Układ stabilizujący temperaturę

Systemy wbudowane - projekt

Piotr Żeberek	407663	gr. 2
Maciej Müller	418133	gr. 2

Spis treści

1 Cel Projektu **2**

2 Wykorzystane Elementy i Sposób Ich Zestawienia **2**

3 Interfejs Użytkowania **4**
3.1 Opis interfejsu na ekranie 4
3.2 Obsługa aplikacji 4

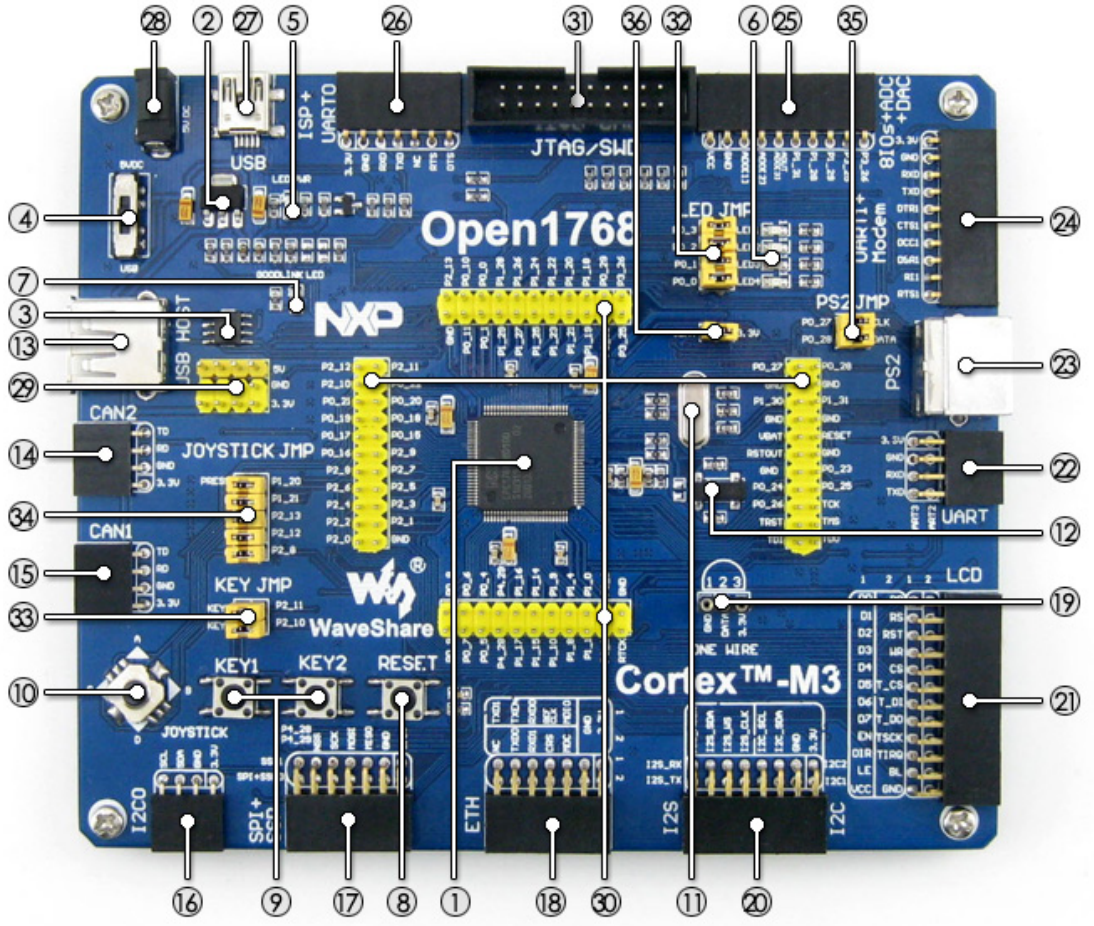
4 Algorytmy Sterowania Grzałką **4**
4.1 Wstępny Opis Algorytmów 4
4.2 Dwustopniowy 5
4.3 PID 5
4.4 Porównanie sterowania dwustopniowego i PID 6

