Systemy wbudowane

Dystrybutor napojów

Konrad Jachimstal	Patryk Janicki	Dobromir Kata
211807	211951	176555

2019/2020

Spis treści

1 Wprowadzenie **4**

2 Wykorzystane elementy **4**
2.1 Platforma Arduino UNO R3 4
2.2 Podzespoły 4

3 Funkcjonalności i opis **5**
3.1 Wykrywanie naczynia 5
3.2 Sygnalizacja stanu za pomocą diody LED 5
3.3 Miganie diody LED podczas nalewania 5
3.4 Pomiar poziomu cieczy w zbiorniku 5
3.5 Wyświetlanie poziomu cieczy w zbiorniku 5
3.6 Komunikaty dźwiękowe z użyciem buzzera 6
3.7 Pomiar temperatury cieczy w zbiorniku 6
3.8 Kontrola programu za pomocą przycisku 6
3.9 Sterowanie pompą 7

4 Podział obowiązków **7**

5 Instrukacja użytkowania **8**
5.1 Schemat 8
5.2 Podłączenie 8
5.3 Wybór ilości płynu 8
5.4 Nalewanie 9
5.5 Poziom płynu w zbiorniku 9

6 Schemat układu sterowania pompą **9**

7 Wykorzystane biblioteki **10**
7.1 NewPing 10
7.2 ShiftRegister74HC595 10
7.3 LiquidCrystal_I2C 10
7.4 DS18B20 10

8 Opis algorytmu **12**
8.1 Sekcje deklaracji stałych 12
8.2 Sekcja deklaracji zmiennych globalnych 12
8.3 Sekcja instancji klas bibliotek 13
8.4 Sekcja inicjalizacji 13

8.5 Sekcja pętli programu 14
8.6 Sekcja zadań 14
8.7 Sekcja dźwięków 16
8.8 Sekcja ekranów 16
8.9 Sekcja funkcji 17
8.9.1 canPerformTask() 17
8.9.2 updateTaskTime() 17
8.9.3 isCupPresent() 17
8.10 Podsumowanie 18

9 Analiza rodzajów i skutków możliwych błędów 18
9.1 Awaria wyświetlacza znakowego i segmentowego 18
9.2 Awaria pompy 18
9.3 Awaria czujników 18
9.4 Awaria diody, przycisków, buzzera 19

10 Bibliografia 19

1 Wprowadzenie

Celem zadania było stworzenie programu dla mikrokontrolera z użyciem różnych funkcjonalności (GPIO, ADC, PWM, Timer, SPI, I2C, itd.) i współpracującego z urządzeniami zewnętrznymi (wyświetlacz, akcelerometr, żyroskop, silnik, itd.). Do realizacji zadania został wybrany projekt **dystrybutora płynu** z jednym zbiornikiem na ciecz, zbudowany na podstawie mikrokontrolera **Arduino UNO R3** (Atmega 328P-AU+CH340G).

2 Wykorzystane elementy

2.1 Platforma Arduino UNO R3

- mikrokontroler ATmega328P (16MHz);
- 32KB pamięci Flash, 2048KB SRAM;
- 1024B EEPROM/HEF;
- UART x1, SPI x2, I2C x1, PWM x6, Input Capture x1, CCP x1;
- Zegar 8-bit x2, zegar 16-bit x1;

2.2 Podzespoły

Urządzenia peryferyjne	Mini pompa wody - MY-DB5 - 5V
	Ultradźwiękowy czujnik odległości JSN-SR04T
	Sonda wodoodporna z czujnikiem temperatury DS18B20
	Wyświetlacz LCD 2x16 z konwerterem I2C LCM1602
	Wyświetlacz LED linijka DC-10SRWA
	Czujnik siły nacisku RA9P - 4kg okrągły
	Buzzer z generatorem 5V 12mm
Układy scalone	Rejestr przesuwny 74HC595 - 8-bitowy
	Tranzystor N-MOSFET IRL540NPBF
	Transoptor jednokanałowy PC817
Inne	Przycisk monostabilny, okrągły 250V/3A
	Dioda Schottky 1N5819 1A
	Dioda LED 5mm czerwona

