GDAŃSK UNIVERSITY OF TECHNOLOGY
FACULTY OF ELECTRONICS, TELECOMMUNICATIONS AND INFORMATICS

Student's name and surname: Paweł Bałbatun
ID: 193652
Cycle of studies: bachelor's degree studies
Mode of study: full-time studies
Field of study: Automatic Control, Cybernetics and Robotics
Profile: Automation Systems

ENGINEERING DIPLOMA THESIS

Title of the thesis: Design of a measurement card for a laboratory stand for measuring physical quantities in HVAC systems, based on a selected design kit with a TFT graphic display.

Title of the thesis (in Polish): Projekt stanowiska pomiarowego podstawowych wielkości fizycznych wykorzystywanych w systemach HVAC w oparciu o wybrany zestaw uruchomieniowy wyposażony w wyświetlacz graficzny TFT

Supervisor: mgr inż. Piotr Darski

SPIS TREŚCI

Streszczenie

Praca przedstawia projekt oraz weryfikację stanowiska pomiarowego dla wybranych wielkości fizycznych typowych w systemach HVAC (temperatura, wilgotność, ciśnienie, przepływ oraz sygnały sterujące $0\,\mathrm{V}–10\,\mathrm{V}$). Rozwiązanie bazuje na zestawie uruchomieniowym STM32F746G-Discovery z wyświetlaczem TFT oraz na dedykowanej karcie pomiarowo–wyjściowej $0\,\mathrm{V}–10\,\mathrm{V}$ zaprojektowanej w KiCad. Oprogramowanie wykonano w środowisku Zephyr RTOS z wykorzystaniem sterowników peryferiów i biblioteki LVGL do obsługi interfejsu graficznego. Przedstawiono wymagania funkcjonalne, projekt części analogowej (tory wejściowe z ochroną i skalowaniem, tor wyjściowy $0\,\mathrm{V}–10\,\mathrm{V}$), architekturę oprogramowania (wątki RTOS, kolejki, sterowniki), a także wyniki walidacji dokładności i powtarzalności pomiarów na podstawie wzorców i porównania z przyrządami referencyjnymi.

0.1. Wstęp

Celem pracy jest zaprojektowanie i weryfikacja stanowiska pomiarowego dla podstawowych wielkości fizycznych spotykanych w HVAC oraz interfejsu sterującego $0\,\mathrm{V}–10\,\mathrm{V}$, z wykorzystaniem zestawu STM32F746G-Discovery (STM32F746NG, Cortex-M7) i dedykowanego modułu pomiarowego PCB. Motywacją jest potrzeba ekonomicznego, dydaktycznego stanowiska do testów i demonstracji algorytmów sterowania.

Omówiono kontekst przemysłowy sygnałów $0\,\mathrm{V}–10\,\mathrm{V}$, przegląd czujników oraz wymagania co do dokładności i izolacji torów.

1. STAN WIEDZY I ZAŁOŻENIA PROJEKTOWE

Celem projektu jest przede wszystkim zaprojektowanie i uruchomienie stanowiska laboratoryjnego, a nie opracowanie nowego modelu teoretycznego. Z tego powodu przegląd literatury ma charakter praktyczny i koncentruje się na zagadnieniach, które były rzeczywiście potrzebne podczas projektu: standardzie sygnału 0–10 V w HVAC, podstawowych algorytmach regulacji oraz ogólnych zasadach projektowania płytek PCB dla układów mieszanych analogowo–cyfrowych. AUAUAUAUAUA

1.1. Sygnał 0–10 V i sterowanie HVAC

W automatyce budynkowej jednym z najczęściej spotykanych sygnałów sterujących jest napięcie 0–10 V. W materiałach producentów czujników, przetworników oraz w opracowaniach dotyczących automatyki budynkowej podkreśla się, że sygnał 0–10 V jest powszechnie używany do sterowania siłownikami przepustnic i zaworów, napędami przetwornic częstotliwości oraz różnymi czujnikami stosowanymi w instalacjach HVAC[4, 12, 13]. Główna zaleta takiego sygnału to prostota: praktycznie każdy sterownik PLC lub system BMS potrafi wygenerować albo zmierzyć napięcie 0–10 V, a jego interpretacja jest intuicyjna (np. 0–100 % odpowiada 0–10 V).

W literaturze naukowej dotyczącej sterowania HVAC większość uwagi poświęca się modelom cieplnym i algorytmom regulacji, a warstwa sprzętowa jest zwykle opisywana dość ogólnie. W artykułach dotyczących energooszczędnego sterowania HVAC stosuje się klasyczne sygnały analogowe (napięciowe lub prądowe), ale szczegóły toru 0–10 V są zazwyczaj schowane wewnątrz sterowników[3, 14]. W niniejszej pracy ten tor został celowo „wyciągnięty na wierzch” w postaci osobnej płytki dydaktycznej, tak aby można było go obserwować i modyfikować podczas zajęć laboratoryjnych.

1.2. Regulacja PI/PID i sekwencje pracy central HVAC

W publikacjach dotyczących sterowania HVAC najczęściej stosuje się regulatory PI lub PID. Do ich strojenia wykorzystuje się zarówno proste metody (np. Ziegler–Nichols), jak i bardziej zaawansowane algorytmy optymalizacyjne[3, 11, 18]. Przykładowo Almabrok i in. prezentują szybką metodę strojenia regulatora PID dla systemu HVAC z użyciem algorytmu optymalizacyjnego[3], jednak sam regulator pozostaje klasycznym dyskretnym PID-em. W przypadku wolnych procesów cieplnych regulatory PI/PID w praktyce zazwyczaj są wystarczające, co potwierdzają również prace przeglądowe dotyczące sterowania HVAC[14].

Jeżeli chodzi o logikę działania całych central wentylacyjnych, często odwołuje się do ustandaryzowanych sekwencji pracy opisanych w wytycznych ASHRAE, w szczególności w dokumencie Guideline 36[7]. Wytyczne te definiują gotowe sekwencje sterowania nagrzewnicą, chłodnicą, odzyskiem ciepła i bypassem, uwzględniając pasmo martwe oraz warunki bezpieczeństwa. W projekcie zastosowano uproszczony wariant takiej sekwencji: pojedynczy regulator PI temperatury, którego wyjście jest podzielone na przedziały odpowiadające grzaniu, chłodzeniu, odzyskowi ciepła oraz pasmowi martwemu.