5 Implementacja **7**
5.1 main.c 7
5.2 timer.c, timer.h 7
5.3 UART.c, UART.h 7
5.4 I2C_TMP2.c, I2C_TMP2.h 8
5.5 PWM.c, PWM.h 8
5.6 tempRegulation.c tempRegulation.h 8
5.7 LCD_screen.c, LCD_screen.h 8
5.8 Zewnętrzne pliki i biblioteki 9

1 Cel Projektu

Celem projektu była realizacja urządzenia utrzymującego zadaną temperaturę, w pudełku bądź innej zamkniętej przestrzeni. Na podstawie odczytu z termometru program dostosowuje moc grzałki (wypełnienie przebiegu PWM) używając jednego z dwóch trybów sterowania: dwustopniowego lub PID opisanych w sekcji 4. Komunikacja z termometrem odbywa się poprzez protokół I2C a pomiary i logi aplikacji przesyłane są na podłączone urządzenie (w naszym przypadku komputer PC) przez UART.

2 Wykorzystane Elementy i Sposób Ich Zestawienia

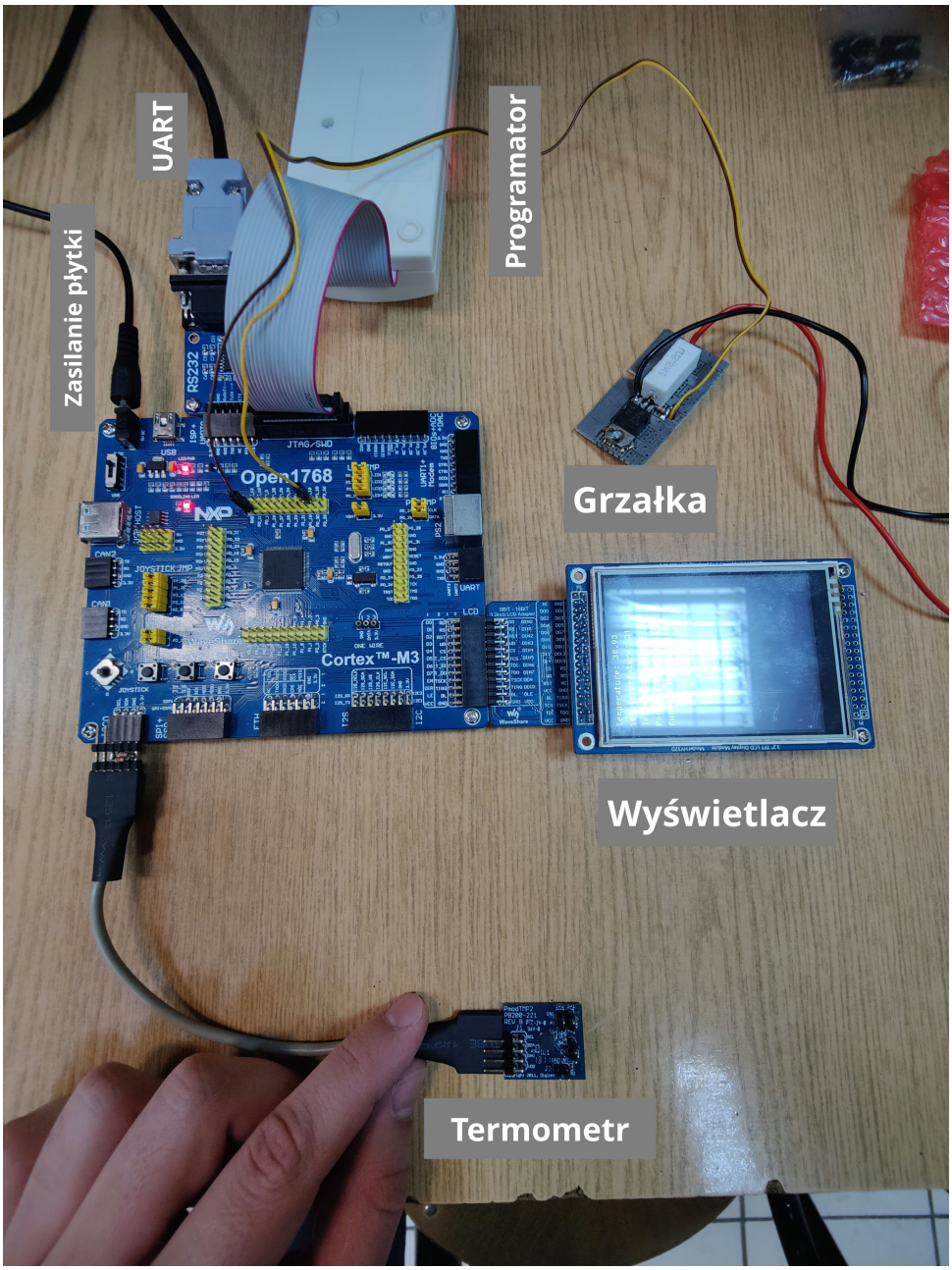


Rysunek 1: Schemat płytki LPC1768 z opisanymi złączami i elementami.

Użyte komponenty:

- płytka LPC1768 z mikrokontrolerem ARM Cortex M3, podłączona do zasilania 3.3V na (28),
- przyciski KEY1 oraz KEY2, wbudowane w płytkę LPC1768, służące do sterowania,
- wyświetlacz LCD podłączony na (21), korzystający ze sterownika ILI9325,
- czujnik temperatury (Pmod TMP2 16-bitowy), korzystający z protokołu I2C, podłączony na (16), gdzie żaden z jumper bloków nie został złączony, dlatego adres urządzenia powinien być ustawiony na 0x4B,

- złączka z pull-up'em, czyli dwa rezystory przylutowane tak, aby prowadziły z SCL i SDA na linię zasilania VCC 3.3V,
