Arduino dla mikrokontrolerów STM32 (2)

W pierwszym artykule z tego cyklu opublikowanym w EP 10/2018 pokazano, jak skonfigurować środowisko programistyczne Arduino do pracy z mikrokontrolerami STM32. Ta część kontynuuje tematykę Arduino i STM32 prezentując jak zacząć pisanie kodu i jak sterować z poziomu kodu podstawowym zasobem mikrokontrolera – portami wejścia/wyjścia.

Pracę należy rozpocząć uruchamiając środowisko programistyczne Arduino. Przygotowanie do pisania kodu obejmuje kilka łatwych do wykonania kroków.

W pierwszej kolejności konieczne jest stworzenie nowego projektu (zwanego sketchem w Arduino). W tym celu użytkownik wybiera z menu głównego środowiska programistycznego kolejno *File* i *New* (skrót klawiaturowy CTRL+N).

W następnym kroku należy wskazać używaną platformę sprzętową. Przykładowo, autor wybrał płytkę NUCLEO-L476RG z 64-pinowym układem STM32L476RG. W tym celu należy ponownie użyć menu głównego środowiska programistycznego. Wybór platformy przebiega w dwóch etapach. Najpierw klikając *Tools* i *Board* wybrać należy *Nucleo-64*. Następnie wybierając *Tools* i *Board Part Number* wybrać należy *Nucleo L476*.

Listing 1. Kod nowego projektu w środowisku programistycznym Arduino

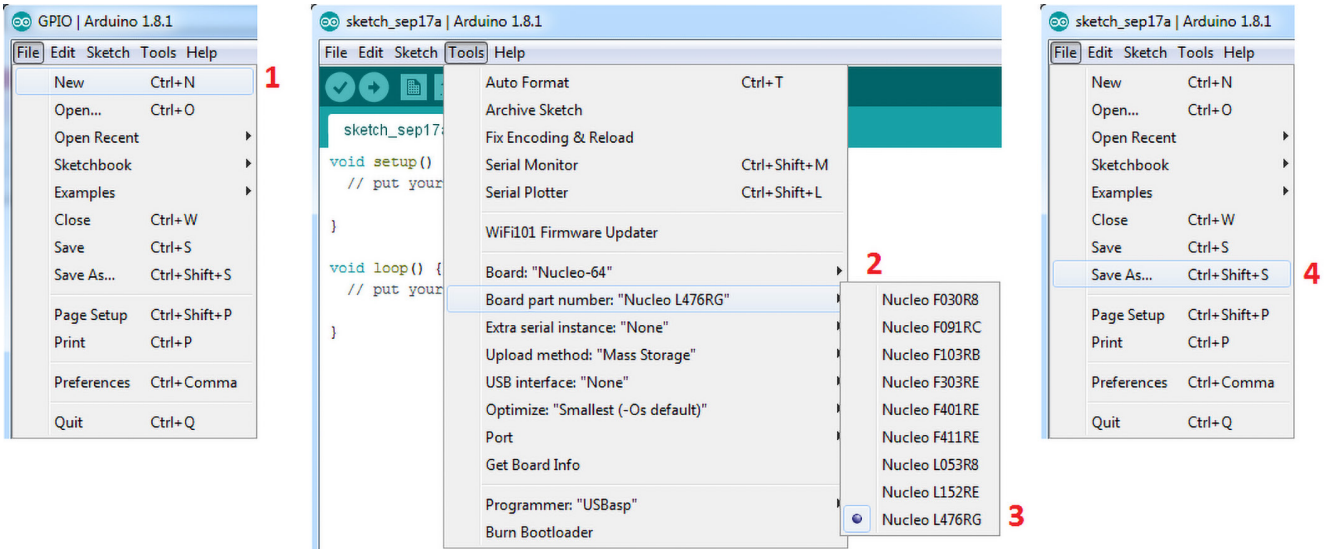
```
void setup()
{
    // put your setup code here, to run once:
}

void loop()
{
    // put your main code here, to run repeatedly:
}
```

