	noTone()
continue	**Stałe**	shiftOut()
return	HIGH | LOW	pulseIn()
goto	INPUT | OUTPUT	Czasu
Składnia języka	true | false	millis()
;	integer constants	micros()
{}	floating point constants	delay()
//	**Typy zmiennych**	delayMicroseconds()
/* */	void	**Matematyczne**
#define	boolean	min()
#include	char	max()
Operacje arytmetyczne	unsigned char	abs()
= (assignment operator)	byte	constrain()
+ (addition)	int	map()
- (subtraction)	unsigned int	pow()
* (multiplication)	word	sqrt()
/ (division)	long	**Trygonometryczne**
% (modulo)	unsigned long	sin()
Operatory porównania	float	cos()
== (equal to)	double	tan()
!= (not equal to)	string - char array	**Losowe**
< (less than)	String - object	randomSeed()
> (greater than)	array	random()
<= (less than or equal to)	**Konwersje**	**Bitów i Bajtów**
>= (greater than or equal to)	char()	lowByte()
Operatory logiczne	byte()	highByte()
&& (and)	int()	bitRead()
|| (or)	word()	bitWrite()
! (not)	long()	bitSet()
Operacje na wskaźnikach	float()	bitClear()
* dereference operator	**Zmienne zakresowe**	bit()
& reference operator	variable scope	**Przerwania zewnętrzne**
Operatory bitowe	static	attachInterrupt()
& (bitwise and)	volatile	detachInterrupt()
| (bitwise or)	const	interrupts()
^ (bitwise xor)	**Narzędzia**	noInterrupts()
~ (bitwise not)	sizeof()	**Komunikacja**
<< (bitshift left)	Funkcje	Serial

Typy pamięci

W mikrokontrolerach programowanych przez Arduino czyli ATmega istnieją trzy rodzaje pamięci:

– pamięć FLASH (przestrzeń programu), przechowywany jest w niej program napisany w Arduino. Dane zapisane w tej pamięci nie są tracone po wyłączeniu zasilania,
– pamięć SRAM (Static Random Access Memory), pamięć na zmienne czyli dane z obliczeń przeprowadzanych przez mikrokontroler. Dane w tej pamięci są tracone po wyłączeniu zasilania,
– pamięć EEPROM jest pamięć do stałego przechowywania danych. Zapisane dane nie są wymazywane po wyłączeniu zasilania. Można jej używać do przechowywania długoterminowego informacji.

Dla przykładu mikrokontroler ATmega168 ma następujące rodzaje pamięci:

– FLASH – 16 kB (z czego 2 kB jest używane dla bootloadera),
– SRAM – 1024 bajtów,
– EEPROM – 512 bajtów.

Ten mikrokontroler stosunkowo małą pamięć SRAM. Już zapisanie do niej przykładowego tekstu: *char tekst[] = „Arduino – Elektronika Praktyczna";* zajmuje ponad 32 bajtów. To może nie wydawać się dużo, ale wystarczy kilka takich tekstów, aby zapełnić 1024 bajty pamięci. Zwłaszcza, gdy jest duża ilość tekstu do wysłania do wyświetlacza czy przez port RS232. Jest wiele sposobów na poradzenie sobie ze zbyt małą ilością pamięci. Część danych można zapisać w pamięci EEPROM. Można również ciągi tekstów przechowywać w pamięci Flash mikrokontrolera co można zrobić z wykorzystaniem funkcji *PROGMEM*: *prog_char tekst[] PROGMEM = {„ Arduino – Elektronika Praktyczna „};*.

Wykorzystanie pamięci EEPROM – sposobu zapisu i odczytu danych zostanie pokazane w dalszej części kursu w przykładach praktycznych. Do obsługi pamięci EEPROM mikrokontrolera przewidziane są funkcje znajdujące się w dodatkowej bibliotece *EEPROM*.

Definiowanie zmiennych

Zmienna jest zarezerwowanym miejscem do przechowywania danych. Składa się ona z nazwy, typu oraz wartości. Na przykład instrukcja *Int PinLED = 13;* tworzy zmienną nazwaną *PinLED* typu *int* i wartości początkowej *13*, która może być używana do wskazywania pinu 13, do którego dołączono diodę LED. Za każdym odwołaniem się do nazwy *PinLED* będzie wskazywana wartość 13, która w tym przypadku jest numerem pinu portu mikrokontrolera. Zdefiniowana zmienną można szybko użyć w dowolnych funkcjach np. *pinMode (PinLED, OUTPUT);*