- grzałka zbudowana z tranzystorów i rezystorów, podłączona na (30) pod pin P1_18 oraz GND z zewnętrznym zasilaniem ustawionym na 6V,
- zewnętrzne urządzenie (PC) do podglądu komunikatów, połączone na (26), za pomocą UART z 9600 bodami wysyłającym 8-bitowe znaki.



Rysunek 2: Obrazek ukazujący, fizyczne podłączenia wszystkich komponentów.

3 Interfejs Użytkowania

3.1 Opis interfejsu na ekranie

W pierwszym wierszu, wyświetlana jest temperatura ostatnio pobrana z czujnika. Jeśli czujnik został odłączony, wtedy wyświetlany jest napis "Error" i tym samym, grzałka przestaje grzać.
W następnym wierszu, wyświetlana jest różnica temperatur, pomiędzy ustaloną temperaturą, a ostatnio odczytaną.
Niżej pokazana jest, aktualna moc grzałki, czyli procent wypełnienia okresu PWM trwającego 1 sekundę, przez wysoki sygnał.
Na końcu wyświetlany jest napis, informujący użytkownika o aktualnie wybranym przez niego trybie sterowania grzałką.
Poniżej tych informacji, może się znaleźć również informacja o błędzie, jeśli odłączymy sensor temperatury.

3.2 Obsługa aplikacji

Obsługa aplikacji jest bardzo prosta. Użytkownik jest wstanie sterować aplikacją, posługując się przyciskami KEY1 i KEY2 umieszczonymi na płycie głównej.
Przycisk KEY1 (patrz rysunek 1, ⑨ po lewej stronie) zwiększa temperaturę, którą chcemy utrzymać na grzałce o 1 °C.
Natomiast **przycisk KEY2** (patrz rysunek 1, ⑨ po prawej stronie) zmniejsza temperaturę, którą chcemy utrzymać na grzałce o 1 °C.
Naciśnięcie obu klawiszy **KEY1 i KEY2** jednocześnie i przytrzymanie ich przez około 2,5 sekundy, zmienia tryb sterowania grzałką.