1.3. Sterowniki HVAC na mikrokontrolerach i RTOS

W literaturze można znaleźć przykłady implementacji sterowania HVAC z użyciem mikrokontrolerów oraz prostych systemów operacyjnych czasu rzeczywistego. W pracy Fernandesa opisano regulator PID przepływu powietrza w instalacji wentylacyjnej, zaimplementowany na mikrokontrolerze, z lokalnym pomiarem oraz sterowaniem pracą wentylatora[11]. Układ ten jest zbliżony koncepcyjnie do projektu z

niniejszej pracy: obejmuje jedną pętlę regulacji, czujnik, element wykonawczy i prosty interfejs użytkownika.

W projekcie jako środowisko firmware'u wykorzystano Zephyr RTOS, czyli lekki system operacyjny czasu rzeczywistego przeznaczony do układów wbudowanych, obsługujący wiele architektur i posiadający rozbudowany zestaw sterowników[23, 24]. Dokumentacja Zephyra pokazuje typowe podejście do struktury aplikacji: logika jest podzielona na wątki, a sprzęt (np. magistrale SPI, wyświetlacze) opisuje się w drzewie urządzeń (Devicetree). Do realizacji interfejsu HMI wykorzystano bibliotekę LVGL, czyli popularną otwartą bibliotekę graficzną dla mikrokontrolerów[16, 22]. LVGL udostępnia gotowe widżety (przyciski, wykresy, listy), dzięki czemu można skupić się na logice HVAC, zamiast implementować od podstaw warstwę graficzną.

1.4. Ramy projektowania PCB dla układów mieszanych

Istotnym elementem projektu jest płytka PCB zawierająca zarówno tory analogowe 0–10 V, jak i cyfrowe interfejsy SPI. W literaturze dotyczącej projektowania układów mieszanych (mixed-signal) powtarza się kilka podstawowych zaleceń: logiczny podział płytki na część analogową i cyfrową, kontrola powrotu prądów w masie oraz rozsądne prowadzenie zasilania[8, 9, 17]. Ott zwraca uwagę, że zamiast dzielić masę na dwie osobne płaszczyzny, korzystniej jest utrzymać jedną wspólną masę i wydzielić część analogową oraz cyfrową głównie geometrycznie oraz odpowiednim prowadzeniem ścieżek[17].

W notach aplikacyjnych firm Analog Devices i Microchip przedstawiono przykładowe projekty płytek dla układów mieszanych, gdzie pokazano m.in. sposób umieszczania przetworników ADC na granicy stref analog/digital, prowadzenia linii SPI oraz stosowania przelotek łączących pola masy[8, 9]. W niniejszym projekcie przyjęto podobne podejście: tor 0–10 V znajduje się w wydzielonej części płytki z osobnym polem masy analogowej GNDA, interfejsy SPI umieszczono bliżej złącza do płytki Discovery, a przetwornik ADS8688 pełni rolę „mostu" pomiędzy częścią analogową a cyfrową[2].

1.5. Wymagania

1.5.1. Funkcjonalne

- Pomiar napięć wejściowych w zakresie $0\,\mathrm{V}$ do $10\,\mathrm{V}$ z rozdzielczością $\leq 10\,\mathrm{mV}$.
- Generacja sygnału wyjściowego $0\,\mathrm{V}$ do $10\,\mathrm{V}$ obciążalność $\geq 5\,\mathrm{mA}$.
- GUI na TFT: wizualizacja trendów, konfiguracja kanałów.

1.5.2. Niefunkcjonalne

EMC, bezpieczeństwo, ESD, kalibracja (offset/gain), testowalność.

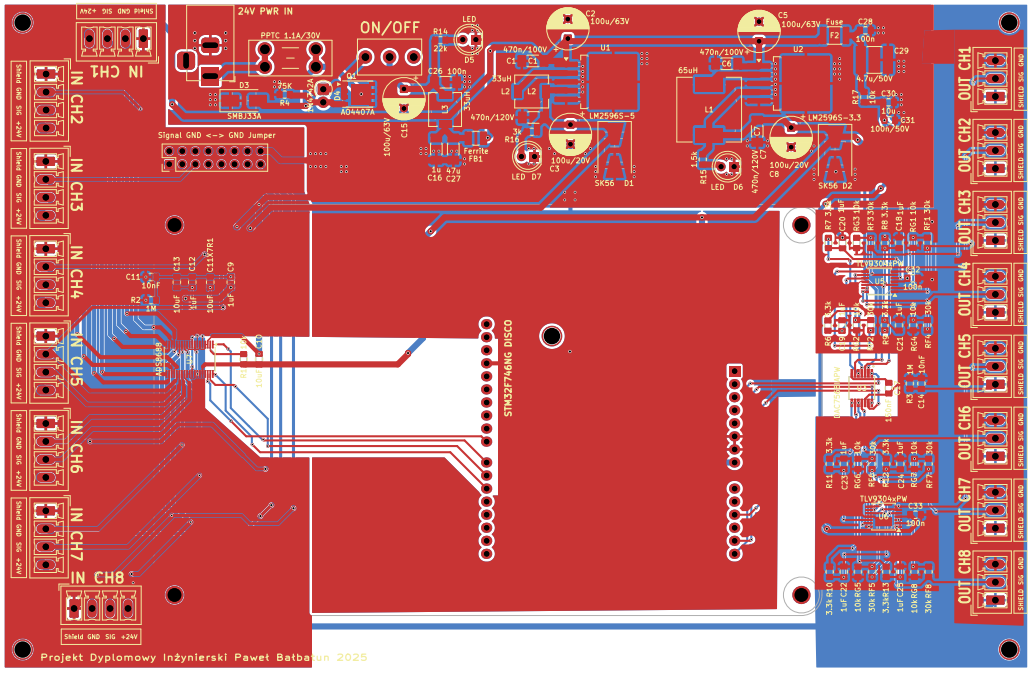
2. PROJEKT CZĘŚCI SPRZĘTOWEJ

2.1. Wymagania sprzętowe — przegląd

Projektowana płytka stanowi jedną, spójną platformę do pomiaru i generacji sygnałów 0–10 V w aplikacjach HVAC[6], współpracującą z zestawem uruchomieniowym STM32F746G-DISCO wyposażonym m.in. w mikrokontroler STM32F7 i panel TFT[19, 20]. Rozwiązanie to ma pełnić rolę uniwersalnego "front-endu"analogowego dla laboratoryjnego sterownika HVAC: umożliwia zarówno rejestrację sygnałów z czujników i przetworników 0–10 V, jak i generację ośmiu niezależnych kanałów 0–10 V do sterowania elementami wykonawczymi (siłowniki przepustnic, zawory mieszające, przetwornice wentylatorów itp.).

