

Graficzne programowanie mikrokontrolerów

M5Stack / UIFlow

Laboratorium 1

Podstawy UIFlow i interfejs operatorski

Czas trwania: 3 × 45 min**Tryb:** studia niestacjonarne**Jednostka:** Katedra Elektrotechniki i Technologii Nadprzewodnikowych**Prowadzący:** dr inż. Michał Łanczont

Cel ćwiczenia

Celem ćwiczenia jest poznanie platformy M5Stack i środowiska UIFlow oraz budowa interfejsu operatorskiego w Blockly. Student wykorzystuje zmienne, warunki, pętle, funkcje i zdarzenia do realizacji automatu stanów, obsługuje przyciski Core i dotyk Core2, oddziela logikę sterowania od prezentacji oraz analizuje wygenerowany kod MicroPython.

1. Wprowadzenie

Wyposażenie: M5Stack Core, przewód USB i komputer z UIFlow; Core2 do krótkiej próby dotyku. Bez elementów wykonawczych.

Organizacja pracy

Zapoznaj się z opisem stosowanych elementów, wykonaj cztery zadania cząstkowe, a następnie połącz przygotowane funkcje w jednym zadaniu końcowym. Polecenia, szczegółowe opisy, algorytmy i testy znajdują się bezpośrednio przy zadaniach. Zachowuj działające wersje projektu i wykorzystuj je w kolejnych etapach.

Obowiązuje zakres określony w poleceniach zadań cząstkowych i zadania końcowego. Opisane możliwości rozbudowy pozwalają pogłębić materiał. Wyniki testów i zdjęcia stanowiska zbierz podczas zajęć, a sprawozdanie opracuj po zajęciach.

Platforma i programowanie blokowe

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**.

Interfejs operatorski i zdarzenia

Interfejs graficzny w systemie wbudowanym pełni dwie podstawowe funkcje: przedstawia stan procesu oraz umożliwia użytkownikowi wydawanie poleceń. W kolejnych laboratoriach ekran będzie wykorzystywany jako panel HMI, dlatego należy rozdzielić **logikę sterowania** od **logiki prezentacji**.

Przy projektowaniu interfejsu należy przyjąć prostą zależność:

zdarzenie → zmiana stanu → aktualizacja GUI

Elementy interfejsu nie powinny samodzielnie przechowywać informacji o stanie procesu. Stan powinien być zapisany w zmiennych programu, a GUI jedynie go prezentować.

Programowanie zdarzeniowe

Zdarzeniem może być m.in. naciśnięcie przycisku, dotknięcie obszaru ekranu, upływ czasu lub zmiana wartości wejścia. Obsługa zdarzenia powinna być możliwie krótka: zmieniać odpowiednią zmienną, wywoływać funkcję i aktualizować interfejs.

Uwaga

Na stanowisku z M5Stack Core używaj przycisków A/B/C. Core2 traktuj jako oddzielne urządzenie z ekranem dotykowym. Nie osadzaj Core2 mechanicznie na Demo Board.

Wskazówki do wariantu realizowanego na zajęciach

W zadaniach cząstkowych stosuj stany ZATRZYMANIE, PRACA i ALARM. Stan PAUZA można wykorzystać przy rozbudowie pomiaru czasu. Nazwa URUCHOM oznacza polecenie, a PRACA stan

programu. Poszczególne przykłady pokazują różne zastosowania przycisków A/B/C; w gotowym projekcie stosuj przypisanie określone w poleceniu zadania.

Platforma i środowisko

M5Stack jest platformą opartą na ESP32. Core udostępnia ekran i przyciski A/B/C, a Core2 pozwala obsługiwać GUI dotykiem. UIFlow przedstawia program w postaci bloków Blockly; podgląd MicroPython pozwala sprawdzić ich znaczenie bez ręcznego pisania kodu.