4 Algorytmy Sterowania Grzałką

4.1 Wstępny Opis Algorytmów

Co sekundę, czytywana jest aktualna temperatura z czujnika temperatury, a następnie wywoływana jest jedna z funkcji regulacji grzałki, wybrana przez użytkownika.

- **readTemp** - ostatnio odczytana temperatura z termometru,
- **setTemp** - ustawiona temperatura przez użytkownika, która powinna być utrzymywana,
- **tempErrorSum** - suma błędów temperatur, służąca do obliczenia całki (części Integral) w PID,
- **deltaTime** - czas odstępu pomiędzy odczytywaniem temperatury, w naszym przypadku jest to 1 sekunda,
- **P_Amp, I_Amp, D_Amp** - stałe wzmocnienia, każdej z części kontrolera PID, odpowiednio nastrojone,
- **Set_DutyCycle()** - to funkcja, która zmienia długość okresu działania PWM w procentach (od 0 do 100).

4.2 Dwustopniowy

W tym sposobie mamy tylko dwa tryby:

- pełna moc grzałki, jeśli odczytana temperatura jest mniejsza od zadanej,
- zerowa moc grzałki, jeśli odczytana temperatura jest większa od zadanej.

Jest to jeden z najprostszych sposobów sterowania.

Algorithm 1 Sterowanie Dwupołożeniowe

Require: $readTemp, setTemp$

function TWO-POSITIONAL-CONTROL
  if $readTemp > setTemp$ **then**
    $heaterPower \leftarrow 0$
    $PWM\_SetDutyCycle(0)$
  else
    $heaterPower \leftarrow 100$
    $PWM\_SetDutyCycle(100)$
  end if
end function

4.3 PID

W tym sposobie wypełnienie przebiegu jest obliczane na podstawie trzech składowych:

- P - proporcjonalny (ang. *proportional*) - uwzględnia aktualny błąd temperatury,
- I - całkujący (ang. *integral*) - uwzględnia sumę błędów w poprzednich krokach,
- D - różniczkujący (ang. *derivative*) - próbuje uwzględnić przyszłe błędy poprzez pochodną.

Składowe te są ważone przez ustalone wzmocnienia P_Amp, I_Amp, D_Amp i sumowane, aby uzyskać nowe wypełnienie przebiegu sterujące grzałką.

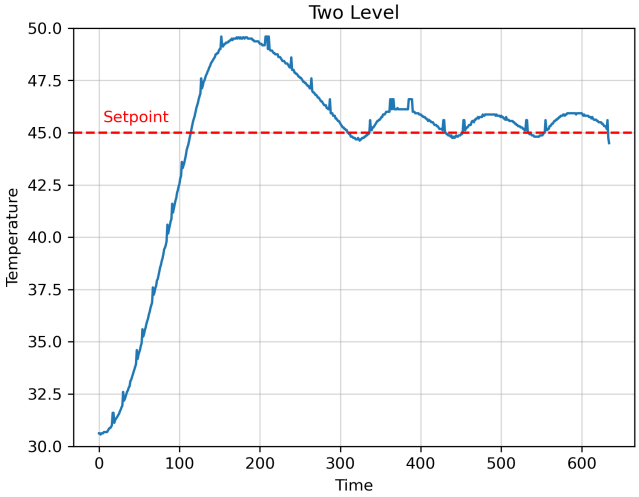
Algorithm 2 Sterowanie PID

Require: $readTemp, setTemp, tempErrorSum, deltaTime, P\_Amp, I\_Amp, D\_Amp$

function PID-CONTROL
  $output \leftarrow 0$
  $tempErrorPrev \leftarrow tempError$
  $tempError \leftarrow setTemp - readTemp$
  if $tempError < 16$ **then**
    $tempErrorSum = tempErrorSum + tempError * deltaTime$
  end if
  $output \leftarrow output + P\_Amp \cdot tempError$
  $output \leftarrow output + I\_Amp \cdot tempErrorSum$
  $output \leftarrow output + D\_Amp \cdot \frac{(tempError - tempErrorPrev)}{deltaTime}$
  $heaterPower \leftarrow output$
  return $output$
end function