3 Funkcjonalności i opis

3.1 Wykrywanie naczynia

Umieszczony w podstawie czujnik ultradźwiękowy wysyła fale prostopadle do górnej części dystrybutora i odbiera echo sygnału. Efektem zakrycia czujnika jest zbyt krótki sygnał, co skutkuje zerową wartością na wyjściu. Program sprawdza, czy od ostatniego poprawnego pomiaru minął czas określony w konfiguracji ($1s$). Pomiar wykonywany jest co $100ms$.

3.2 Sygnalizacja stanu za pomocą diody LED

Dioda LED umieszczona nad naczyniem sygnalizuje czy naczynie zostało wykryte, zgodnie ze stanem uzyskanym przy pomocy czujnika ultradźwiękowego. W przypadku zapalenia się diody urządzenie jest gotowe do rozpoczęcia procesu nalewania.

3.3 Miganie diody LED podczas nalewania

Podczas trwania procesu nalewania dioda naprzemiennie otrzymuje na wyjście stan wysoki i niski, przy czym każdy z nich trwa $500ms$, sygnalizując tym samym nalewanie.

3.4 Pomiar poziomu cieczy w zbiorniku

Przybliżony poziom cieczy uzyskiwany jest za pomocą czujnika nacisku umieszczonego pod zbiornikiem. Program odczytuje wartość na wejściu analogowym,przy czym rezystancja odzwierciedla siłę nacisku. Otrzymywana wartość jest skalowana z pominięciem wagi pustego zbiornika. Należy mieć jednak na uwadze, że czynniki zewnętrzne mogą wpływać na wynik pomiaru - temperatura (10%) oraz wilgotność (20%). Pomiar jest wykonywany co $1s$.

3.5 Wyświetlanie poziomu cieczy w zbiorniku

Piny wyświetlacza 10-segmentowego w kolorze czerwonym zostały połączone z dwoma złączonymi 8-bitowymi rejestrami przesuwnymi (łącznie 16 bitów). Przetwarzana wartość została wyskalowana tak, aby przy pokryciu pompy wodą w zbiorniku wynosiła zero. Wartości otrzymywane z czujnika są z zakresu $0-1024$. Otrzymana wartość z zakresu $800-1024$ (z uwzględnieniem wagi zbiornika i elementów podtrzymujących) mapowana jest na $0-10$, co odpowiada liczbie segmentów 10-poziomowego wyświetlacza.

3.6 Komunikaty dźwiękowe z użyciem buzzera

Urządzenie zostało wyposażone w buzzer z generatorem, którego dźwięk informuje o dokonanej akcji lub stanie urządzenia. Zasilanie buzzera sterowane jest poprzez wyjście $GPIO$ z PWM na mikrokontrolerze, a poprzez odpowiednie napięcie na wyjściu ustalany jest ton, który pozwala na zróżnicowanie sygnałów dźwiękowych.

Sygnały dźwiękowe wydawane przez urządzenie:

- sygnał ciągły - czas trwania:
  - użycie przycisku przełączenia programu - $200ms$;
  - użycie przycisku akcji - $1s$;
  - samoczynne przełączenie ekranu na wyświetlaczu LCD - $100ms$;
  - pomyślne zakończenie procesu nalewania - $500ms$;
  - zmiana stanu - wykryte naczynie - $50ms$;
- sygnał odmowy - sekwencja:
  - sygnał ciągły - $200ms$;
  - brak sygnału - $50ms$;
  - sygnał ciągły - $500ms$;

3.7 Pomiar temperatury cieczy w zbiorniku

Pomiar temperatury odbywa się za pomocą interfejsu OneWire. Pomiar i wyświetlenie wartości następuje co $20s$. Wartość zaokrąglana jest do jednego miejsca po przecinku i wyświetlana w stopniach Celsjusza.