W Setup ustaw wartości początkowe i zbuduj ekran. W Loop wykonuj krótkie czynności okresowe. Procedury zdarzeń reagują na przyciski lub dotyk. Nazwy własnych zmiennych i funkcji zapisuj po polsku bez znaków diakrytycznych; komunikaty ekranowe mogą zawierać polskie znaki.

Przygotowanie stanowiska

Oprogramowanie układowe i wersja UIFlow muszą być zgodne z modelem Core/Core2. Instalację i sprawdzenie połączenia wykonuje prowadzący przed zajęciami. Kolejne zadania rozwijają jeden projekt; po każdym etapie zapisz działającą wersję.

Zmienne i funkcje

Zmienna nastawa przechowuje liczbę 0–100, a stan określa tryb aplikacji. Funkcja aktualizuj_gui() wyświetla te wartości; nie zmienia nastawy ani stanu. Przycisk zgłasza żądanie, funkcja logiki ocenia warunek przejścia, a GUI prezentuje wynik.

Zdarzenie a odczyt poziomu

Pojedyncze naciśnięcie powinno powodować pojedynczą akcję. W Core użyj zdarzeń przycisków, a w Core2 zdarzeń obszarów dotykowych. Samo przytrzymanie palca nie powinno wielokrotnie przełączać stanu. Obszary dotykowe nie mogą się nakładać.

Automat stanów panelu

Stan	Działanie
ZATRZYMANIE	Licznik czasu nie rośnie; możliwa zmiana nastawy.
PRACA	Licznik czasu rośnie; wyświetlany jest stan i nastawa.
ALARM	Praca zablokowana; widoczna przyczyna alarmu.

Próg alarmu w tym ćwiczeniu jest umowny: nastawa > 80. Obniżenie nastawy nie kasuje alarmu. C w stanie ALARM oznacza ZERUJ, dozwolone tylko przy nastawie ≤ 80; reset prowadzi do ZATRZYMANIE. Kolejne naciśnięcie C dopiero uruchamia pracę.

```
zdarzenie A/B -> zmien nastawa -> ogranicz 0..100
sprawdz_alarm() -> aktualizuj_gui()
zdarzenie C -> ocen przejście stanu -> aktualizuj_gui()
```

Pomiar czasu

Użyj timera lub różnicy kolejnych odczytów zegara. Nie stosuj długiego oczekiwania blokującego reakcję na przyciski. Czas pracy zliczaj wyłącznie w PRACA.

2. Zadania cząstkowe

Wykonuj zadania w podanej kolejności, rozwijając ten sam projekt. Każde zadanie zawiera polecenie, opis wykonania, przykłady i sposób sprawdzenia wyniku. Przykłady pokazują również dodatkowe

warianty działania; zakres wymagany określa polecenie na początku zadania. Zapisuj działającą wersję po każdym etapie.

Schemat organizacji programu

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

Zadanie 1. Ekran i inicjalizacja

Polecenie zadania

Utwórz ekran z nazwą ćwiczenia, nastawą, stanem i licznikiem sekund. W Setup ustaw nastawa=0, stan=ZATRZYMANIE i czas_pracy=0. Nadaj elementom GUI czytelne nazwy i uruchom program.

Warunek odbioru: Po restarcie ekran pokazuje zawsze wartości początkowe.

Sposób wykonania i przykłady

Wykorzystaj poniższe opisy i algorytmy do realizacji polecenia. Szczegółowe warianty wykraczające poza polecenie są możliwościami rozbudowy tego samego zadania.

Pierwszy program

Przykład zastosowania

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.

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

Budowa ekranu operatorskiego

Przykład zastosowania

Zaprojektuj ekran zawierający: tytuł aplikacji, pole wartości zadanej, pole aktualnego stanu, wskaźnik pracy oraz dwa elementy sterujące odpowiadające funkcjom URUCHOM i ZATRZYMANIE.