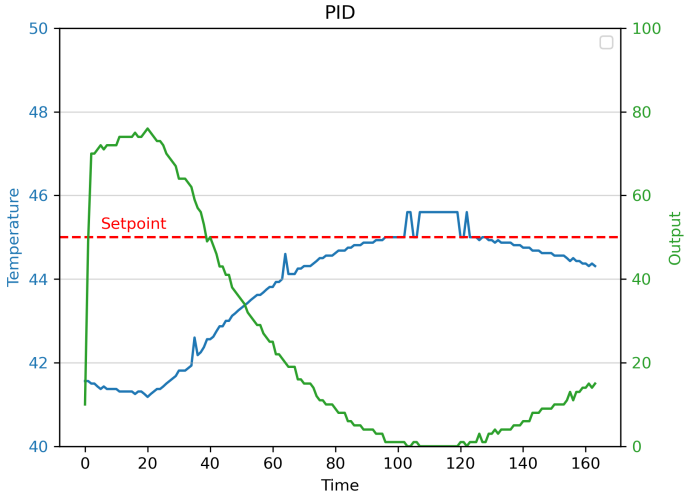
4.4 Porównanie sterowania dwustopniowego i PID

Termometr wraz z grzałką zamknięto w kartonowym pudełku wraz z wątpliwą izolacją z pianki tworząc układ zamknięty i ustawiono regulator na 45 °C. Zadaną temperaturę próbowano ustalić opisanymi wcześniej sposobami. W pomiarach temperatury występują drobne, nagłe skoki, które mogą być wynikiem chociażby ruszania pudełkiem, w którym przeprowadzany był pomiar.



Rysunek 3: Zmiany temperatury w czasie przy stabilizacji sterowaniem dwustopniowym.

Przy sterowaniu dwustopniowym obserwujemy charakterystyczne przestrzelenia wynikające z 'bezwładności' układu. Program włącza i wyłącza grzałkę w zależności od temperatury stricte termometru, a nie układu. Po wyłączeniu grzałki i rozejściu się gradientu temperatury po pudełku końcowa temperatura jest średnio wyższa od zadanej.



Rysunek 4: Zmiany temperatury oraz % wypełnienia przebiegu (tutaj jako 'output') przy stabilizacji sterowaniem PID.

Użyte parametry PID:

- P_Amp=20.0
- I_Amp=0.01

- D_Amp=0.5

Zabrany przebieg niestety jest skromniejszy niż w przypadku sterowania dwustopniowego z uwagi na pośpiech na ostatnich zajęciach, ale nadal możemy zauważyć kilka ciekawych aspektów sterowania PID.

- W początkowych chwilach zielona krzywa (wypełnienie przebiegu) potrzebuje kilku pomiarów, aby człony związane z całkowaniem i różniczkowaniem mogły być poprawnie obliczone. Stąd też jej dziwny przebieg.
- Z dokładnością do szumu w pomiarach temperatura nie przekroczyła zadanej jak miało to miejsce w sterowaniu dwustopniowym.
- Po osiągnięciu zadanej temperatury zaczęła ona znowu spadać, a wypełnienie przebiegu rosnąć. O ile nie jesteśmy w stanie uniknąć takich oscylacji to ideą sterowanie PID jest taki dobór parametrów, aby oscylacje te zminimalizować.