- dokładność pomiaru: 0.5°C (w zakresie od -10°C do 85°C);
- zakres pomiaru: od -55°C do 125°C;

3.8 Kontrola programu za pomocą przycisku

W urządzeniu zlokalizowano dwa przyciski monostabilne. Program wykrywa naciśnięcie przycisku i wykonuje odpowiednie akcje. Zielony przycisk służy do przełączenia ilości płynu, jaki ma zostać nalany. Dostępne wartości dotyczące ilości płynu zdefiniowane są w pamięci programu. Czerwony przycisk służy do rozpoczęcia procesu nalewania. Akcje wykonywane są w momencie, gdy przycisk zostanie zwolniony - "on key up".

3.9 Sterowanie pompą

Układ elektroniczny został podzielony na dwie części: jedna jest powiązana elektrycznie ze sterownikiem i jej zadaniem jest sterowanie diodą LED w transoptorze, druga część obejmuje układ dołączony do tranzystora.

Pompa zasilana jest z zewnętrznego układu zasilania, odseparowanego galwanicznie od mikrokontrolera. Włączanie oraz wyłączanie pompy obsługiwane jest za pomocą pinu cyfrowego z PWM[1] podającego zasilanie na transoptor, który steruje bramką / załączeniem tranzystora MOSFET odpowiedzialnego za pracę pompy. Ze względu na indukcyjny charakter obciążenia, równolegle do odbiornika, została dodana dioda zabezpieczająca i chroniąca tranzystor przed zniszczeniem od napięcia samoindukcji, które obciążenie wygeneruje podczas wyłączania.

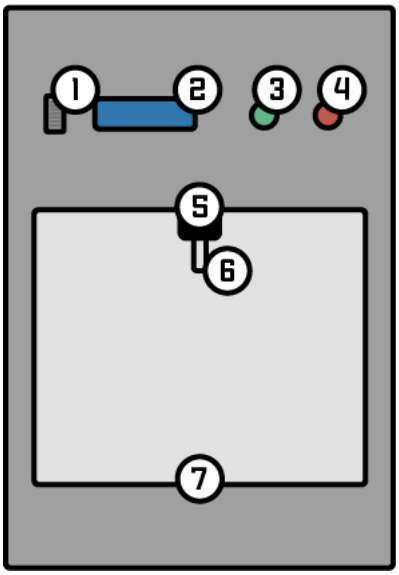
4 Podział obowiązków

	Konrad Jachimstal	Patryk Janicki	Dobromir Kata
Dokumentacja	•	•	•
Pompa, transoptor		•	•
Wyświetlacze, rejestr	•	•	
Czujnik ultradźwiękowy	•	•	•
Czujnik nacisku	•	•	•
Czujnik temperatury	•	•	•
Dioda LED	•	•	•
Przyciski	•	•	•
Sygnały dźwiękowe		•	
Przewody	•		•
Zaangażowanie	80%	90%	80%

[1] PWM - https://enterius.eu/wsparcie/artykuly-techniczne/modulacja-pwm-co-to-jest

5 Instrukacja użytkowania

5.1 Schemat



1. Poziom płynu w zbiorniku (0 - 10)
2. Główny wyświetlacz
3. Przycisk zmiany programu
4. Przycisk akcji (nalewania)
5. Dioda sygnalizująca stan
6. Kran do dystrybucji płynu
7. Czujnik wykrywający naczynie

5.2 Podłączenie

W urządzeniu znajdują się dwa gniazda, USB oraz gniazdo DC. Aby urządzenie działało poprawnie należy zasilić oba gniazda. Po podłączeniu gniazda USB do prądu na ekranie LCD pojawi się ekran główny.

5.3 Wybór ilości płynu

W celu zmiany domyślnej wartości ilości płynu ($40ml$) należy raz wcisnąć zielony przycisk (3), co będzie skutkowało pojawieniem się ekranu wyboru ilości płynu na wyświetlaczu (2). Aby zmienić tę wartość należy wcisnąć zielony przycisk (3), co spowoduje przełączenie wartości na ekranie (2). Dostępne wartości to: $40ml$, $100ml$, $150ml$, $200ml$, $250ml$, $330ml$. Aby zapisać wybraną ilość płynu należy odczekać $4s$, wtedy usłyszymy dzwięk a na ekranie głównym (2) pojawi się wybrana przez nas ilość płynu.