Przyjmij trzy stany aplikacji: ZATRZYMANIE, PRACA oraz ALARM. Zadbaj o czytelny podział ekranu i stałe położenie najważniejszych informacji.

Sposób realizacji i weryfikacja

1. Najpierw rozmieść elementy statyczne, potem wartości dynamiczne i na końcu elementy sterujące; unikniesz późniejszego przesuwania całego interfejsu.
2. Przyjmij stałą konwencję prezentacji: nazwa parametru, wartość, jednostka; stan procesu powinien być widoczny bez wchodzenia do dodatkowego ekranu.
3. Przed programowaniem logiki wykonaj test czytelności: z odległości około 0,5–1 m powinny być rozpoznawalne stan i wartość zadana.

Sprawdzenie: Ekran zawiera komplet informacji, a URUCHOM/ZATRZYMANIE są jednoznacznie odróżnione od elementów tylko informacyjnych.

Powiązanie GUI ze zmiennymi

Utwórz zmienne `stan`, `nastawa` i `licznik`. Przygotuj funkcję:

```
aktualizuj_gui()
```

Funkcja powinna wyświetlać bieżące wartości zmiennych i zmieniać wygląd kontrolki stanu zależnie od `stan`.

Sposób realizacji i weryfikacja

1. Ustal wartości początkowe `stan=ZATRZYMANIE`, `nastawa=0`, `licznik=0` i dopiero potem wywołaj `aktualizuj_gui()`.
2. W funkcji prezentacji zamień dane na tekst tylko w jednym miejscu; nie przechowuj osobnych kopii tych samych wartości wyłącznie na potrzeby GUI.
3. Dodaj wizualne rozróżnienie ZATRZYMANIE/PRACA/ALARM, ale nie używaj samego koloru jako jedynej informacji - wyświetl także nazwę stanu.

Pseudokod / zapis algorytmu:

```
funkcja aktualizuj_gui():
    pokaz(nastawa)
    pokaz(licznik)
    pokaz(stan)
    ustaw_wyglad_kontrolki(stan)
```

Zadanie 2. Nastawa i wspólna funkcja GUI

Polecenie zadania

Przycisk A zwiększa nastawę o 5, B zmniejsza ją o 5. Ogranicz wartość do 0–100. Utwórz `aktualizuj_gui()` i wywołuj ją po zmianie nastawy. Wprowadź opis NORMALNA dla nastawy ≤ 80 oraz PRÓG PRZEKROCZONY dla > 80 ; oprócz tekstu zmieniaj kolor.

Warunek odbioru: Sprawdź nastawy 0, 80, 85 i 100 oraz kolejne naciśnięcia poza zakresem. Funkcja GUI nie zmienia nastawy.

Zapis projektu

Zachowaj wersję po zadaniu 2. Ten sam ekran i funkcja będą używane w automacie stanów oraz w zadaniu końcowym.

Sposób wykonania i przykłady

Wykorzystaj poniższe opisy i algorytmy do realizacji polecenia. Szczegółowe warianty wykraczające poza polecenie są możliwościami rozbudowy tego samego zadania.

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 naciśnień A, 10 naciśnień B oraz reset C; zwróć uwagę, czy jedno naciśnięcie powoduje dokładnie jedną zmianę.

Sprawdzenie: 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()
```

Rezultat przykładu

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

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
```

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)
```

Obsługa przycisków Core

Przypisz:

- A – zwiększenie wartości zadanej,
- B – zmniejszenie wartości zadanej,
- C – przełączenie ZATRZYMANIE/PRACA.

Ogranicz wartość zadaną do zakresu 0–100. Po każdej zmianie wywołaj funkcję aktualizującą GUI.

Sposób realizacji i weryfikacja

1. Dla A i B wykonuj zmianę nastawy krokowo, np. o 1 lub 5, a następnie ogranicz wynik do 0–100.
2. Dla C zaimplementuj świadome przejście ZATRZYMANIE/PRACA; nie przełączaj do PRACA, jeśli aktywny jest ALARM.
3. Przetestuj wartości graniczne 0 i 100 oraz wielokrotne naciskanie przycisków poza zakresem.

Pseudokod / zapis algorytmu:

