Wyższa Szkoła Handlowa w Radomiu

Systemy Wbudowane

Laboratorium 1

1 Cel zajęć

Celem ćwiczenia jest stworzenie prostej aplikacji pozwalającej na sterowanie diodami LED. Sterowanie odbywać się będzie poprzez oprogramowane odpowiednio przyciski.

2 Wprowadzenie

Niniejsze ćwiczenie stanowi wstęp do świata programowania układów wbudowanych. Świat układów jest bogaty, nie ogranicza się do jednego modelu mikropocesora czy architektury. Specjaliści z tej dziedziny muszą wykazać się nie tylko umiejętnościami programowania, lecz także podstawowej wiedzy dotyczącej funkcjonowania samych układów elektronicznych oraz możliwości podłączania do nich dodatkowych modułów czy też budowania z nich dedykowanych systemów informatycznych, takich jak instalacje pomiarowe, transmisyjne czy badawcze.

2.1 Układ wbudowany

W ramach laboratoriów wykorzystywany będzie układ Arduino Leonardo. Układy z rodziny Arduino (oraz Arduinopodobnych) cechuje przede wszystkim prostota działania. Układy zaprojektowane zostały by w prosty i przystępny sposób pasjonaci budowy systemów programowalnych mogli osiągnąć swoje cele – tworzyć proste roboty, systemy sterowania, systemy pomiarowe itp. W przeciwieństwie do bardziej rozbudowanych układów płytki Arduino posiadają przyłącze programistyczne (zwane najczęściej Debug Adapter – przyłącze odpluskwiania) oraz program rozruchowy dzięki czemu nie jest konieczny zakup dodatkowego osprzętu (który nierzadko kosztuje kilkadziesiąt/kilkaset złotych). Same układy nie są specjalnie skomplikowane. Posiadany na potrzeby zajęć laboratoryjnych układ Arduino Leonardo wyposażony jest w procesor Atmel z rodziny AVR - ATmegaXU4, należący do rodziny ATMega32U4, jednak posiadający dodatkowo zintegrowany kontroler USB. Płytka posiada 18 wyprowadzeń (PINs) cyfrowych: D0-D13 zgodne z innym układem Arduino – Uno. Dodatkowe trzy wyprowadzenia (D14-D16) znajdują się przy wyjściu ICSP (In-Cirtuit Serial Programming) - 1, 3 oraz 4 igła (PIN). D17 natomiast nie posiada bezpośredniego wyprowadzenia – znajduje się ono przy diodzie RX. Aby je obsużyć należy bądź to wykorzystać odpowiednie przyłącze na płytce drukowanej, bądź przyłączyć się bezpośrednio do nogi diody RX. Wyprowadzenia D0-D13 ponadto pełnią rolę wejść/wyjść PWM (Pulse Width Modulation), co pozwala na sterowanie szerokością impulsu cyfrowego (szczególnie efektowne w przypadku diod LED, efektywne w przypadku wykorzystania kontroli przesyłania danych). Arduino Leonardo posiada wyprowadzenia analogowe. Tego typu wejścia/wyjścia mogą zostać zastosowane do obsługi potencjometrów, konwertera AD oraz DA, pomiarów napięciowych czy też prostego systemu dźwiękowego. Wykorzystywana płytka posiada 12 wyprowadzeń analogowych – A0-A11. Wyprowadzenia A6-A11 mieszczą się pod wyprowadzeniami cyfrowymi, odpowiednio D4, D6, D8, D9, D10, oraz D12.

INFORMACJA: Wyprowadzenia A0-A5 stanowią jednocześnie wyprowadzenia cyfrowe, odpowiednio D18-D23.

Ponadto płytka posiada wyprowadzenia magistrali I2C/TWI, szyny RX/TX, portu szeregowego, przycisku Reset (opcja wyprowadzenia na zewnętrzny przycisk) oraz wyprowadzenia wspomnianego wyżej kontrolera USB (możliwość podłączenia jako modułu).