W ostatnim kroku należy zapisać projekt na dysku twardym. Jednocześnie warto zmienić nazwę projektu na bardziej intuicyjną, gdyż domyślna nazwa zawiera tylko słowo sketch oraz nazwę aktualnego miesiąca i dnia. Chcąc zapisać projekt oraz zmienić jego *nazwę* użytkownik znowu korzysta z menu głównego środowiska programistycznego wybierając kolejno *File* i *Save as…* (skrót klawiaturowy CTRL+Shift+S). W ten sposób otworzone zostanie okno, które pozwoli na wybranie ścieżki na dysku twardym, pod którą zapisany zostanie projekt. Jednocześnie okno pozwala na wpisanie nowej nazwy projektu.

Arduino i STM32 – struktura kodu

Każdy stworzony od nowa projekt (sketch) zawiera domyślnie kilka linii kodu tworzących szkielet aplikacji. Kod ten pokazano w **listingu 1**. Składa się on z dwóch funkcji. Pierwsza funkcja nosi nazwę *setup*. Jej przeznaczenie można łatwo rozszyfrować zapoznając się z komentarzem.



Rysunek 1. Kolejne kroki przygotowania środowiska Arduino do pisania kodu dla wybranej platformy z układem STM32

Listing 2. Przykładowa aplikacja kalkulatora

```
int zmienna1, zmienna2, wynik;

int dodawanie(int a, int b)
{
    return a + b;
}

void setup()
{
    // put your setup code here, to run once:
    zmienna1 = 2;
    zmienna2 = 1;
}

void loop()
{
    // put your main code here, to run repeatedly:
    wynik = dodawanie (zmienna1, zmienna2);
    zmienna1 = zmienna1 + 1;
    zmienna2 = zmienna2 + 1;
    if (zmienna1 == 5)
    {
        zmienna2 = 3;
    }
}
```

Programista umieszcza w niej kod, który ma zostać wykonany tylko raz, na początku aplikacji, a więc po doprowadzeniu napięcia zasilania do mikrokontrolera. Są to głównie funkcje konfigurujące peryferia mikrokontrolera (np. ustawiające parametry przetwornika A/C lub interfejsu UART) lub inicjujące podzespoły zewnętrzne (np. moduł Bluetooth). Druga funkcja otrzymała nazwę *loop*. Ponownie w rozszyfrowaniu jej przeznaczenia pomaga komentarz. Jest to pętla nieskończona, w której programista umieszcza funkcje wykonywane cyklicznie (np. odczyt stanu przycisku lub odebranie danych z czujnika).

Zatem programista tworzy kod aplikacji uzupełniając ciała funkcji *setup* i *loop* o dodatkowy kod języka Arduino. Kod ten składa się z typowych elementów języka programowania takich jak zmienne, funkcje, instrukcje warunkowe czy też operacje matematyczne. Przyjrzyjmy się sposobie ich użycia na przykładzie prostej aplikacji kalkulatora realizującego dodawanie dwóch liczb. Kod (pokazany na **listingu 2**) zawiera kolejno:

- deklarację trzech zmiennych: *zmienna1*, *zmienna2* i *wynik*,
- definicję funkcji *dodawanie* realizujących działanie dodawania,
- wewnątrz funkcji *setup*:
- inicjalizację zmiennych *zmienna1* i *zmienna2* wartościami początkowymi,
- wewnątrz funkcji *loop*:
- wywołanie funkcji *dodawanie* i przypisanie zwracanej wartości do zmiennej *wynik*,
- inkrementację zmiennych *zmienna1* i *zmienna2*.
- pętlę warunkową *if* sprawdzającą czy wartość zmiennej *zmienna1* wynosi 5, a jeśli jest to prawda, zmieniającą wartość zmiennej *zmienna2* na 3.

Porty wejścia/wyjścia – podstawowe informacje

Mając skonfigurowane środowisko programistyczne Arduino oraz znając podstawy programowania Arduino można przejść do kolejnego poziomu wtajemniczenia, jakim jest umiejętność sterowania portami wejścia/wyjścia.