```
nastawa = ogranicz(nastawa + krok, 0, 100)
if C and stan != ALARM:
    stan = PRACA if stan == ZATRZYMANIE else ZATRZYMANIE
aktualizuj_gui()
```

Zadanie 3. Stan pracy i blokada alarmowa

Polecenie zadania

Dodaj stany ZATRZYMANIE, PRACA i ALARM według opisu ze wstępu. C przełącza PRACA/ZATRZYMANIE, a w ALARM realizuje warunkowy reset. Przekroczenie 80 ma pierwszeństwo przed uruchomieniem i natychmiast ustawia ALARM. Wyświetlaj przyczynę oraz nazwę stanu.

Warunek odbioru: Dla 85 nie można uruchomić pracy. Po obniżeniu do 80 i resecie układ pozostaje zatrzymany.

Sposób wykonania i przykłady

Wykorzystaj poniższe opisy i algorytmy do realizacji polecenia. Szczegółowe warianty wykraczające poza polecenie są możliwościami rozbudowy tego samego zadania.

Prosta maszyna stanów

Utwórz zmienną **stan** przechowującą jeden z trzech stanów:

ZATRZYMANIE, URUCHOM, PAUZA

Przypisz przyciski:

Przycisk	Działanie
A	przejście do stanu PRACA
B	przejście do stanu PAUZA
C	przejście 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()
```

Kontrolka stanu i alarm

Dodaj warunek alarmowy: gdy **nastawa** przekroczy 80, aplikacja przechodzi do stanu **ALARM**. W stanie alarmowym:

- kontrolka powinna zmienić kolor,
- na ekranie powinien pojawić się komunikat,
- **URUCHOM** nie może uruchomić programu,
- **ZERUJ** powinien skasować alarm po obniżeniu wartości zadanej.

Sposób realizacji i weryfikacja

1. Oddziel warunek wykrycia alarmu od procedury jego kasowania; samo obniżenie nastawy nie powinno automatycznie uruchamiać systemu.
2. Po wejściu do **ALARM** zapamiętaj ten stan, zablokuj **URUCHOM** i pokaż przyczynę alarmu.
3. **ZERUJ** może przywrócić **ZATRZYMANIE** tylko wtedy, gdy przyczyna alarmu już nie występuje; po resetie wymagaj osobnego **URUCHOM**.

Pseudokod / zapis algorytmu:

```
if nastawa > 80:
    stan = ALARM

if ZERUJ and nastawa <= 80:
    stan = ZATRZYMANIE
```

Rozdzielenie logiki i prezentacji

Uporządkuj program tak, aby zmiana stanu była realizowana w oddzielnej funkcji, a wygląd ekranu w funkcji `aktualizuj_gui()`. Prowadzący powinien móc wskazać w programie część odpowiedzialną za logikę procesu i część odpowiedzialną za prezentację.

Sposób realizacji i weryfikacja

1. Utwórz funkcje typu `zadanie_uruchomienia()`, `zadanie_zatrzymania()`, `sprawdz_alarm()` oraz `aktualizuj_gui()`.
2. Funkcje zdarzeń przycisków/dotyku powinny jedynie zgłaszać żądanie lub zmieniać dane wejściowe; decyzję o dozwolonym przejściu wykonuje logika stanu.
3. Wprowadź jedno miejsce odpowiedzialne za okresowe zwiększanie licznika pracy w **PRACA**.

Pseudokod / zapis algorytmu:

```
petla:
    sprawdz_alarm()
    if stan == PRACA and minela_1s:
        licznik += 1
```

```
aktualizuj_gui_gdy_zmiana()
```

Rezultat przykładu

Panel operatorski reaguje na zdarzenia, poprawnie prezentuje stan programu i nie zawiera powielonej logiki aktualizacji elementów GUI w wielu miejscach programu.