Od strony zasilania przewidziano instalacyjne wejście DC (nominalnie 24 V) z podstawowym torem ochronnym (odwrotna polaryzacja, przepięcia, wstępna filtracja), a następnie podział zasilania na osobne gałęzie dla części cyfrowej, analogowej i elementów interfejsowych. Taki układ zmniejsza wpływ zakłóceń na pomiary i stabilizuje pracę torów. Interfejs do świata zewnętrznego obejmuje osiem wejść 0–10 V przygotowanych do bezpiecznego próbkowania przez przetwornik A/C oraz osiem wyjść 0–10 V realizowanych przez przetwornik C/A i wzmacniacze operacyjne.

Istotnym założeniem projektowym było zachowanie kompatybilności elektrycznej i mechanicznej z płytką STM32F746G-DISCO. W centralnej części PCB przewidziano złącze ddupoint, którego raster i położenie odpowiada złączu rozszerzeń zestawu Discovery. Dzięki temu całość tworzy układ kanapkowy: płytka z analogowym interfejsem pełni funkcję karty pomiarowej, a zestaw uruchomieniowy zapewnia moc obliczeniową oraz interfejs użytkownika (TFT z panelem dotykowym).



Rysunek 2.1: Widok płytki PCB z zaprojektowanym rozmieszczeniem elementów i złącz. W centralnej części znajduje się obszar montażu płytki STM32F746G-DISCO.

2.2. Moduł zasilania

Układ zasilania płytki został zaprojektowany tak, aby bezpiecznie przyjąć instalacyjne napięcie stałe (do ok. 24 V) i rozdzielić je na dwie stabilne linie robocze: +5 V oraz +3,3 V. Schemat modułu zasilania przedstawiono na rysunku **??**.

Na wejściu zastosowano gniazdo J1 (DC jack 5,5×2,1 mm), do którego doprowadzane jest napięcie z zewnętrznego zasilacza. Bezpośrednio za złączem znajduje się polimerowy bezpiecznik samoresetujący F1 (PPTC 1,1 A/30 V), pełniący rolę zabezpieczenia nadprądowego w przypadku zwarcia na płytce lub błędnego podłączenia odbiorników. Równolegle do wejścia umieszczono diodę TVS D3 (SMBJ33A) tłumiącą przepięcia.

Ochronę przed odwrotną polaryzacją zasilania zrealizowano w oparciu o tranzystor P-MOSFET mocy Q1 (AO4407A w obudowie SO–8 lub równoważny)[5]. Tranzystor włączono w konfiguracji "idealnej diody": jego źródło jest połączone z wejściem zasilania, dren z resztą układu, a bramka sterowana jest poprzez rezystor R4 i diodę zenera D4 (1N4742A)[1]. Przy poprawnej polaryzacji tranzystor przewodzi z minimalnym spadkiem napięcia na kanale, a w przypadku odwrotnego podłączenia zasilacza blokuje przepływ prądu i chroni dalsze stopnie zasilania.

Za sekcją ochronną pracuje wyłącznik SW1 odcinający cały moduł zasilania. Dioda LED D5 z rezystorem szeregowym R14 sygnalizuje obecność napięcia po stronie wejściowej; pozwala to na szybką kontrolę stanu zasilania przed przetwornicami.

Kolejnym etapem jest wstępna filtracja C-L-C napięcia wejściowego . Dławik L3 (33 µH) oraz kondensatory C15 (100 µF/63 V), C26 (100 nF), C27 (47 µF) i C16 (1 µF) tworzą filtr typu PI, ograniczający wahania napięcia oraz szpilki prądowe związane z pracą przetwornic impulsowych. Dodatkowy koralik ferrytowy FB1, włączony szeregowo w linii 24 V, poprawia tłumienie zakłóceń o wyższych częstotliwościach, które mogłyby przenikać do dalszych części instalacji.

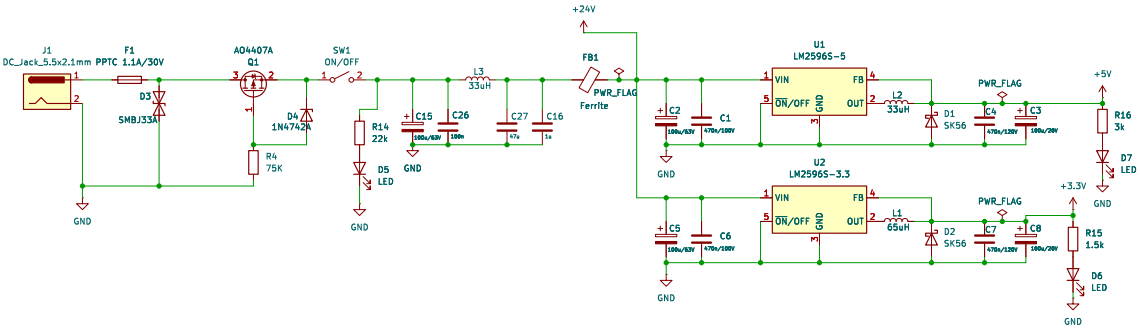
Konwersję napięcia na poziomy logiczne realizują dwie niezależne przetwornice buck z rodziny LM2596S[15]. Układ U1 (LM2596S-5) generuje linię +5 V. W jego torze znajdują się dławik L2 (33 µH), dioda Schottky'ego D1 (SK56) oraz kondensatory wejściowe C2 (100 µF/63 V) i C1 (470 nF) oraz wyjściowe C4 i C3. Analogicznie układ U2 (LM2596S-3,3) dostarcza linię +3,3 V z użyciem dławika L1 (68 µH), diody D2 (SK56) oraz zestawu kondensatorów C5, C6 po stronie wejściowej i C7, C8 po stronie wyjściowej. Elementy zostały dobrane zgodnie z zaleceniami producenta, tak aby zapewnić stabilność pętli regulacji dla zakładanych obciążeń oraz odpowiednio niski poziom tętnień.