5 Implementacja

5.1 main.c

Główny plik aplikacji, w którym zaimplementowana jest struktura działania programu. Na początku, inicjalizuje on wszystkie potrzebne komponenty i wyświetla stałe znaki na wyświetlaczu, po czym przechodzi do głównej pętli programu. W tej pętli, uruchamiane są polecenia, jeśli odpowiednie flagi timera są aktywne, a które są odpowiednio ustawiane w plikach timer.c/h. Każda flaga po aktywacji i wykonaniu interesujących nas instrukcji, jest resetowana, aby mogła znowu zostać aktywowana.
W pliku tym, znajduje się także funkcja **updateButtonsStatus()**, obsługująca przyciski, wywoływana jest co 50ms, a która sprawdza stan aktualnie naciśniętego przycisku. Jeśli któryś z klawiszy zostanie naciśnięty to jego stan zostaje zmieniony na 1 w strukturze **buttonsStatus**, dzięki czemu w pętli głównej mogą one zostać obsłużone.
Wyjątkiem jest naciśnięcie obu przycisków, które zwiększa wartość **buttonsStatus.BOTHKEYS**. Jeśli przyciski będą te trzymane wystarczająco długo i wartość **buttonsStatus.BOTHKEYS** osiągnie odpowiednio wysoką wartość (**CONTROL_SWITCH_THRESHOLD**), to w pętli głównej zmieniany zostaje tryb kontroli grzałki.

5.2 timer.c, timer.h

W plikach tych, znajduje się struktura **timerStatus**, która zawiera flagi przedziałów czasu. A także zaimplementowana jest funkcja inicjalizująca TIMER0 mikrokontrolera, który zwiększa się o 1, co jedną milisekundę i wywołuje funkcję **TIMER0_IRQHandler**. To właśnie ona, kontroluje flagi w uprzednio wspomnianej strukturze. Robi to, za pomocą zmiennej **tick**, która zwiększana jest o jeden co milisekundę, a następnie ze względu na jej wartość, aktywowane są odpowiednie flagi przedziałów czasu.

5.3 UART.c, UART.h

Pliki konfigurujące protokół UART na pinach P0.2, P0.3 z 8-bitową długością słowa i 9600 bodach. Zawierają one funkcje inicjalizujące UART, a także funkcje, które umożliwiają na wysłanie ciagu znaków, liczb całkowitych oraz liczb zmienno-przecinkowych. UART służy głównie do wysyłania informacji w celu debugowania programu, wystąpienia błędów i innych komunikatów informujących o poprawności działania programu.

5.4 I2C_TMP2.c, I2C_TMP2.h

W plikach tych, korzystamy z biblioteki do obsługi protokołu I^2C, zajmują się one połączeniem z sensorem temperatury, odczytywaniem i konwertowaniem otrzymanej wartości temperatury.
Funkcja **I2C_SignalEvent()** wychwytuje zdarzenia protokołu i aktywuje flagę błędu I^2C, jeśli nie jesteśmy w stanie otrzymać temperatury z czujnika.
TMP2_Initialize() oraz **TMP2_Read_Event()** służą do połączenia z sensorem, wysyłając sygnał restartu (czyli 2 sygnały staratu, bez stopu pomiędzy nimi).
convertTemperature(), konwertuje wartość otrzymaną z czujnika na stopnie Celsjusza i zapisuje je do zmiennej globalnej **currentTemperature**.
Funkcja **readTemperature()** czyta aktualną temperaturę z bufora termometru, po czym wywołuje **convertTemperature()**.

5.5 PWM.c, PWM.h

PWM służący do sterowania grzałką, został skonfigurowany na porcie P1.18, z długością cyklu ustawioną na jedną sekundę, konfigurację tą ustawia funkcja **PWM_Init()**. Następnie do sterowania wypełnieniem cyklu długości PWM, wysokim sygnałem, stworzona została funkcja **PMW_SetDutyCycle()**, która jako argument przyjmuję liczbę w zakresie 0 do 100 jako procent wypełnienia cyklu.