2.2 Środowisko programistyczne oraz język programowania

Tak jak w przypadku pozostałych układów i rozwiązań, tak i w przypadku Arduino kod naszego programu możemy tworzyć w dowolnym edytorze tekstowym. Za kompilację oraz przeniesienie kodu do układu odpowiadają narzędzia avr-gcc (dostępne na większości systemów operacyjnych), z czego avr-gcc kompiluje kod, a avrdude wgrywa kod do urządzenia.

Ponieważ jednak główny nacisk zajęć kierujemy na tworzenie kodu oraz rozbudowę układu laboratoryjnego, na

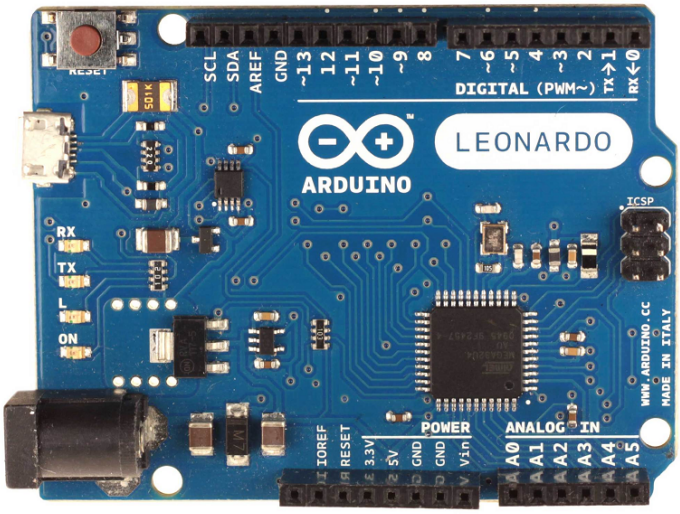

Figure 1: Widok płytki Arduino Leonardo.

zajęciach wykorzystywać będziemy środowisko Arduino IDE (obecnie wersja 1.8.1). Arduino IDE pisane jest w języku Java dlatego jego wersja dostępna jest na wszystkie systemy operacyjne. Interfejs posiada możliwość wyboru wersji językowej (w tym języka polskiego).

Generalnie przyjęło się, że kod pisany jest w języku C/C++. Oczywiście stwierdzenie to jest prawdziwe – kod musi być zgodny ze standardem C bądź C++ (w tym istnieje możliwość wykorzystywania C++11/C++14 – w zależności od wersji wykorzystywanego kompilatora avr). Należy mieć jednak na uwadze, że sam język programowania wykorzystywany w Arduino IDE to Processing. Sam język oraz jego IDE stworzony został na potrzeby sztuki (głównie grafiki i muzyki) i oryginalnie wykorzystuje on język Java. Celem stworzenia go było ułatwić programowanie osobom, które do tej pory nie potrafiły programować bądź programowanie większych aplikacji sprawiało im problemy (zwłaszcza nauka języka Java). Każdy program tworzony jest jako szkic (sketch) i posiada przeważnie dwie funkcje:

setup – istnieje w każdym projekcie; odpowiada za ustawienie zmiennych, przygotowanie struktur oraz obiektów wykorzystywanych w głównej funkcji; kod tej funkcji wykonywany jest zawsze PRZED głównym programem i zawsze JEDEN RAZ (jej kod nigdy nie wykona się dwa razy).

draw/play/loop – nazwa drugiej funkcji zależna jest od rodzaju szkicu. Generalnie przyjęło się, że jej nazwa ma odpowiadać zadaniu, które wykonuje. W przypadku Arduino wykorzystywana jest funkcja loop, będąca wieczną pętlą programu (odpowiednik `while(1)` bądź `for (;;)`).