5.4 Nalewanie

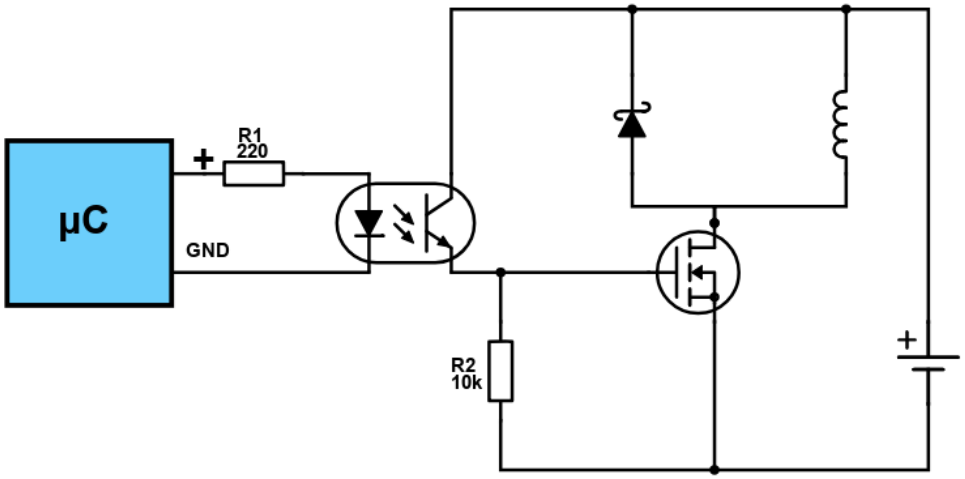
Przed rozpoczęciem nalewania należy ustawić naczynie w wyznaczonym miejscu (7). Poprawne ustawienie naczynia zostanie zasygnalizowane zapaleniem się diody (5) oraz sygnałem dźwiękowym. Nalewanie z kranu (6) rozpocznie się po naciśnięciu czerwonego przycisku (4). Podczas nalewania dioda (5) zacznie migać oraz na ekranie (2) pojawi się odpowiadni komunikat informujący o procesie nalewania. Jeżeli podczas nalewania naczynie zostanie usunięte, proces zostanie przerwany. Zakończenie nalewania zasygnalizowane zostanie dźwiękiem, zgaśnięciem diody (5) oraz odpowiednim komunikatem na ekranie głównym (2).

5.5 Poziom płynu w zbiorniku

Poziom płynu odzwierciedlony jest na wyświetlaczu (1) znajdującym się po lewej stronie panelu przedniego urządzenia. W przypadku kiedy poziom płynu będzie niewystarczający nalewanie nie zostanie rozpoczęte a na ekranie (2) pojawi się stosowny komunikat. W przypadku gdy podczas nalewania poziom płynu w zbiorniku spadnie poniżej minimum, nalewanie zostanie przerwane.

6 Schemat układu sterowania pompą

Podczas konstrukcji urządzenia zaszła potrzeba separacji galwanicznej dwóch bloków układu (platformy oraz pompy).



7 Wykorzystane biblioteki

7.1 NewPing

Biblioteka[2] wykorzystywana do pomiaru odległości z wykorzystaniem czujnika ultradźwiękowego SR04T. Do pomiaru odległości wykorzystywane są dwa piny (TRIGGER i ECHO), obsługiwane w bibliotece za pomocą Port Registers[3]. W celu dokonania pomiaru na pinie TRIGGER wyzwalany jest sygnał wysoki na $10\mu s$, co powoduje wysłanie przez sensor 8 impulsów w częstotliwości $40kHz$. Detekcja ultradźwięków realizowana jest na pinie ECHO w trybie odczytu. Długość trwania stanu wysokiego na tym pinie jest proporcjonalna do odległości.