Zastosowanie dwóch niezależnych przetwornic zamiast jednego źródła z liniowymi stabilizatorami wtórnymi ma kluczowe znaczenie w kontekście sprawności i wydzielania ciepła. Przy typowych prądach pobieranych przez mikrokontroler, przetwornik A/C i przetwornik C/A z 24 V na 5 V/3,3 V skutkowałoby znaczącym poborem mocy. Przetwornice impulsowe LM2596S pozwalają ograniczyć straty do pojedynczych watów nawet przy pełnym obciążeniu linii 5 V i 3,3 V.

Analogicznie do sygnalizacji becności napięcia na linii 24 V, stan szyn wyjściowych jest sygnalizowany diodami LED: D7 (dla linii +5 V) z rezystorem R16 oraz D6 (dla linii +3,3 V) z rezystorem R15. Na schemacie umieszczono również znaczniki PWR_FLAG, ułatwiające kontrolę ciągów zasilania w narzędziu CAD i zapobiegające fałszywym ostrzeżeniom o "niezasilonych"sieciach. Całość tworzy spójny tor: *wejście i zabezpieczenia → filtracja wstępna → konwersja 24 V na +5 V*/+3,3 V → dystrybucja i sygnalizacja, co przekłada się na stabilną pracę układu..

Na poziomie PCB cały moduł zasilania został umieszczony w górnej części płytki jak widać na obrazie **??**, możliwie blisko gniazda wejściowego oraz oddalony od wrażliwych części analogowych. Ścieżki prowadzące prądy impulsowe z przetwornic LM2596S zaprojektowano jako szerokie i możliwie

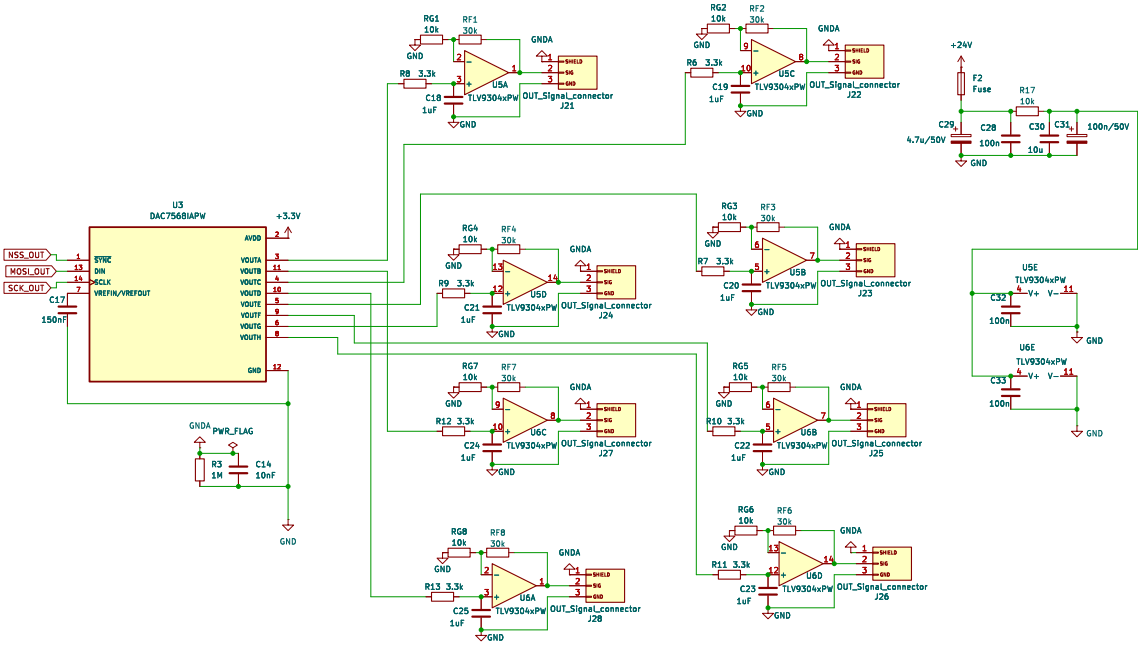
krótkie, z lokalnymi polami masy minimalizującymi powierzchnię pętli prądowych. Dzięki temu ograniczono emisję zakłóceń przewodzonych i promieniowanych oraz uproszczono separację pomiędzy strefą mocy a strefą pomiarową.



Rysunek 2.2: Moduł zasilania płytki z wejściem instalacyjnym 24 V, torem ochrony przed przepięciami i odwrotną polaryzacją oraz przetwornicami step-down LM2596 generującymi linie +5 V i +3,3 V.

2.3. Tor wyjściowy 0–10 V: przetwornik DAC + wzmacniacze operacyjne

Tor wyjściowy generujący sygnały 0–10 V oparto na ośmiokanałowym przetworniku cyfrowo–analogowym U3 (DAC7568IAPW)[10] współpracującym z dwoma czterokanałowymi wzmacniaczami operacyjnymi U5 i U6 (TLV9304xPW)[21]. Dzięki temu możliwe jest niezależne sterowanie wszystkimi ośmioma wyjściami analogowymi. Schemat toru wyjściowego przedstawiono na rysunku **??**.



Rysunek 2.3: Schemat toru wyjściowego 0–10 V wraz z wzmacniaczami operacyjymi.

