

Graficzne programowanie mikrokontrolerów

M5Stack / UIFlow

Laboratorium 1

Wprowadzenie do M5Stack i UIFlow

Czas trwania: 2 × 45 min**Tryb:** studia stacjonarne**Jednostka:** Katedra Elektrotechniki i Technologii Nadprzewodnikowych**Prowadzący:** dr inż. Michał Łanczont

Cel ćwiczenia

Celem ćwiczenia jest zapoznanie studenta z platformą M5Stack, środowiskiem UIFlow oraz podstawowym sposobem tworzenia programów w języku graficznym Blockly. Student poznaje strukturę programu, zmienne, instrukcje warunkowe, pętle, funkcje oraz podstawy programowania zdarzeniowego.

Wstęp

Nazewnictwo w programie

W instrukcji przyjęto polskie nazwy zmiennych, funkcji tworzonych przez studenta, stanów programu i komunikatów wyświetlanych na ekranie. Nazwy wymagane przez bibliotekę, standard lub interfejs sprzętowy (np. UART, RS485, I2C, Modbus RTU, UID, nazwy klas i bloków bibliotecznych) pozostają zgodne z dokumentacją techniczną.

Platforma M5Stack

M5Stack jest modułową platformą systemów wbudowanych opartą na mikrokontrolerach z rodziny ESP32. Typowy moduł Core zawiera m.in. ekran LCD, przyciski użytkownika, interfejsy komunikacyjne oraz wyprowadzenia umożliwiające dołączenie modułów zewnętrznych.

W laboratorium stosowane będą dwa urządzenia:

- **M5Stack Core + Demo Board** – główne stanowisko do obsługi wejść, wyjść, czujników, napędów i komunikacji,
- **M5Stack Core2** – dodatkowe urządzenie wykorzystywane m.in. do budowy GUI dotykowego i komunikacji pomiędzy węzłami.

UIFlow i Blockly

UIFlow umożliwia tworzenie programu z bloków reprezentujących instrukcje, operacje na zmiennych, funkcje oraz obsługę sprzętu.

Typowy program można rozpatrywać jako trzy warstwy:

1. **inicjalizacja** – ustawienie początkowego stanu urządzenia - grupa bloku **Setup**,
2. **logika programu** – warunki, pętle i funkcje - grupa bloku **Loop**,
3. **obsługa zdarzeń** – reakcja na przyciski, dotyk i inne wejścia.

Dostępne są dwie gałęzie rozwojowe środowiska programowania graficznego UIFlow, gałąź 1.x i 2.x. Generacja pierwsza jest już praktycznie nierozwijana, a jedynie aktualizowana o biblioteki dla nowych komponentów wprowadzanych na rynek. Zalecane jest stosowanie gałęzi drugiej, aktualnej i rozwijanej. Już tylko z jej poziomu możliwe jest programowanie nowych modeli mikrokontrolerów produkowanych przez M5Stack.

Omówienie struktury interfejsu środowiska programistycznego zostanie przeprowadzona na wykładzie, a najważniejsze jego elementy omówione przez prowadzącego ćwiczenia laboratoryjne.

Uwaga

Przed rozpoczęciem zajęć należy upewnić się, że wersja zainstalowanego oprogramowania układowego UIFlow jest **aktualna**. W razie konieczności konieczne jest aktualizacja w programie **M5Burner**. Dostępna wersja mikrokontrolera → **M5Stack Core 16MB**.

Przebieg ćwiczenia

Schemat organizacji programu

zdarzenie wejściowe → zmiana zmiennej/stanu → funkcja logiki → aktualizacja GUI

Zadanie 1. Pierwszy program

Do wykonania

Utwórz program, który po uruchomieniu wyświetla na ekranie tekst:

Witaj M5Stack

Następnie dodaj drugi napis zawierający numer ćwiczenia laboratoryjnego.

Sprawdź:

- gdzie w UIFlow definiowana jest zawartość ekranu,
- które bloki wykonywane są podczas inicjalizacji,
- jak zmiana właściwości obiektu wpływa na wygląd GUI.

Sposób realizacji i weryfikacja

1. Rozdziel elementy wykonywane tylko podczas startu od elementów, które będą później zmieniane w reakcji na zdarzenia.
2. Nadaj obiektom GUI jednoznaczne nazwy, np. `etykieta_tytul` i `etykieta_wartosc`; unikaj nazw domyślnych utrudniających dalszą pracę.
3. Po uruchomieniu sprawdź zachowanie po ponownym wgraniu programu i restarcie urządzenia; ekran powinien zawsze startować w tym samym stanie.