7.2 ShiftRegister74HC595

Biblioteka[4] usprawnia obsługę rejestrów przesuwnych, dodatkowo obsługuje łączenie rejestrów. W tym przypadku połączono dwa 8-bitowe rejestry przesuwne 74HC595, które pozwalają na obsługę 10 segmentów. Biblioteka wykorzystuje piny DATA (dane), CLOCK (przesunięcie) oraz LATCH (odblokowanie). W bibliotece wykorzystano funkcję shiftOut z Arduino, która zarządza przekazaną zmienną i przekazuje dane na pin DATA, sterując jednocześnie pinem CLOCK.

7.3 LiquidCrystal_I2C

Biblioteka[5] wykorzystuje interfejs I2C do komunikacji z konwerterem wyświetlacza. Komunikacja odbywa się przy użyciu biblioteki Wire, która wysyła sekewncje bitów do urządzeń na magistrali I2C. Biblioteka odwołuje się do konwertera, zlokalizowanego pod adresem $0x27$, natomiast wykorzystanie adresów wewnętrznych konwertera pozwala na wykonywanie akcji związanych z obsługą wyświetlacza.

7.4 DS18B20

Sterowanie czujnikiem temeratury DS18B20 odbywa się za pomocą biblioteki DS18B20[6], który komunikuje się za pomocą interfejsu 1-Wire (OneWire)[7]. Do jednej lini danych możemy podłączyć wiele urządzeń, które posiadają

[2] NewPing - https://bitbucket.org/teckel12/arduino-new-ping
[3] Port Registers - https://www.arduino.cc/en/Reference/PortManipulation
[4] ShiftRegister74HC595 - https://github.com/Simsso/ShiftRegister74HC595
[5] LiquidCrystal_I2C - https://github.com/fdebrabander/Arduino-LiquidCrystal-I2C-library
[6] DS18B20 - https://github.com/matmunk/DS18B20
[7] OneWire - https://pl.wikipedia.org/wiki/1-Wire

swój unikatowy 64-bitowy adres. Przesyłanie rozpoczyna się sekwencją bitów, wysłaniem impulsu reset (zwarciu na 480µs do masy), a następnie potwierdzenie obecności przez każde urządzenie poprzez zwracie linii danych do masy na 60µs. Logiczna jedynka to krótki (1-15µs) impuls o stanie niskim, następnie stan wysoki o długości 60µs. Opadające zbocze impulsu aktywuje przerzutnik, który taktuje wewnętrzny mikroprocesor, co powoduje odczyt danych z lini po ok. 30µs od momentu pojawienia sie zbocza narastającego. Ze względu na wewnętrzne opóźnienia urządzenia czas trwania pojedyńczego sygnału musi wynosić 60µs. Po przesłaniu 8 bitów następuje wysłanie komendy (rozkazu, ośmiobitowego). Ewentualne błędy w transmisji mogą być wykryte za pomocą wbudowanego algorytmu CRC-8[8]. Biblioteka wykorzystuje polecenia 0xF0 oraz 0x33 do wykrywania urządzeń podłączonych do magistrali OneWire.

[8] CRC-8 - https://pl.wikipedia.org/wiki/Cyklicznykodnadmiarowy

8 Opis algorytmu

Program został stworzony w podstawowym IDE dla Arduino, umieszczony w projekcie o nazwie *fluid – dispenser.ino*. W celu zwiększenia czytelności kod został podzielony na sekcje.