W każdym systemie elektronicznym mikrokontroler poprzez swoje wyprowadzenia, a dalej ścieżki na płytce PCB dołączony jest do pewnej liczby podzespołów, z którymi musi się komunikować. Komunikację należy tu rozumieć jako umiejętność wysyłania i odbierania informacji w postaci sygnału elektrycznego (napięciowego). Zadanie to realizują porty wejścia/wyjścia.

Porty mikrokontrolera mogą pracować w dwóch trybach. Pierwszy z nich to GPIO (*General Purpose Input Output*). W trybie tym programista ma bezpośrednią kontrolę nad portami i może sterować nimi z poziomu aplikacji. Porty w trybie GPIO wykorzystywane są przede wszystkim do prostych operacji wejścia wyjścia np. odczytywania stanów przycisków lub sterowania stanem (włącz/wyłącz) przekaźników, kluczy tranzystorowych, diod LED itp.

Listing 3. Kod programu odczytującego stan przycisku i zmieniającego stan diody LED

```
void setup()
{
    // put your setup code here, to run once:
    pinMode(PA5, OUTPUT);
    pinMode(PC13, INPUT);
}

void loop()
{
    // put your main code here, to run repeatedly:
    if (digitalRead(PC13) == 0)
    {
        digitalWrite(PA5, HIGH);
        delay(500);
        digitalWrite(PA5, LOW);
        delay(500);
    }
}
```

Drugi tryb pracy portów wejścia/wyjścia nazywany jest trybem alternatywnym. W trybie ty porty są połączone z peryferiami mikrokontrolera, aby umożliwić ich komunikowanie się ze światem zewnętrznym. Przykładowo porty mogą być wykorzystane przez przetwornik A/C do odczytywania wartości napięcia z wyprowadzenia mikrokontrolera, przez timer do generowania sygnału PWM poza układ lub przez interfejsy komunikacyjne do realizacji transmisji danych między mikrokontrolerem i innymi układami.

Porty wejścia/wyjścia – funkcje Arduino

W artykule tym spośród dwóch wymienionych trybów portów wejścia/wyjścia przedstawiony zostanie jeden, podstawowy – GPIO. Do tego celu środowisko Arduino udostępnia trzy funkcje: *pinMode*, *digitalWrite* oraz *digitalRead*.

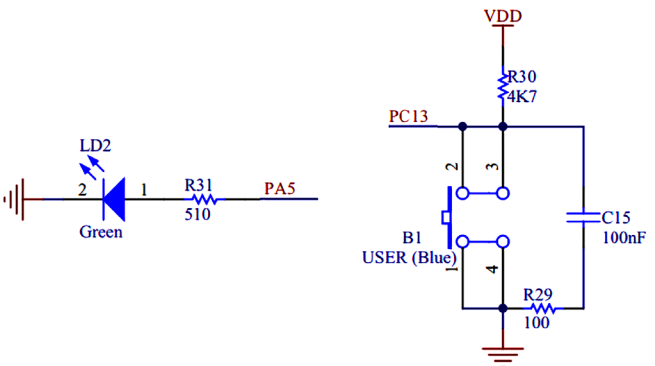
Funkcja *pinMode* odpowiada za skonfigurowanie wybranego przez programistę portu. Funkcja ta przyjmuje dwa argumenty. Pierwszym argumentem jest oznaczenie portu np. PA0, PB4 itp. Drugi argument określa konfigurację. Dostępne są trzy warianty: INPUT (wejście), OUTPUT (wyjście), INPUT_PULLUP (wyjście z aktywnym rezystorem połączonym z dodatnim potencjałem napięcia zasilania).

Funkcja *digitalWrite* pozwala na ustawianie wartości logicznej na wyjściu portu. Analogicznie do poprzedniego przypadku funkcja ta również przyjmuje dwa argumenty. Pierwszym argumentem jest oznaczenie portu. Drugi argument to poziom napięcia odpowiadający wartości logicznej: HIGH (poziom wysoki, wartość 1), LOW (poziom niski, wartość 0).