Zadanie 4. Zdarzenia dotykowe i podgląd kodu

Polecenie zadania

Na Core2 utwórz mały ekran z przyciskami PLUS, MINUS i START/STOP oraz etykietą nastawy. Powiąż dotyk z takimi samymi regułami zmiany nastawy jak na Core; nie przenoś jeszcze całego panelu. Sprawdź jedno naciśnięcie i przytrzymanie. W podglądzie MicroPython wskaż zmienną, warunek, funkcję i obsługę zdarzenia.

Warunek odbioru: Jedno dotknięcie powoduje jedną zmianę. Student wskazuje cztery odpowiadające blokom fragmenty kodu.

Do zadania końcowego wróć do projektu na Core. Próba dotyku służy przygotowaniu do pracy z terminalem Core2 w ćwiczeniu 4.

Sposób wykonania i przykłady

Wykorzystaj poniższe opisy i algorytmy do realizacji polecenia. Szczegółowe warianty wykraczające poza polecenie są możliwościami rozbudowy tego samego zadania.

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.

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

Przepływ informacji w panelu operatorskim

przycisk / dotyk → zdarzenie → zmiana stan/nastawa → aktualizuj_gui() → ekran

GUI dotykowe na Core2

Na Core2 przygotuj analogiczny panel sterujący, wykorzystując zdarzenia dotykowe. Zaimplementuj co najmniej trzy pola aktywne: URUCHOM, ZATRZYMANIE i ZERUJ. Zdarzenie dotykowe nie może być realizowane przez ciągle sprawdzanie współrzędnych bez kontroli stanu naciśnięcia.

Sposób realizacji i weryfikacja

1. Zdefiniuj obszary dotykowe tak, aby nie nakładały się i były większe niż sam napis; ułatwia to obsługę palcem.
2. Reaguj na zdarzenie naciśnięcia/zwolnienia lub zmianę stanu dotyku, a nie na każdą iterację pętli z palcem przyłożonym do ekranu.
3. Dodaj krótką informację zwrotną po dotknięciu, np. zmianę wyglądu przycisku lub komunikat wykonanej akcji.

Pseudokod / zapis algorytmu:

```
if zdarzenie_dotyku == WCISNIETY and trafiono(URUCHOM):
    zadanie_uruchomienia()
if zdarzenie_dotyku == WCISNIETY and trafiono(ZATRZYMAJ):
    zadanie_zatrzymania()
```

3. Zadanie końcowe**Polecenie zadania końcowego****Panel pomiaru czasu pracy**

Rozbuduj panel na Core o licznik czasu aktywnej pracy w formacie mm:ss oraz prostą kontrolkę migającą tylko w PRACA. Zachowaj nastawę, przyciski, alarm i funkcję GUI z zadań częściowych. Nie twórz nowej aplikacji.

- Zatrzymanie i alarm wstrzymują naliczanie; dotychczasowy czas pozostaje widoczny. Po ponownym starcie licznik kontynuuje pomiar.
- Przy każdym starcie programu czas wynosi zero. Miganie wykonuj okresowo, bez długiego blokowania pętli.
- Wyświetlaj jednocześnie nastawę, czas, stan i przyczynę alarmu. Kolor uzupełnia komunikat tekstowy.

Testy i odbiór

Próba	Oczekiwana reakcja
Start, następnie stop	Czas rośnie tylko w PRACA.
Nastawa 85 podczas pracy	ALARM, czas zatrzymany, start zablokowany.
Nastawa 80 i C	ZATRZYMANIE; kolejne C uruchamia.
Restart urządzenia	Nastawa 0, czas 00:00, ZATRZYMANIE.