8.1 Sekcje deklaracji stałych

W tej sekcji zadeklarowano piny odpowiadające poszczególnym funkcjonalnościom (IN/OUT NUMBERS), indeksy poszczególnych zadań (TASK NAMES - więcej informacji w punkcie "8.6 Sekcja zadań") oraz inne wartości wykorzystywane w programie (CONFIGURATION) takie jak czas detekcji naczynia, czasy wygasania ekranów oraz dostępne pojemności.

```
// ================== IN/OUT NUMBERS ==================
// Buttons
#define TOGGLE_BUTTON_PIN 4  // IN
#define BUZZER_PIN 5 // OUT
...
// ==================== TASK NAMES ====================
#define MEASURE_DISTANCE_TASK 0
#define TOGGLE_BUTTON_TASK 1
...
// ================== CONFIGURATION ==================
#define CUP_DETECTION_TIME 1000
#define BUTTONS_DEBOUNCE_TIME 200
...
```

8.2 Sekcja deklaracji zmiennych globalnych

Sekcja ta zapisuje dane ulegające zmianie podczas działania programu np. takie jak: czas trwania programu, ton brzmienia buzzera, czas trwania dzwięku, stan diody czy stan przycisków.

```
// ==================== VARIABLES ====================
// Multitasking millis variables
unsigned long currentMillis = 0;
unsigned long prevMillis[11];
// Button states
boolean toggleButtonPressed = false;
...
```

8.3 Sekcja instancji klas bibliotek

W celu przyspieszenia pracy i zmniejszenia ilości kodu użyto zewnętrznych bibliotek przeznaczonych do obsługi wykorzystanych w urządzeniu elementów. W tej sekcji zadeklarowano instancje klas bibliotek dla:

- czujnika ultradźwiękowego;
- rejestru przesuwnego;
- wyświetlacza LCD;
- czujnika temperatury;

```
// ==================== INSTANCES ====================
NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE);
ShiftRegister74HC595 sr(2, SHIFTREG_SERIAL_DATA_PIN, SHIFTREG_...);
LiquidCrystal_I2C lcd(0x27, 16, 2);
DS18B20 ds(TEMPERATURE_SENSOR_PIN);
```

8.4 Sekcja inicjalizacji

Inicjalizacja programu rozpoczyna się ustawieniem pinów na wyjście lub wejście w zależności od pełnionej funkcji. Kolejno inicjowany jest wyświetlacz oraz na jego ekranie wyświetlany jest ekran główny (informacja o temperaturze oraz wybrana pojemność). Szczegóły implementacji wyświetlania ekranów opisane zostały w sekcji 8.8.

```
// ====================== SETUP ======================
void setup() {
  pinMode(TOGGLE_BUTTON_PIN, INPUT);
  pinMode(ACTION_BUTTON_PIN, INPUT);
  pinMode(BUZZER_PIN, OUTPUT);
  pinMode(PUMP_PIN, OUTPUT);
  pinMode(DIODE_PIN, OUTPUT);

  lcd.init();
  lcd.backlight();
  displayHomeScreen();
}
```

8.5 Sekcja pętli programu

W głównej pętli programu zapisywane jest aktualny czas od rozpoczęcia pracy. Jest on wykorzystywany do sterowania czasem poszczególnych zadań, ze względu na brak wykorzystania opóźnień (*delay*). Następnie sprawdzane są warunki zajścia poszczególnych akcji (szczegóły obsługi zadań w sekcji 8.6). Dodatkowo w pętli umieszczono również przechwytywanie stanu dźwięków oraz diod (szczegóły w sekcji 8.6).

```
// ======================= LOOP =======================
void loop() {
  currentMillis = millis();

  if (canPerformTask(MEASURE_DISTANCE_TASK, 100)) {
    performMeasureDistanceTask();
  }

  ...

// Handle sounds
  handleSoundsTask();

// Handle diodes
  handleDiodesTask();

// Set pump state
  digitalWrite(PUMP_PIN, isPumpActive);
}
```