Funkcja *digitalRead* służy do odczytywania wartości logicznej z wejściu portu. Jedynym argumentem funkcji jest nazwa portu. Wartość zwracana przez funkcję informuje o poziomie napięcia: HIGH (poziom wysoki, wartość 1), LOW (poziom niski, wartość 0).

Porty wejścia/wyjścia – przykładowa aplikacja

Płytka NUCLEO-L476RG dysponuje dwoma podzespołami podłączonymi do portów wejścia/wyjścia mikrokontrolera. Jest to dioda LED oraz przycisk. Odpowiedni schemat pokazano na **rysunku 2**.



Rysunek 2. Schemat elektryczny diody LED i przycisku podłączonych do portów wejścia/wyjścia mikrokontrolera na płytce NUCLEO-L476RG

Dioda LED połączona jest jednym pinem do ujemnego potencjału zasilania oraz drugim pinem przez rezystor do portu PA5. Łatwo zatem stwierdzić, że poziom wysoki na porcie spowoduje przepływ prądu przez diodę LED i wywoła efekt jej świecenia. Z kolei poziom niski na porcie wstrzyma przepływ prądu i efekt świecenia nie wystąpi.

Jeden styk przycisku połączony jest do ujemnego potencjału zasilania, natomiast drugi ze styków połączony jest przez rezystor do dodatniego potencjału napięcia zasilania i jednocześnie bezpośrednio do portu PC13. W takiej konfiguracji w warunkach nienaciśniętego przycisku (styki rozwarte) na linii PC13 ustabilizuje się stan wysoki. Z kolei po wciśnięciu przycisku (styki zwarte) stan zmieni się z wysokiego na niski.

Po zapoznaniu się najpierw z funkcjami Arduino do obsługi portów wejścia/wyjścia, a następnie po przeanalizowaniu schematu dołączonych do mikrokontrolera podzespołów, jesteśmy gotowi do napisania programu odczytującego stan przycisku i zmieniającego stan diody LED. Zaprezentowany na listingu 2 kod zawiera kolejno:

- wewnątrz funkcji *setup*:
- wywołanie funkcji *pinMode* konfigurującej port PA5 jako wyjście,
- wywołanie funkcji *pinMode* konfigurującej port PC13 jako wejście,
- wewnątrz funkcji *loop*:
- użycie instrukcji warunkowej, która poprzez wywołanie funkcji *digitalRead* sprawdza poziom napięcia na wejściu portu PC13,
- w wypadku poziomu niskiego na wejściu portu PC13:
- wywołanie funkcji *digitalWrite* ustawiającej stan wysoki na porcie PA5,
- wywołanie funkcji opóźniającej *delay* 500ms,
- wywołanie funkcji *digitalWrite* ustawiającej stan niski na porcie PA5,
- ponowne wywołanie funkcji opóźniającej *delay* (500 ms).

W efekcie działania tej aplikacji każde wciśnięcie przycisku wywoła zaświecenie diody LED na czas 500 ms.

Po napisaniu kodu program należy skompilować wybierając przycisk *Verify* (skrót klawiaturowy CTRL+R). Po zakończeniu procesu kompilacji należy wgrać program do pamięci mikrokontrolera wybierając przycisk *Upload* (skrót klawiaturowy CTRL+U).

Podsumowanie

Drugi artykuł z cyklu traktującego o Arduino i STM32 przechodzi od teorii do praktyki. W materiale zaprezentowano podstawy programowania oraz sposób sterowania portami wejścia/wyjścia mikrokontrolera.

Kolejne artykuły rozwijać będą aspekt prostych aplikacji korzystających z zasobów mikrokontrolera. Tematyka obejmować będzie przetwornik A/C (aplikacja: odczytywanie napięcia na pinie mikrokontrolera), interfejs UART (aplikacja: komunikacja szeregowa z komputerem) oraz Timer (wytwarzanie na pinie sygnału PWM).

Szymon Panecki
STMicroelectronics
szymon.panecki@st.com