5.6 tempRegulation.c tempRegulation.h

Pliki te zawierają funkcję, koordynujące czasem działania grzałki i których działanie zostało wytłumaczone za pomocą pseudokodu w sekcji 4. Zawierają one również zmienne globalne, które są wyświetlane ekran, lub potrzebne do obliczeń.
Funkcja **calculatePID()** oblicza czas działania grzałki, w ten sam sposób co regulatory PID tzn. że obliczane są 3 człony. Człon P (Proportional), który kompensuje uchyb bieżący, obliczany jest poprzez pomnożenie stałej wzmocnienia proporcjonalnego **Amplification_P** i różnicy temperatur pomiędzy odczytaną temperaturą a zadaną temperaturą do utrzymania:
$Amplification\_P \cdot temperatureError$
Człon I (Integral) kompensujący akumulację uchybów z przeszłości, obliczany jako suma różnic temperatur, pomnożona przez stałe wzmocnienie różniczkowe **Amplification_I**:
$Amplification\_I \cdot sumTemperatureError$
Człon D (Derivative) kumulujący przewidywane uchyby w przeszłości, wyliczany z pomnożenia stałej wzmocnienia pochodnej **Amplification_D** razy różnicy, różnicy temperatur aktualnej i poprzedniej, podzielone przez odstęp czasu pomiędzy nimi:
$Amplification\_D \cdot \frac{temperatureError - temperatureErrorPrev}{deltaTime}$
Funkcja **twoPositionalControl()**, sterująca grzałką w bardziej prymitywny sposób, a mianowicie, wyłącza ona grzałkę jeśli temperatura odczytana z sensora jest większa od zadanej temperatury do utrzymania, inaczej grzałka cały czas grzeje.
Obie te funkcje, obliczają również moc grzałki.

5.7 LCD_screen.c, LCD_screen.h

W plikach tych umieszczone są funkcję, służące do wyświetlania interfejsu użytkownika na ekran LCD oraz funkcja **initLCDScreen()**, która inicjalizuje LCD oraz sterownik ILI9325.
Funkcja **drawLetter()**, rysuje podany znak do wyświelacza, korzystając ze zdefiniowanej czcionki zawartej w plikach asciiLib.c/h, jako prostokąt składający się w 8x16 pixeli, rozpoczynając od zadanych współrzędnych.
Funkcja **drawString()**, która tak samo jak poprzednia funkcja, służy do wyświetlania tym razem ciągu znaków na ekran.

Funkcja **drawIntNumer()**, wyświetla liczbę całkowitą, ujemną bądz dodatnią, resetując odpowiednią ilość znaków (maksymalnie **MAX_NUMBER_LENGTH**), poprzez wyświetlenie pustego znaku.
Funkcja **drawFloatNumber()**, działająca tak samo jak funkcja drawIntNumber(), tylko wyświetlająca liczbę zmienno-przecinkową.
Poza tymi funkcjami istnieją jeszcze funkcje **SetBackground()**, która zapełnia cały ekran jedynm wybranym kolorem.
Funkcja **drawConstantDataOnScreen()**, wyświetlająca elementy UI, które nie zmieniają się podczas działania aplikacji. A **updateDataOnScreen()**, aktualizuje odpowiednio liczby wyświetlane na UI, które zmieniają się podczas działania programu.

5.8 Zewnętrzne pliki i biblioteki

- **LCD_ILI9325.c, LCD_ILI9325.h, Open1768_LCD.c, Open1768_LCD.h** - odpowiednie pliki sterownika do wyświetlacza LCD oraz jego konfiguracji i inicjalizacji.
- **asciiLib.c, asciiLib.h** - pliki z czcionkami ASCII 16-bitowe, którymi posługujemy się do wyświetlania wybranej czcionki na ekranie LCD.