Przetwornik DAC7568 jest zasilany z linii +3,3 V. Komunikację z płytką STM32F746G-DISCO realizuje poprzez magistralę SPI: linie `NSS_OUT`, `MOSI_OUT` i `SCK_OUT` zostały wyprowadzone z MCU i doprowadzone do odpowiednich pinów układu U3. Z uwagi na charakter przetwornika (układ typu "write-only") nie przewidziano linii MISO; konfiguracja rejestrów i aktualizacja wyjść odbywa się wyłącznie poprzez wysyłanie ramek danych z mikrokontrolera.

Wyprowadzenie `VREFIN/VREFOUT` służy do ustalenia napięcia referencyjnego, w projekcie wy-

korzystano wewnętrzne źródło odniesienia przetwornika, dlatego pin został odsprzęgnięty kondensatorem C17 (150 nF) umieszczonym możliwie blisko wyprowadzeń, zgodnie z zaleceniami producenta[10]. Masę części analogowej doprowadzono do masy analogowej GNDA, która na PCB prowadzona jest jako wydzielona wyspa z kontrolowanym połączeniem do wspólnej GND poprzez elementy R2/C11 (opisane szerzej w podrozdziale o torze wejściowym).

Każdy z ośmiu kanałów wyjściowych DAC (VOUTA–VOUTH) jest dalej kształtowany przez prosty filtr dolnoprzepustowy RC na wejściu wzmacniacza: rezystor szeregowy (R6–R13, typowo 3,3 kΩ) oraz kondensator do masy (C18, C21–C25, 1 µF). Wyznacza to częstotliwość odcięcia rzędu

$$f_c \approx \frac{1}{2\pi RC} \approx \frac{1}{2\pi \cdot 3,3\ \text{k}\Omega \cdot 1\ \mu\text{F}} \approx 48\ \text{Hz}, \tag{2.1}$$

co skutecznie tłumi szum oraz poszarpanie przebiegu pochodzące z aktualizacji DAC, a jednocześnie jest w pełni wystarczające dla powolnych procesów w systemach HVAC.

Wzmacniacze U5A–U5D oraz U6A–U6D pracują w konfiguracji nieodwracającej i są zasilane z linii +24 V (wg producenta maksymalne napięcie zasilania to 40 VDC). Jednak w ramach zabezpieczenia na torze zasilającym idącym do wzmacniaczy znajdzuje się polimerowy bezpiecznik samoresetujący F2 oraz filtr dolnoprzepustowy C-R-C skłądający się z rezystora R17, kondensatorów elektrolitycznych C29 i C31 i ceramicznych c28, C30. Filtr usuwa wszelkie niechciane zakłocenia które mogą występować w torze zasilania.

Zastosowanie wzmacniaczy o szerokim zakresie napięć zasilania pozwala uzyskać odpowiedni zapas napięciowy dla wyjść 0–10 V bez konieczności stosowania dodatkowych przetwornic podwyższających. Dla każdego kanału zastosowano identyczną sieć sprzężenia zwrotnego: rezystor do masy RG (10 kΩ) oraz rezystor w pętli sprzężenia RF (30 kΩ). Wzmocnienie napięciowe pojedynczego toru wynosi więc

$$A_v = 1 + \frac{R_F}{R_G} = 1 + \frac{30\ \text{k}\Omega}{10\ \text{k}\Omega} = 4. \tag{2.2}$$

Przy referencji DAC rzędu 2,5 V umożliwia to uzyskanie pełnego zakresu 0–10 V na wyjściu wzmacniacza, z zapasem na niewielkie tolerancje i błędy kalibracji.

Wyjścia poszczególnych wzmacniaczy są wyprowadzone na uniwersalne złącza J21–J28. Każde złącze udostępnia linię sygnałową 0–10 V, odniesienie SG (signal ground) oraz pin SHIELD przeznaczony do ekranowania przewodów.

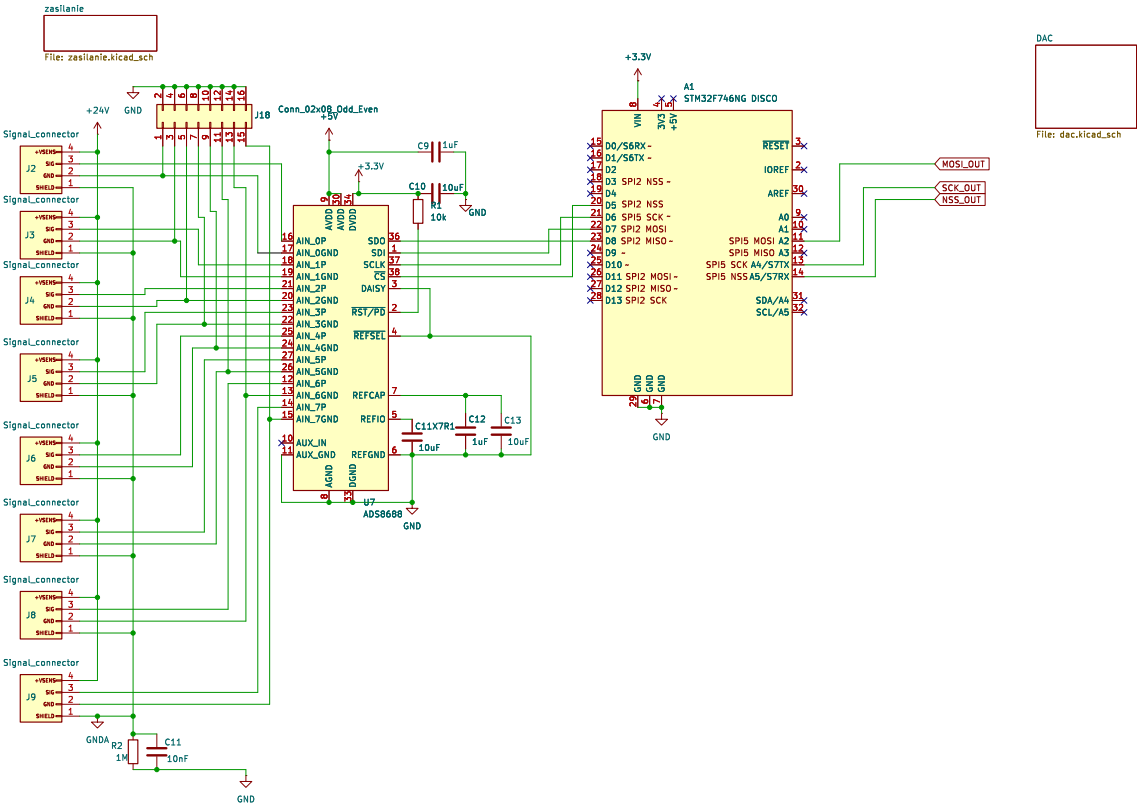
Na płytce PCB **??** złącza wyjściowe zostały rozmieszczone wzdłuż prawej krawędzi w regularnym rastrze, co ułatwia prowadzenie przewodów i daje poczucie "przepływu"sygnałów przez sterownik (od lewej do prawej). Ścieżki sygnałowe pomiędzy wzmacniaczami a złączami są możliwie krótkie i prowadzone nad ciągłą płaszczyzną masy, co redukuje indukcyjność pętli i przesłuchy pomiędzy kanałami.