Kryteria zaliczenia

Działający panel, poprawne ograniczenia, trzy stany, alarm z ręcznym resetem, pomiar czasu oraz wyjaśnienie funkcji GUI i zdarzeń. Zadania cząstkowe podlegają demonstracji.

Sprawozdanie

Zapisz projekt UIFlow, zdjęcie panelu, diagram trzech stanów oraz wyniki czterech testów. Na tej podstawie przygotuj krótkie sprawozdanie PDF i dołącz projekt. Podczas zajęć zbierz materiały; redakcja sprawozdania odbywa się po zajęciach.

Możliwość rozbudowy: pełne przeniesienie panelu na Core2 lub format hh:mm:ss.

Możliwości rozbudowy zadania końcowego

Zakres rozszerzeń

Zadanie końcowe możesz rozbudować o opisane niżej funkcje. Wykorzystaj działające moduły przygotowane w zadaniach cząstkowych. Możliwości rozbudowy nie zwiększają obowiązkowego zakresu zadania końcowego. Dobierz testy oraz dokumentację do faktycznie zrealizowanych funkcji.

Pomiar czasu z pauzą i animacją

Wykorzystanie przygotowanych funkcji

Rozszerzenie projektu wykonaj przez **rozbudowę programu powstającego w zadaniach 1–4**. Zachowaj ekran z zadania 1, licznik z zadania 2, klasyfikację wartości z zadania 2, funkcję aktualizacji interfejsu z zadania 2 i automat stanów z zadania 3. Zadanie 4 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**.

Dotykowy panel procesu z alarmem

Wykorzystanie przygotowanych funkcji

Rozszerzenie projektu jest scaleniem efektów zadań 1–4. Użyj ekranu z zadania 1, zmiennych programu z zadania 1, obsługi przycisków Core z zadania 2, obsługi dotyku Core2 z zadania 4, alarmu z zadania 3 oraz rozdzielania logiki i prezentacji z zadania 3. Wszystkie elementy mają pracować na wspólnych zmiennych i być odświeżane przez jedną funkcję interfejsu.

Rozbuduj projekt do panelu sterowania symulowanym procesem. Panel ma zawierać pola **URUCHOM**, **ZATRZYMAJ** i **ZERUJ**, nastawę 0–100, licznik czasu pracy oraz kontrolkę stanu. W stanie **PRACA** licznik czasu zwiększa się, w **ZATRZYMANIE** pozostaje zatrzymany, a w **ALARM** polecenie **URUCHOM** jest blokowane. Na Core2 sterowanie wykonaj z użyciem ekranu dotykowego.

Rozszerzenie: dodaj komunikat wyjaśniający przyczynę blokady uruchomienia oraz licznik wystąpień alarmu.

Dodatkowe testy przy rozbudowie zadania końcowego

W zależności od wybranych funkcji sprawdź poniższe wymagania. Dotyczą one odpowiednich rozszerzeń projektu.

- 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.
- przygotuje czytelny panel GUI,
- zastosuje obsługę zdarzeń,
- powiąże GUI ze zmiennymi programu,
- utworzy funkcję aktualizującą interfejs,
- zaimplementuje co najmniej trzy stany pracy,
- potrafi wyjaśnić różnicę pomiędzy stanem programu a jego prezentacją.

Dokumentowanie dodatkowych funkcji

Przygotuj sprawozdanie PDF dotyczące wykonanego rozszerzenia i dołącz projekt UIFlow. Udokumentuj zastosowane rozwiązania, wybierając odpowiednie elementy poniższej listy.

- kod programu w UIFlow,
- zdjęcia działającego stanowiska,
- krótkiego opisu zastosowanego algorytmu,
- wnioski.
- zdjęcia działającego stanowiska lub czytelne zrzuty ekranu,
- krótki opis zastosowanego algorytmu wraz ze schematem blokowym, diagramem stanów lub pseudokodem,
- opis sposobu testowania programu,