Za pomocą tej funkcji linia *PinLED* o wartości 13 (13 linia mikrokontrolera) zostaje

skonfigurowana jako wyjcie. Zaletą zmiennej w tym przypadku jest to, że wystarczy określić wartość pinu raz a używać wiele razy. Więc jeśli później zdecydujemy się na zmianę z pinu 13 na pin 12, wystarczy zmienić numer pinu w jednym miejscu w kodzie programu. Zmienna ma inne zalety w postaci możliwości przechowywania wartości liczbowej. Co najważniejsze, można zmienić wartości zmiennej za pomocą prostej komendy (wskazane przez znak równości). Na przykład komenda *PinLED = 12;* zmienia wartość zmiennej na wartość 12. Zauważyć można że nie jest już potrzebne określenie typu zmiennej. Wystarczy tylko raz wskazać jej typ. Oznacza to, że nazwa zmiennej jest na stałe związane z rodzajem, tylko jego wartość się zmienia. Przed przypisaniem wartości do zmiennej zawsze w pierwszej kolejności należy ją zdefiniować. W definiowaniu zmiennych ważna jest deklaracja odpowiedniego jej typu. W **tabeli 3** wymieniono typy zmiennych oraz zakresy ich wartości. Ich zastosowanie będzie zależne od typu obliczeń jakie będą przeprowadzane w programie. Zmienne domyślnie są przechowywane w pamięci SRAM mikrokontrolera. Jak w języku C, zmienne mogą być inicjowane:

```
Char znak;
Int wartosc = 33;
```

Pierwsza deklaracja deklaruje zmienną bez wartości początkowej, natomiast drugiej zmiennej *wartosc* jest nadawana wartość początkowa 33. W zmiennych ważny jest również zakres jej działania. Zależy on od miejsca deklaracji zmiennej. Zmienne definiowane przed strukturami *setup()* oraz *loop()* będą zmiennymi globalnymi i ich zakres działania będzie w całym przygotowanym programie. Zmienne definiowane w funkcjach lub w strukturach *setup()* czy *loop()* będą działały tylko w nich:

```
void setup ()
{
  Int PinLED = 13;
  pinMode (pin, OUTPUT);
  digitalWrite (pin, HIGH);
}
```

W tym przypadku wartość *PinLED* zmieniać się może tylko wewnątrz struktury *setup()*. Jeśli zmienna jest globalna, jej wartość można zmienić w dowolnym miejscu w kodzie programu, co oznacza, że trzeba zrozumieć cały program aby wiedzieć co się stanie. Jeśli zmienna ma ograniczony zakres, działanie programu jest łatwiej zrozumieć.

Tworzenie funkcji

Funkcje czyli swego rodzaju procedury pozwalają programiście na dzielenie programu na moduły dzięki czemu jest bardziej zrozumiały oraz dane moduły (funkcje) mogą być wykonywane wielokrotnie bez potrzeby powtarzania kodu programu. Funkcje mogą wykonywać określone zadanie wielokrotnie np. funkcja opóźnienia która może być wykorzystana w programie wielokrotnie. Wywołanie funkcji powoduje wykonanie zawartego w niej programu i powrót po jego wykonaniu do programu głównego. Funkcje mogą posiadać parametry wejściowe jak np. w przypadku funkcji opóźnienia może to być czas opóźnienia. Mogą również one zwracać wynik obliczeń. Jak wspomniano zalety funkcji uwidaczniają się gdy trzeba coś w programie wielokrotnie powtórzyć. W programie bardzo często będą wykorzystywane funkcje czy to własne czy z wykorzystywanych bibliotek. Funkcja ma swoją nazwę oraz w nawiasie mogą się znajdować jej argumenty. Funkcje należy w pierwszej kolejności zdefiniować. W tym celu podaje się jej argumenty (identyczne jak typy zmiennych) oraz typ wartości zwracanej przez funkcje. W przypadku, gdy funkcja nie będzie zwracała żadnych wartości lub nie będzie miała żadnych wartości wejściowych wykorzystuje się do tego zaznaczenia słowo *void*: *void Delay_100ms(void);*. Funkcja ta będzie powodować opóźnienie programu o 100 ms.

Niżej umieszczono przykładową funkcję do mnożenia dwóch liczb:

```
Int Mnozenie(int x, int y){
Int wynik;
```

```
Wynik = x * y;
Return wynik;
}
```

W powyższym przypadku deklarowana jest funkcja mnożenia o nazwie *Mnozenie* która ma dwa argumenty typu *int*. Funkcja zwraca wartość typu *int* (iloczyn). Rezultat działania funkcji jest zapisywany do zmiennej lokalnej *wynik*. Komenda *return* umożliwia zwrócenie wartości obliczeń przez funkcję. Użycie funkcji może być następujące:

```
void loop{
int i = 2;
int j = 3;
int k;
k = Mnozenie(i, j); // wynik mnożenia to 6
}
```

W przykładzie zadeklarowano dwie zmienne *i* i *j* o wartość 2 i 3 oraz zmienną na *k* na ich iloczyn. Wywołanie funkcji mnożenie z parametrami *i* i *j* spowoduje wykonanie funkcji i wykonanie mnożenia dwóch wartości zapisanych do zmiennych *i* i *j* co da wynik 6 i jego zapis do zmiennej *k*. Dzięki przykładowej funkcji w każdej chwili w programie gdy będzie potrzebne mnożenie dwóch liczb wystarczy wywołać funkcje mnożenie podając jako jej parametry mnożone liczby.

Przykładowy program

W ramach podsumowania części teoretycznej na **listingu 2** pokazano prosty program powodujący pulsowanie diody LED.