Prócz samego języka Processing Arduino IDE wykorzystuje również bibliotekę Wiring (biblioteka dla języka C/C++). Dzięki niej zapis i odczyt z poszczególnych wyprowadzeń jest znacznie prostszy. Przykład kodu wysyłającego impuls w C/C++ z wykorzystaniem standardowej biblioteki wejścia/wyjścia:

```
#include <avr/io.h>

DDRB |= _BV(PB5);
PORTB |= _BV(PB5);
```

Przykład kodu wykorzystującego bibliotekę Wiring:

```
pinMode(13, OUTPUT);
digitalWrite(13, HIGH);
```

INFORMACJA Biblioteka Wiring dostosowywana jest do aktualnie wybranego modelu płytki Arduino. Dlatego ważne jest przed rozpoczęciem kodowania by wybrać odpowiedni model z Narzędzia.

Na koniec warto wspomnieć, że w tworzonych projektach można wykorzystywać mnemoniki Assemblera dla procesora AVR. Odwołania do niego tworzy się jako fragment kodu C++ w funkcji

```
asm volatile ("mnemonik operacja"
"mnemonik operacja:"
);
```

Dla systemów wbudowanych należy pamiętać o użyciu słowa `volatile` – dzięki niemu (w ogólnym skrócie) komendy napisane w assemblerze **NIE BĘDĄ** optymalizowane przez kompilator (zakładamy, że pisząc własny kod niskopoziomowy sami dbamy o optymalizację).

W niniejszym ćwiczeniu wykorzystamy następujący układ testowy:

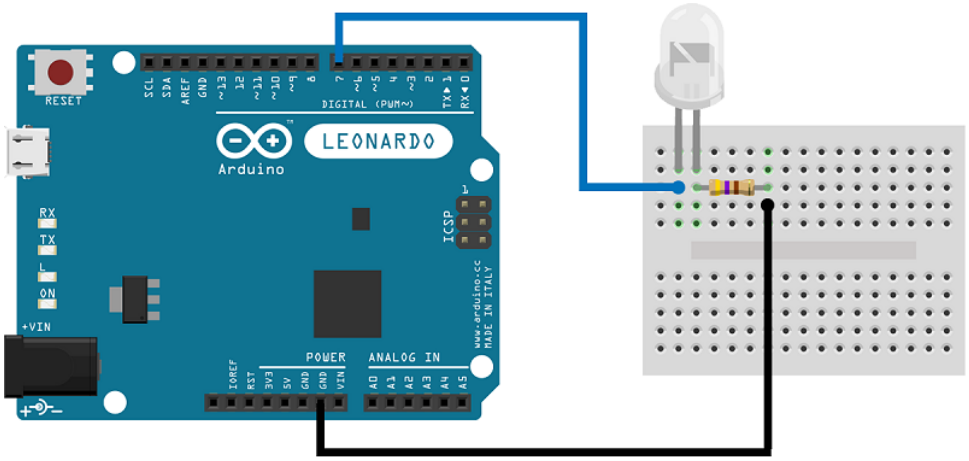

Figure 2: Przykład podłączenia LED do układu Arduino Leonardo.

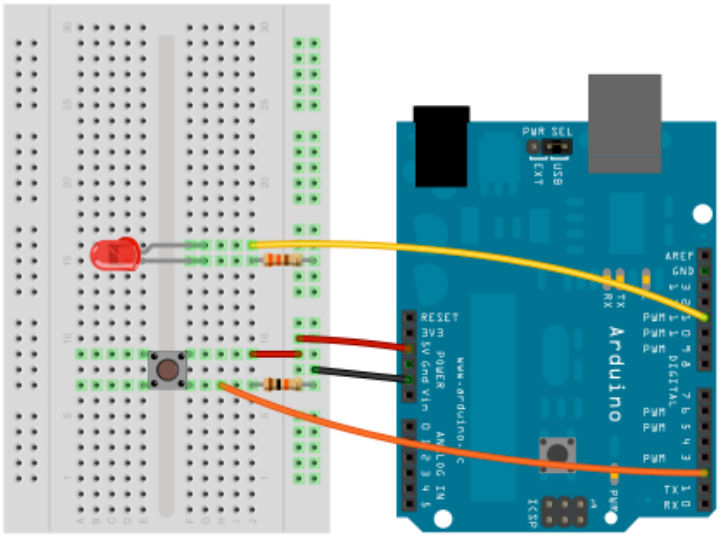

Figure 3: Przykład podłączenia LED oraz przycisku do badanego układu.