Warunek odbioru: Prowadzący wskazuje obiekt GUI, a student potrafi pokazać blok, który ustawia jego właściwość.

Zadanie 2. Zmienna i licznik

Utwórz zmienną:

```
licznik = 0
```

Program powinien:

1. zwiększać licznik po naciśnięciu przycisku A,
2. zmniejszać licznik po naciśnięciu przycisku B,
3. zerować licznik po naciśnięciu przycisku C,
4. wyświetlać aktualną wartość na ekranie.

Sposób realizacji i weryfikacja

1. Przed obsługą przycisków ustaw `licznik=0` i od razu wyświetl tę wartość; dzięki temu stan wewnętrzny i ekran są zgodne od początku.
2. W procedurze każdego przycisku wykonuj kolejno: zmiana zmiennej, ograniczenie (jeżeli jest potrzebne), odświeżenie ekranu.
3. Przetestuj serię co najmniej 10 naciśnięć A, 10 naciśnięć B oraz reset C; zwróć uwagę, czy jedno naciśnięcie powoduje dokładnie jedną zmianę.

Warunek odbioru: Wartość na ekranie jest zawsze zgodna ze zmienną, także po zerowaniu.

Pseudokod / zapis algorytmu:

```
po_nacisnieciu_A:
    licznik = licznik + 1
    aktualizuj_ekran()

po_nacisnieciu_C:
    licznik = 0
    aktualizuj_ekran()
```

Wymagany rezultat

Na ekranie powinna być widoczna aktualna wartość zmiennej, a zmiana powinna następować natychmiast po obsłużeniu odpowiedniego zdarzenia.

Zadanie 3. Instrukcja warunkowa

Zmodyfikuj program tak, aby kolor napisu zależał od wartości licznika:

- wartość < 0 – kolor sygnalizujący wartość ujemną,
- wartość $= 0$ – kolor neutralny,
- wartość > 0 – kolor sygnalizujący wartość dodatnią.

Zastanów się, w którym miejscu programu najlepiej umieścić logikę odpowiedzialną za aktualizację wyglądu interfejsu.

Sposób realizacji i weryfikacja

1. Zapisz warunek jako trzy rozłączne przypadki: wartość ujemna, zero, wartość dodatnia; żaden przypadek nie może być pominięty.
2. Zmieniaj wyłącznie sposób prezentacji (np. kolor), nie modyfikuj w tym miejscu wartości licznika.
3. Sprawdź wartości graniczne -1, 0 i +1; to najszybszy test poprawności rozgałęzienia.

Pseudokod / zapis algorytmu:

```
if licznik < 0:
    kolor = UJEMNY
elif licznik == 0:
    kolor = NEUTRALNY
```

```
else:
    kolor = DODATNI
```

Zadanie 4. Funkcja aktualizująca GUI

Utwórz funkcję:

```
aktualizuj_ekran()
```

Funkcja powinna:

1. odczytać aktualną wartość zmiennej,
2. zaktualizować napis na ekranie,
3. ustawić właściwy kolor,
4. nie zmieniać samej wartości licznika.

Następnie wywołuj ją po każdym zdarzeniu zmieniającym stan programu.

Sposób realizacji i weryfikacja

1. Funkcja ma być jedynym miejscem, w którym przeliczana jest treść i kolor etykiety licznika.
2. Nie umieszczaj w funkcji obsługi przycisków ani zmiany wartości `licznik`; funkcja prezentacji ma być pozbawiona efektów ubocznych względem logiki procesu.
3. Po refaktoryzacji sprawdź, czy w procedurach A/B/C nie pozostały powielone bloki ustawiające tekst lub kolor.

Pseudokod / zapis algorytmu:

```
funkcja aktualizuj_ekran():
    etykieta_wartosc = str(licznik)
    ustaw_kolor_zaleznie_od(licznik)
```

Zadanie 5. Prosta maszyna stanów

Utwórz zmienną `stan` przechowującą jeden z trzech stanów:

ZATRZYMANIE, URUCHOM, PAUZA

Przypisz przyciski:

Przycisk	Działanie
A	przejdź do stanu PRACA
B	przejdź do stanu PAUZA
C	przejdź do stanu ZATRZYMANIE

Na ekranie należy wyświetlać aktualny stan programu.

