Ćwiczenie 0: Wprowadzenie

Instrukcja laboratorium

Mariusz Chilmon <mariusz.chilmon@ctm.gdynia.pl>

2023-12-13

> When the terrain disagrees with the map, trust the terrain.
>
> — *Swiss army proverb*

Środowisko programistyczne

Płytka ewaluacyjna *Arduino Uno* z procesorem ATmega328P.

Kompilator AVR Toolchain v3.7 bazujący na kompilatorze GCC.

Programator AVRDUDE v7.1.

IDE Visual Studio Code (nie mylić z Visual Studio) z wtyczką *C/C++* umożliwiającą nawigowanie po kodzie C i C++ oraz automatyczne uzupełnianie kodu w tych językach.

Katalog roboczy `Embedded` w katalogu `Dokumenty`. Kody źródłowe ćwiczeń należy umieszczać w `Embedded/Code`. W `Embedded/Datasheets` umieszczone są noty katalogowe mikrokontrolerów oraz dokumentacje płytek ewaluacyjnych.

Pobieranie kodu i instrukcji do ćwiczenia

1. Wyczyść zawartość katalogu `Embedded/Code`.
2. Uruchom Visual Studio Code.
3. Wciśnij *Ctrl* + *Shift* + *P* i wpisz polecenie *git clone*.
4. Sklonuj repozytorium Git https://github.com/vmario/amw-embedded-lab-XX.git, gdzie *XX* to numer ćwiczenia. Nie wybieraj opcji *Clone from GitHub*, ale wpisz adres i wybierz *Clone from URL*.
5. Wybierz katalog `Embedded/Code` do zapisania projektu.

Kompilacja programu

1. Wciśnij *Ctrl* + *Shift* + *B* i wybierz zadanie *all*.

Wgrywanie wsadu

1. Wciśnij *Ctrl* + *Shift* + *B* i wybierz zadanie *program*.

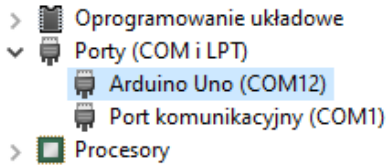
Zadanie *program* przebudowuje też program wynikowy, jeżeli zostały wprowadzone jakieś zmiany w plikach źródłowych. Zatem, jeżeli chcesz skompilować program i od razu go wgrać, możesz pominąć zadanie *all*.

Jeżeli programator zgłosi problem:

```
avrdude error: cannot open port \\.\COM2: Nie mozna odnalezc
    okreslonego pliku.

avrdude error: unable to open programmer arduino on port \\.\COM2
```

należy ustawić odpowiedni numer portu w zmiennej `port` w pliku `.vscode/settings.json`. Aby określić numer portu, należy uruchomić *Menedżer urządzeń*, rozwinąć listę *Porty (COM i LPT)*, a następnie podłączyć płytkę ewaluacyjną i zaobserwować nowy numer portu oznaczony nazwą *Arduino Uno* lub *Urządzenie szeregowe USB*.



Rysunek 1: Programator na liście urządzeń (tu: *COM12*)

Opis skonfigurowanych zadań

all Buduje program.

clean Czyści wszystkie pliki wynikowe (usuwa efekt budowania).

erase Czyści pamięć mikrokontrolera i wgrywa bootloader. Wymaga zewnętrznego programatora.

program Buduje program i wgrywa go do mikrokontrolera.

Ustawianie bitów w rejestrach

Aby ustawić bity, np. o numerze 2 i 3 w rejestrze `REG` można zapisać wprost wartość do rejestru:

```
REG = 0b00001100;
```

Powoduje to jednak wyzerowanie wszystkich bitów oprócz 2 i 3. Jeżeli chcemy zachować pozostałe bity niezmienione, należy posłużyć się operatorem przypisania złożonym z operatorem alternatywy bitowej *OR*:

```
REG |= 0b00001100;
```

Jeżeli chcemy wyzerować bity 2 i 3, pozostawiając pozostałe niezmienione, używamy operatora przypisania złożonego z operatorem koniunkcji bitowej *AND* i operatora negacji *NOT*:

```
REG &= ~0b00001100;
```

Binarne formaty zapisu liczb

Język C++14 wprowadził binarny zapis liczb w postaci `0b00001100`. We wcześniejszych wersjach tego języka i w języku C (za wyjątkiem rozszerzeń wprowadzanych przez pewne kompilatory) taki format nie jest dostępny i zazwyczaj stosowana jest notacja szesnastkowa postaci `0x0C`. Możliwe jest też składanie liczb za pomocą operatora przesunięcia bitowego i operatora alternatywy bitowej lub, w przypadku kompilatora AVR-GCC, makra `_BV()`. Poniższe zapisy są zatem równoważne:

```
REG = 0b00001100;
REG = 0x0C;
REG = (1 << 2) | (1 << 3)
REG = _BV(2) | _BV(3);
```

Użycie operatora przesunięcia bitowego lub makra `_BV()` umożliwia ustawianie bitu wskazanego przez zmienną, np.: `REG |= _BV(bitNumber)`, gdzie zmienna `bitNumber` przechowuje numer bitu, który chcemy ustawić.