W strukturze *setup()* jest konfigurowana linia 13 mikrokontrolera jako wyjście. Do tego wyjścia dołączona jest dioda LED. W strukturze *loop()* wykonywane są w nieskończonej pętli instrukcje z których pierwsza powoduje ustawienie linii 13 w stan wysoki (wyłączenie diody LED). Kolejna funkcja *delay* z parametrom 1000 powoduje opóźnienie działania programu o 1 sekundę (1000 ms). Po opóźnieniu wykonywana jest instrukcja ustawiająca stan niski na linii 13 po czym następuje wykonanie kolejnej funkcji opóźnienia o 1 sekundę. Po tej instrukcji działanie programu rozpoczyna się od początku co powoduje miganie diody dołączonej do pinu 13 mikrokontrolera. Z praktycznym działaniem tego programu będzie się można zapoznać w kolejnej części kursu.

Podsumowanie

W pierwszej części kursu Arduino opisano podstawowe funkcje i składnię języka Arduino. Są to informacje niezbędne do podjęcia programowania z tym systemie. W następnych częściach kursu zostanie pokazane środowisko programistyczne Arduino IDE wraz z instalacją zestawu Arduino UNO i jego uruchomieniem.

Marcin Wiązania
marcin.wiazania@ep.com.pl

Tabela 2. Biblioteki w Arduino

Biblioteki standardowe	
EEPROM	odczyt zapis do pamięci EEPROM
Ethernet	biblioteka funkcji sieciowych ETHERNET z wykorzystaniem modułu Arduino Ethernet Shield
Firmata	biblioteka komunikacji z komputerem z wykorzystaniem RS232
LiquidCrystal	biblioteka obsługi wyświetlaczy LCD
SD	biblioteka obsługi kart pamięci SD
Servo	biblioteka obsługi napędów servo
SPI	biblioteka obsługi interfejsu SPI (Serial Peripheral Interface)
SoftwareSerial	biblioteka obsługi programowej interfejsu komunikacyjnego RS232
Stepper	biblioteka obsługi silników krokowych
Wirebi	biblioteka obsługi interfejsu TWI/I^{2}C (Two Wire Interface)
Biblioteki komunikacyjne:	
Messenger	biblioteka do przetwarzania wiadomości tekstowych z komputera
NewSoftSerial	ulepszona biblioteka do obsługi programowej transmisji RS232
OneWire	biblioteka obsługi interfejsu 1Wire
PS2Keyboard	biblioteka obsługi klawiatury z interfejsem PS2
Simple Message System	biblioteka umożliwia wysyłanie wiadomości pomiędzy komputerem a Arduino
SSerial2Mobile	umożliwia wysyłanie wiadomości tekstowych lub mail za pomocą telefonu komórkowego (za pomocą poleceń AT)
Webduino	biblioteka serwera WWW z wykorzystaniem Arduino Ethernet Shield
X10	biblioteka umożliwia transmisje po liniach zasilających
XBee	umożliwia komunikacje z API XBee
SerialControl	umożliwia zdalną kontrolę innych Arduino za pomocą interfejsu RS232
Biblioteki do obsługi czujników:	
Capacitive Sensing	biblioteka dla czujników pojemnościowych
Debounce	biblioteka do obsługi przycisków
Obsługa wyświetlaczy i matryc LED:	
Improved LCD library	biblioteka obsługi wyświetlaczy LCD
GLCD	biblioteka obsługi graficznych LCD z kontrolerem KS0108
LedControl	biblioteka sterująca 7-segmentowymi wyświetlaczami LED oraz LED'ami z kontrolerami MAX7221 lub MAX7219
LedDisplay	biblioteka obsługi wyświetlaczy z kontrolerem HCMS-29xx
Generatory:	
Tone	biblioteka umożliwia generowanie dźwięku na dowolnym pinie mikrokontrolera
TLC5940	Umożliwia obsługę 16 kanałowego i 12 bitowego kontrolera PWM
Data i godzina:	
DateTime	biblioteka realizująca zegar i kalendarz
Metro	biblioteka umożliwiające odmierzanie stałych odcinków czasu
MsTimer2	biblioteka generująca przerwanie co czas odmierzony w milisekundach
Tekstowe:	
TextString	biblioteka obsługi tekstów
PString	biblioteka zapisu tekstu do bufora
Streaming	uproszona biblioteka funkcji print()

Listing 2. Przykładowy program napisany dla Arduino

```
void setup() {
  pinMode(13, OUTPUT);  //konfiguracja linii 13 jako wyjście
}

void loop() {
  digitalWrite(13, HIGH);   // wyłączenie diody LED
  delay(1000);              // opóźnienie 1 sekundy
  digitalWrite(13, LOW);    // włączenie diody LED
  delay(1000);              // opóźnienie 1 sekundy
}
```

Tabela 3. Zakresy typów zmiennych

Typ	Zakres
boolean	True, False
char	-128 do 127
unsigned char	0 do 255
byte	0 do 255
int	-32768 do 32767
unsigned int	0 do 65,535
word	0 do 65535
long	-2147483648 do 2147483647
unsigned long	0 do 4294967295
float	3,4028235E+38 do -3,4028235E+38
double	(wartość 4-bajtowa)
string	ciąg znaków