Podsumowując, tor wyjściowy ma strukturę: *STM32 → DAC7568 → filtr RC → wzmacniacz nieodwracający o wzmocnieniu 4 → złącze sygnałowe z ekranem i zasilaniem pola*, co zapewnia zarówno elastyczność sterowania, jak i zgodność z powszechnie stosowanym standardem 0–10 V.

2.4. Tor wejściowy 0–10 V: interfejs pomiarowy

Tor wejściowy odpowiada za obsługę czujników przemysłowych (pasywnych i aktywnych) oraz bezpieczne doprowadzenie sygnałów 0–10 V do wielokanałowego przetwornika A/C ADS8688[**TI_ADS8688**]. Schemat tej części układu na zdjęciu **??**. Przetwornik ten integruje w sobie przełączany multiplekser wejściowy, programowalne zakresy napięciowe oraz wewnętrzny front-end zabezpieczający, co pozwala uprościć zewnętrzny tor analogowy.

Po lewej stronie schematu rozmieszczono osiem identycznych złączy wejściowych (oznaczonych jako Signal_connector, J2–J9). Na każdym złączu wyprowadzono cztery piny:



Rysunek 2.4: Schemat toru wejściowego 0–10 V wraz z przetwornikami ADC oraz połączenia do mikrokontrolera.

- SIG — linia sygnałowa 0–10 V,
- SG — dedykowany powrót sygnałowy (signal ground) dla danego kanału,
- SHIELD — ekran przewodu, przeznaczony do podłączenia oplotu lub ekranu kabla,
- +24 V — zasilanie czujnika/konwertera z instalacji 24 V.

Takie ustandaryzowane złącza pozwalają na łatwą zamianę czujników pomiędzy kanałami oraz ograniczają liczbę pomyłek przy okablowaniu.

Linia SIG każdego złącza jest prowadzona bezpośrednio do odpowiedniego pinu wejściowego przetwornika (AIN_xP), natomiast powrót SG trafia do przypisanego ujemnego wejścia odniesienia kanału (AIN_xGND). W ten sposób tworzony jest układ wejściowy o charakterze pseudo-różnicowym, w którym każdy kanał ma swój lokalny powrót odniesiony do tej samej masy analogowej GNDA. Taka topologia kompensuje spadki napięć i przesłuchy na przewodzie powrotnym oraz poprawia odporność na zakłócenia wspólne przy dłuższych odcinkach okablowania, co jest kluczowe w instalacjach HVAC rozproszonych na dużej przestrzeni.

W torze wejściowym nie stosuje się dodatkowych dzielników ani zewnętrznych filtrów antyaliasingowych; skalowanie i zabezpieczenie wejść realizuje wewnętrzny front-end przetwornika ADS8688[**TI_ADS8688**], który oferuje kilka przełączanych zakresów pomiarowych oraz obwody ograniczające prądy przy przepięciach.

Piny SHIELD wszystkich wejść są połączone do wspólnego ekranu wylanego pod ścieżkami sygnału i docciągniętego względem masy przez filtr RC: rezystor R2 (1 MΩ) równolegle z kondensatorem C11 (10 nF) pomiędzy GNDA i GND. Zapewnia to upływ ładunków statycznych oraz tłumienie składowych o wysokiej częstotliwości, a jednocześnie ogranicza stałoprądowe prądy pętli masy i chroni

ekran przed "przeciąganiem"potencjału przez inne urządzenia podłączone do tej samej instalacji. Rozwiązanie to stanowi jednocześnie kontrolowany punkt połączenia masy analogowej i cyfrowej — GNDA jest używana w torach wejściowych i wyjściowych, natomiast GND stanowi referencję dla logiki cyfrowej i zasilania 3,3 V.

Część analogowa przetwornika ADS8688[**TI_ADS8688**] jest zasilana z linii +5 V (kondensator C9 1 µF blisko pinów AVDD/AGND), natomiast część cyfrowa z linii +3,3 V (kondensator C10 10 µF przy DVDD/DGND). Linia RST/PD jest podciągnięta rezystorem 10 kΩ do +3,3 V i może być sterowana z mikrokontrolera, co umożliwia programowe resetowanie przetwornika oraz przechodzenie w tryb uśpienia w stanie bezczynności.

Odniesienie napięciowe przetwornika realizowane jest wewnętrznie; piny REFCAP, REFIO i REFGND są odsprzęgnięte kondensatorami klasy X7R (C12 i C13 po 1 µF) umieszczonymi możliwie blisko wyprowadzeń, zgodnie z zaleceniami producenta[**TI_ADS8688**]. Zapewnia to niskoszumowe, stabilne napięcie odniesienia, co bezpośrednio przekłada się na rozdzielczość efektywną przetwornika.

Na schemacie przewidziano również dodatkowe złącze J18 w postaci listwy goldpin z możliwością założenia zworek. Złącze to służy do konfigurowania połączenia pomiędzy masą sygnałową (SG), wykorzystywaną jako powrót dla wejść 0–10 V, a ogólną masą układu (GND). Poprzez odpowiednie ustawienie zworek użytkownik może zdecydować, czy masa sygnałowa ma pozostać możliwie "czysta"i odniesiona głównie do GNDA (praca z czujnikami pasywnymi), czy też powinna zostać zwarta z masą zasilania w celu zasilania czujników aktywnych, wymagających wspólnego potencjału odniesienia dla toru zasilania i sygnału.