Sposób realizacji i weryfikacja

1. Traktuj `stan` jako jedyne źródło informacji o trybie pracy; nie twórz dodatkowych niezależnych flag PRACA/PAUZA.
2. Przy każdej zmianie stanu odśwież ekran i wykonaj czynności wejścia do stanu, np. wyzerowanie animacji po ZATRZYMANIE.
3. Narysuj trzy węzły ZATRZYMANIE, URUCHOM i PAUZA oraz strzałki odpowiadające przyciskom; porównaj diagram z implementacją.

Pseudokod / zapis algorytmu:

```
A -> stan = PRACA
B -> stan = PAUZA    (tylko gdy PRACA)
C -> stan = ZATRZYMANIE

petla:
    if stan == PRACA: wykonaj_prace()
```

Zadanie 6. Analiza wygenerowanego kodu

Przełącz widok programu z Blockly na kod MicroPython.

Odszukaj fragmenty odpowiadające za:

1. utworzenie zmiennej,
2. instrukcję warunkową,
3. funkcję,
4. reakcję na naciśnięcie przycisku,
5. aktualizację elementu GUI.

Cel tego etapu

Nie jest wymagane samodzielne pisanie programu w MicroPython. Celem jest pokazanie, że program graficzny jest reprezentacją zwykłego programu tekstowego.

Sposób realizacji i weryfikacja

1. Dla każdego wskazanego bloku Blockly odnajdź odpowiadający mu fragment MicroPython i zaznacz relację jeden-do-jednego lub jeden-do-wielu.
2. Zwróć uwagę, że procedury zdarzeniowe są funkcjami wywoływanymi przez mechanizm zdarzeń, a nie częścią liniowo wykonywanej pętli.
3. Wskaż, gdzie w kodzie przechowywany jest stan programu oraz w którym miejscu wykonywana jest aktualizacja GUI.

Warunek odbioru: Student potrafi wyjaśnić znaczenie co najmniej czterech fragmentów wygenerowanego kodu bez ich przepisywania.

Zadanie końcowe**Integracja zadań etapowych**

Zadanie końcowe wykonaj przez **rozbudowę programu powstającego w zadaniach 1–6**. Zachowaj ekran z zadania 1, licznik z zadania 2, klasyfikację wartości z zadania 3, funkcję aktualizacji interfejsu z zadania 4 i automat stanów z zadania 5. Zadanie 6 wykorzystaj do porównania gotowego rozwiązania z wygenerowanym kodem MicroPython. Nie twórz osobnego programu końcowego.

Rozbuduj projekt do aplikacji odliczającej czas pracy. Wyświetl czas w formacie **hh:mm:ss**. Przyciski powinny przełączać program pomiędzy stanami ZATRZYMANIE, PRACA i PAUZA. Prosta animacja ma działać wyłącznie w stanie PRACA, zatrzymywać się w PAUZA, a po przejściu do ZATRZYMANIE wracać do położenia początkowego.

Rozszerzenie: wykorzystaj licznik z wcześniejszych zadań jako licznik wykonanych cykli animacji oraz wyświetl krótką polską informację opisującą aktualny stan: **PRACA**, **PAUZA** albo **ZATRZYMANIE**.

Kryteria zaliczenia

Ćwiczenie uznaje się za zaliczone, jeżeli student:

- uruchomi program na urządzeniu,
- poprawnie obsłuży przyciski,
- wykorzysta co najmniej jedną zmienną,
- zastosuje instrukcję warunkową,
- utworzy i wykorzysta funkcję,
- zrealizuje prostą maszynę stanów,
- potrafi wyjaśnić zasadę działania swojego programu.

Sprawozdanie

Podstawą zaliczenia ćwiczenia jest sprawozdanie, które dotyczy wykonania zadania końcowego. Poszczególne zadania (od 1-6) etapowe zaliczane są na zasadzie demonstracji. Sprawozdanie należy przesłać w formie pliku **.pdf* oraz pliku z kodem programu do weryfikacji. W dokumencie należy przedstawić:

- kod programu w UIFlow,
- zdjęcia działającego stanowiska,
- krótkiego opisu zastosowanego algorytmu,
- wnioski.