8.6 Sekcja zadań

W celu zwiększenia czytelności kodu oraz uogólnienia czasu odstępu pomiędzy zadaniami, wykorzystano tablicę `prevMillis[]` oraz funkcję `canPerformTask()`. Jako indeksy tablicy użyte zostały stałe. W elementach tablicy przechowywany jest czas ostatniego wykonania zadania. Szczegóły implementacji funkcji w sekcji 8.9.

```
// ======================= TASKS =======================
// ---------------------- TOO: Measure distance task
void performMeasureDistanceTask() {
  int distance = sonar.ping_cm();
```

```
  if (distance > 30) return;
  ...
}

// ----------------------- T01: Toggle button task
void perfomToggleButtonTask() {
  if (isPumpActive) {
    return;
  }
  ...
}

// ----------------------- T02: Action button task
void perfomActionButtonTask() {
  if (isCupPresent() && !isVesselAlreadyFilled) {
    pumpStartTime = currentMillis;
    isPumpActive = true;
    isDiodeBlinking = true;
  }
  ...
}

// ----------------------- T03: Expire toggle task
void performExpireToggleTask() {
  isToggleActivated = false;
  ...
}

// ----------------------- T04: Expire no vessel task
void performExpireNoVesselTask() {
  noVesselActivated = false;
  displayHomeScreen();
  ...
}
```

8.7 Sekcja dźwięków

Operacje na dźwiękach wykonywane są za pomocą podania napięcia na wyjście pinu w odpowiednich sekwencjach oraz odstępach czasowych. Manipulując napięciem uzyskujemy odpowiedni ton dzwięku wydobywający się z podłączonego buzzera zawierającego generator. W sekcji kodu zawarto zmienne pozwalające obliczyć czas, który upłynął od zmiany wartości dźwięku.

```
// ----------------------- S00: Beep
boolean beepSoundActive = false;
unsigned long beepSoundStartTime = 0;
void handleBeepSound() {
  ...
  int buzzerValue = beepSoundActive ? beepTone : 0;
  analogWrite(BUZZER_PIN, buzzerValue);
}

// ----------------------- S01: Negative
boolean negativeSoundActive = false;
unsigned long negativeSoundStageTime = 0;
unsigned int negativeSoundStage = 0;
boolean negativeSoundStageSetted = false;
void handleNegativeSound() {
  ...
}
```

8.8 Sekcja ekranów

Każdy ze stanów pracy urządzenia posiada ekran informujący użytkownika o wykonywaniu akcji lub zmianie ustawień. Dodatkowo podczas przełączania zawartości wyświetlacza ustawiane są zmienne globalne (`isHomeScreenActive`, `isDoneScreenActive`, ...), które informują o tym, który ekran jest aktualnie wyświetlony. Zmienne te pozwalają określić, czy inny ekran może zostać przełączony w danym momencie.

```
// ----------------------- SC00: Home
void displayHomeScreen() {
  isHomeScreenActive = true;
  lcd.clear();
  lcd.print("TEMP:     " + String(temperature, 1) + "C");
  lcd.setCursor(0, 1);
```

```
  lcd.print("SELECTED: " + volumeText(VOLUMES[toggleStage]));
}
...
```

8.9 Sekcja funkcji

8.9.1 canPerformTask()

Funkcja `canPerformTask()` zwraca prawdę, jeśli zadanie może zostać ponownie wykonane, jednocześnie aktualizując czas wykonania zadania. Wykorzystywana jest ona przy wykonywaniu poszczególnych zadań.

```
// Alternative for delay() function
boolean canPerformTask(int index, unsigned long ms) {
// Check if task will should be performed
  if (abs(currentMillis - prevMillis[index]) >= ms) {
// Update last perform time
    updateTaskTime(index);
    return true;
  }
  return false;
}
```

8.9.2 updateTaskTime()

Funkcja aktualizuje czas zadania o podanym indeksie.

```
// Update prev time
void updateTaskTime(int index) {
  prevMillis[index] = currentMillis;
}
```

8.9.3 isCupPresent()