Takie rozwiązanie ma kilka zalet:

- umożliwia elastyczną konfigurację toru wejściowego w zależności od typu podłączonych czujników (pasywne z odseparowaną masą sygnałową vs. aktywne wymagające wspólnego potencjału zasilania i sygnału),
- poprawia kompatybilność z typowymi przetwornikami 0–10 V, które zakładają wspólną masę zasilania i wyjścia napięciowego,
- pozwala na łatwe eksperymentowanie i diagnostykę w warunkach laboratoryjnych — zmiana konfiguracji sprowadza się do przełożenia zworek, bez konieczności modyfikacji PCB ani rozcinania ścieżek.

Komunikacja z mikrokontrolerem odbywa się po magistrali SPI: linie SDI, SDO, SCLK i CS zostały połączone z odpowiednimi pinami do MCU. STM32F7 wyposażony jest w dwa oddzielne interfejsy SPI. Dzięki temu oba przetworniki (ADS8688 i DAC7568) mogą współdzielić komunikować się z MCU niezależnie od zajętości??? interfejsów; obie magistrale działają na osobnych wątkach.

Na poziomie PCB **??** przetwornik ADS8688 oraz złącza wejściowe zostały umieszczone w lewej części płytki, w niewielkiej odległości od siebie. Ścieżki sygnałowe SIG/SG prowadzone są nad pełną płaszczyzną masy GNDA, z zachowaniem odstępów pomiędzy kanałami, co ogranicza pojemnościowe sprzężenia krzyżowe. Ekrany SHIELD tworzą osobną, częściowo wylaną strefę połączoną z GNDA poprzez elementy R2/C11.

2.5. Projekt PCB i separacja stref

Widok płytki PCB przedstawiono na rysunku wygenerowanym z narzędzia CAD **??**. Płytka została zaprojektowana jako dwuwarstwowa, z rozległymi polami masy w obu warstwach. Główna powierzchnia zajmowana jest przez prostokątny obrys odpowiadający formatowi płytki Discovery, w którego obrębie

przewidziano wycięty obszar (strefę bez elementów i ścieżek) stanowiący miejsce montażu zestawu STM32F746G-DISCO. W czterech narożach oraz w pobliżu złączy umieszczono otwory montażowe umożliwiające sztywne zamocowanie całości do płyty bazowej lub obudowy.

Pod względem funkcjonalnym płytkę można podzielić na trzy główne strefy:

1. **Strefa mocy** — w górnej części, obejmująca gniazdo zasilania J1, układ ochrony (F1, D3, D4, Q1, L3, FB1) oraz przetwornice LM2596S. Ścieżki o dużych prądach i impulsowych zmianach prądu prowadzone są lokalnie, nad fragmentarycznymi polami masy, co ogranicza emisję zakłóceń.

2. **Strefa analogowa** — przy lewej i prawej krawędzi, obejmująca przetwornik ADS8688, przetwornik DAC7568, wzmacniacze TLV9304 oraz złącza wejściowe i wyjściowe. W tych częściach płytki zastosowano wydzieloną masę GNDA, która jest łączona z ogólną masą GND w jednym punkcie poprzez elementy R2/C11.

3. **Strefa cyfrowa** — w centralnej części, wokół rastru GPIO mikrokontrolera, gdzie prowadzone są linie SPI. Tutaj masa GND jest wylana jako osobna wyspa, a przejścia do masy analogowej są kontrolowane.

Rozdzielenie tych stref w przestrzeni płytki ogranicza przesłuchy pomiędzy torami, a jednocześnie pozwala na intuicyjną analizę układu podczas uruchamiania i diagnostyki. Szerokości ścieżek dobrano zgodnie z przewidywanymi prądami (najszersze dla linii 24 V, 5 V i 3,3 V, węższe dla linii sygnałowych). Przy przejściach pomiędzy warstwami stosowane są przelotki w "gęstych"grupach, tak aby zapewnić zwarty powrót prądu między płaszczyznami masy.

Dodatkowym elementem ułatwiającym pracę z płytką jest rozbudowana warstwa opisowa (silkscreen), na której zaznaczono nazwy złączy, kierunki numeracji pinów, oznaczenia kanałów wejściowych i wyjściowych oraz podstawowe kierunki przepływu sygnału (strzałki). Dzięki temu użytkownik może korzystać z płytki w warunkach laboratoryjnych praktycznie bez konieczności sięgania do schematu.

Opisany w niniejszym rozdziale projekt części sprzętowej stanowi bazę dla dalszych rozważań dotyczących implementacji algorytmów sterowania i architektury oprogramowania w rozdziałach poświęconych części programowej pracy.

3. FIRMWARE

Część programowa projektu pełni rolę interfejsu pomiędzy opracowaną platformą sprzętową a docelowym użytkownikiem – studentem realizującym ćwiczenia. Jej zadaniem jest nie tylko poprawne sterowanie wyjściami analogowymi i akwizycja sygnałów 0–10 V z przetworników, ale również prezentacja stanu układu w czytelnej formie graficznej, umożliwienie zmiany parametrów regulatora oraz wygodne przełączanie konfiguracji laboratoryjnych bez konieczności rekompilacji oprogramowania.

Z tego powodu firmware nie został zbudowany jako prosta aplikacja „bare–metal" ani w oparciu o szablony generatora STM32CubeMX, lecz jako aplikacja systemu Zephyr RTOS, z wykorzystaniem biblioteki `LVGL` do budowy interfejsu użytkownika. W kolejnych podrozdziałach opisano motywację tego wyboru, strukturę aplikacji, opis modelu konfiguracji systemu HVAC w oparciu o JSON, logikę regulacji oraz działanie interfejsu HMI.

3.1. Wybór środowiska uruchomieniowego

Projekt wymaga równoległego i deterministycznego wykonywania kilku zadań: odświeżania HMI na wyświetlaczu TFT, generacji sygnałów 0–10 V (komunikacja SPI z przetwornicą DAC) oraz obsługa drugiej magistrali SPI do komunikacji z przetwornicą ADC. Wszystkie te funkcje muszą być realizowane przy zachowaniu powtarzalnych czasów reakcji. Z tych powodów jako środowisko uruchomieniowe wybrano `Zephyr RTOS` zintegrowany z biblioteką graficzną `LVGL`.