Jak można wywnioskować, w celu wykonania ćwiczenia niezbędne będą nam diody świecące, przyciski, rezystory (220-230 ohm) oraz płytka połączeniowa. Proszę zauważyć, że płytka ta pozwala na tworzenie dowolnego układu wbudowanego bez konieczności lutowania elementów na stałe.

W przypadku doboru odpowiedniego rezystora do diody i/lub przycisku można posłużyć się kalkulatorem:

`http://serwis-tv.com/opornik.html`

`http://kalkulator.majsterkowicza.pl/oblicz/kod_paskowy_rezystora`

2.3 Przykładowy kod

```
1  typedef unsigned int uint
2  typedef const unsigned int cuint
3
4  cuint btn1 = 2, btn2 = 3;
5  cuint led1 = 10, led2 = 11;
6
7  uint btn1S, btn2S;
8
9  void setup()
10 {
11     pinMode(btn1, INPUT);
12     pinMode(btn2, INPUT);
13     pinMode(led1, OUTPUT);
14     pinMode(led2, OUTPUT);
15 }
16
17 void loop()
18 {
19     btn1S = digitalRead(btn1);
20     btn2S = digitalRead(btn2);
21     if (btn1S)
22     {
23         digitalWrite(led2, HIGH);
24     }
25     else {
26         digitalWrite(led2, LOW);
27     }
28     if (btn2S)
29     {
30         digitalWrite(led1, HIGH);
31     }
32     else
33     {
34         digitalWrite(led1, LOW);
35     }
36 }
```

3 Zadania do wykonania

1. Zastanowić się nad optymalizacją kodu (zmiana na stałe, eliminacja niepotrzebnych poleceń, czyszczenie kodu).

2. Dołożyć i oprogramować 3 diodę w taki sposób, by sygnalizowała naciśnięcie obu przycisków.

3. Dołożyć kod zabezpieczający przez tzw. iskrzeniem przycisków. Iskrzeniem nazywa się stan, w którym użytkownik nie dociśnie przycisku (np. dotknie delikatnie niewłaściwy przycisk), a płytka zareaguje na akcję.

4. Przebudować program tak, by użytkownik nie musiał stale wciskać przycisku dla określonego efektu świetlnego. Przykładowo po wciśnięciu przycisku 1 program ma pamiętać stan diody przypisany do wciśniętego

przycisku. Jeżeli przycisk zostanie ponownie naciśnięty stan diody ma się zmienić. Należy zastanowić się nad rozwiązaniem obu przycisków wciśniętych - w jaki sposób przechować stan dla diody 3.

5. Należy dołożyć jeszcze 2 diody. Diody powinny zapalać się i gasić jedna po drugiej (szeregowo), w określonym czacie (np. co 1,5 sekundy). W przypadku naciśnięcia przycisku 1 czas ten powinien się skracać o określony czas (np. 0,3 sekundy). Naciśnięcie 2 przycisku powinno natomiast powodować zwiększenia tego czasu. Naciśnięcie dwóch przycisków jednocześnie ma zmieniać kierunek szeregu.
DLA STUDENTÓW STUDIÓW NIESTACJONARNYCH:

6. Należy sporządzić sprawozdanie z przebiegu ćwiczenia według podanego szablonu (dostępny na stronie).

4 Materiały wykorzystane przy opracowaniu

http://www.microchip.com/wwwproducts/en/ATmega32u4

http://www.instructables.com/id/Step-by-Step-Guide-to-the-Arduino-Leonardo/

https://www.arduino.cc/en/Hacking/BuildProcess

https://processing.org

http://blog.nettigo.pl/post/85619491578/czemu-powsta\T1\lo-arduino

http://wiring.org.co

http://www.uni-koeln.de/phil-fak/muwi/ag/praktikum/assembler_arduino.pdf

http://blog.codebender.cc/wp-content/uploads/2014/03/blink_.png

http://breakoutjs.com/examples/getting_started/images/hello_world_schematic.png