Funkcja pozwala na sprawdzenie, czy naczynie zostało wykryte. Liczba prawidłowych pomiarów czujnika ultradźwiękowego jest zliczane poprzez zmienną globalną `echoMeasurementsCount` w zadaniu $MEASURE\_DISTANCE\_TASK$. W przypadku gdy naczynie zostanie postawione na czujniku, oczekiwane są nieprawidłowe pomiary (brak pomiaru). Jeśli liczba prawidłowych pomiarów jest równa 0, a czas od ostatniego pomiaru jest większy od $CUP\_DETECTION\_TIME$, to wówczas uznajemy, że naczynie zostało wykryte.

```
// Check if ultrasound sensor is covered
boolean isCupPresent() {
  return echoMeasurementsCount == 0 &&
         currentMillis - echoMeasurementTime >= CUP_DETECTION_TIME;
}
...
```

8.10 Podsumowanie

W powyższym przedstawieniu zostały zaprezentowane jedynie najważniejsze, elementarne fragmenty kodu, mające istotny wpływ na strukturę i działanie programu. W celu zapoznania się z całym kodem, należy zasięgnąć źródła dostępnego w repozytorium[9].

9 Analiza rodzajów i skutków możliwych błędów

9.1 Awaria wyświetlacza znakowego i segmentowego

Ze względu na to, że wyświetlacz jest urządzeniem końcowym i nie pobierami z niego żadnych danych, nie jesteśmy w stanie wykryć jego awarii.

9.2 Awaria pompy

Pompa działa jako samodzielne urządzenie przez co nie możemy wykryć jej awarii.

9.3 Awaria czujników

Wykorzystanie czujniki pozwalają na autodetekcję niesprawności ze względu na to, że podczas działania programu pobierane są z nich dane. Podczas gdy dane pobrane z czujnika będą wykraczały po za wartości regularne lub czujnik nie będzie wysyłał odczytów, możemy stwierdzić, że dany czujnik uległ uszkodzeniu.

[9] Repozytorium - https://github.com/janicky/fluid-dispenser

9.4 Awaria diody, przycisków, buzzera

Spalenie diody, przycisków bądź buzzera nie jest możliwe do zdiagnozowania przez system, ponieważ jest urządzeniem końcowym. Uszkodzenie diody oraz buzzera nie przerywa pracy urządzenia. W przypadku uszkodzenia przycisków dalsze użytkowanie urządzenia jest niemożliwe.

Ryzyko	**Znaczenie**	**Reakcja**	**Skutki**
Uszkodzenie wyświetlacza	Średnie	Brak	Brak informacji o stanie płynu w zbiorniku. Brak informacji o temperaturze płynu. Brak możliwości ustawienia ilości płynu do nalania.
Uszkodzenie diody lub buzzera	Nieznaczące	Brak	Brak komunikatów o ukończeniu zadania, zapisaniu ustawienia. Brak komunikatu błędu.
Uszkodzenie przycisku	Krytyczne	Brak	Brak możliwości rozpoczęcia nalewania lub zmiany ilości płynu.
Uszkodzenie czujnika ultradźwiękowego	Nieznaczące	Umożliwienie nalewania	Możliwość rozpoczęcia nalewania bez wykrycia naczynia.
Uszkodzenie czujnika nacisku	Nieznaczące	Sygnalizacja na wyśw.	Brak informacji o stanie płynu w zbiorniku. Możliwe dalsze korzystanie z urządzenia.
Uszkodzenie czujnika temperatury	Nieznaczące	Komunikat błędu	Brak informacji o temperaturze płynu.
Uszkodzenie pompy	Krytyczne	Brak	Brak możliwości rozpoczęcia nalewania.

10 Bibliografia

[1] MOSFET jako przełącznik,
`https://www.electronics-tutorials.ws/pl/tranzystor/mosfet-jako-przelacznik.html`

[2] Tranzystory mocy w układach elektroniki mocy,
`https://elektronikab2b.pl/technika/36255-tranzystory-mocy-w-ukladach-elektroniki-mocy-i-ich-sterowanie-cz.-1`

[3] Wykorzystanie funkcji shiftOut w rejestrze przesuwnym
`https://www.arduino.cc/reference/en/language/functions/advanced-io/shiftout/`

[4] Failure mode and effects analysis,
`https://pl.wikipedia.org/wiki/FMEA`