Zastosowanie systemu czasu rzeczywistego zamiast prostego „bare–metal" pozwala w prosty sposób rozdzielić zadania i przypisać je do osobnych wątków: wątku interfejsu użytkownika, wątków akwizycji i przetwarzania danych oraz wątków odpowiedzialnych za komunikację ze sprzętem. Dzięki temu możliwe jest jednoczesne utrzymanie płynnego odświeżania UI, bezpiecznej obsługi przerwań z przetworników oraz terminowego odświeżania wyjść analogowych sterujących urządzeniami HVAC.

3.1.1. Dlaczego Zephyr + LVGL?

Zephyr udostępnia deterministyczny kernel o prostym, ale wystarczająco elastycznym modelu współbieżności. Wątki o priorytetach, kolejki *workqueue*, przerwania oraz timery wysokiej rozdzielczości umożliwiają zdefiniowanie jasnego podziału zadań i ich wzajemnych zależności. W projekcie wykorzystano tę możliwość do wydzielenia ścieżki krytycznej czasowo (akwizycja i generacja sygnałów) od zadań mniej wrażliwych na opóźnienia (obsługa UI, diagnostyka, ładowanie konfiguracji). Tryb *tickless* ogranicza narzut związany z tykaniem systemowym, a jednocześnie pozwala zachować stałe okresy odświeżania UI oraz stabilne czasy próbkowania.

Istotną zaletą Zephyra jest spójny ekosystem sterowników oraz wspólna warstwa konfiguracji sprzętu. Standardowe API dla interfejsów SPI, I^2C, ADC, DAC/PWM, GPIO czy DMA, uzupełnione mechanizmami `DeviceTree` i `Kconfig`, redukuje do minimum ilość kodu "klejącego" między aplikacją a warstwą sprzętową. Dla mikrokontrolerów z rodziny STM32 dostępne są gotowe drivery oraz integracja z biblioteką HAL producenta, co znacząco przyspiesza uruchomienie peryferiów, takich jak kontroler SPI współpracujący z zewnętrznymi przetwornikami A/C i C/A. Zmiana konfiguracji sprzętu (np. inny pin CS, dodatkowy kanał) sprowadza się najczęściej do modyfikacji pliku `.overlay` `DeviceTree`, bez ingerencji w logikę aplikacyjną.

Zephyr dostarcza ponadto oficjalny subsystem `LVGL`, odpowiedzialny za integrację biblioteki graficznej z systemem operacyjnym. W praktyce oznacza to gotową obsługę zegara *tick*, wątku renderują-

cego, przydziału pamięci oraz sterowników wyświetlaczy i urządzeń wejściowych. Konfiguracja odbywa się poprzez parę `Kconfig+DeviceTree`, gdzie określa się m.in. rozmiary buforów ekranu, sposób odświeżania (*flush callback*) oraz mapowanie wejścia dotykowego. Dzięki temu warstwa HMI może zostać zaimplementowana w całości w LVGL, bez konieczności ręcznego "sklejania" sterownika wyświetlacza, sterownika dotyku i logiki zadań systemowych.

Wybór Zephyra jest również uzasadniony dostępnością narzędzi deweloperskich oraz ekosystemu okołosystemowego. Jednolity system budowania oparty na `CMake` i menedżerze `west`, wbudowany logger, konsola `shell`, obsługa trwałych ustawień (NVS), różne systemy plików (FAT, LittleFS) oraz zintegrowany framework testowy (`twister`) ułatwiają utrzymanie projektu oraz jego automatyzację. W kontekście pracy inżynierskiej ważna jest także skalowalność: Zephyr oferuje stosy sieciowe (Ethernet, BLE, IP), jak również bootloader `MCUboot`, co otwiera drogę do przyszłego rozszerzenia urządzenia o zdalne aktualizacje czy komunikację sieciową bez zmiany fundamentów projektu.

Biblioteka LVGL, wykorzystana jako warstwa HMI, jest naturalnym wyborem dla mikrokontrolerów klasy STM32F7 z kolorowym TFT. Zapewnia rozbudowany zestaw widżetów (przyciski, suwaki, listy rozwijane, wykresy) oraz mechanizmy układu obiektów (flex, grid), co pozwala zbudować zarówno ekran diagnostyczny, jak i bardziej złożony panel sterowania. LVGL jest aktywnie rozwijana, dobrze udokumentowana i szeroko stosowana w systemach wbudowanych, a jej integracja z Zephyrem jest oficjalnie wspierana. Użycie powszechnie znanego stosu Zephyr+LVGL zmniejsza ryzyko problemów z utrzymaniem projektu w przyszłości.

Nie bez znaczenia pozostaje kwestia licencjonowania i wsparcia społeczności. Zephyr jest projektem rozwijanym pod licencją `Apache-2.0`, a LVGL pod licencją `MIT`. Obie licencje są permisywne i sprzyjają wykorzystaniu w projektach komercyjnych oraz akademickich. Aktywne społeczności użytkowników oraz profesjonalne wsparcie firm współtworzących te projekty ułatwiają rozwiązywanie problemów oraz zapewniają długoterminową stabilność stosu programowego.

3.1.2. Porównanie z podejściem bare–metal

W podejściu bare–metal cała logika wielozadaniowa musiałaby zostać zaimplementowana ręcznie w oparciu o przerwania, timery sprzętowe i pętlę główną. Oznacza to konieczność samodzielnego zaprojektowania i utrzymania harmonogramu zadań, mechanizmów synchronizacji oraz kolejek zdarzeń, a także integracji z DMA i priorytetami przerwań. W przypadku aplikacji łączącej akwizycję danych, generację sygnałów 0–10 V oraz rozbudowany interfejs HMI oznacza to znacząco wyższą złożoność kodu oraz większe ryzyko błędów w sytuacjach granicznych (np. zakleszczenia, niekontrolowany jitter czasów odświeżania, utrata próbek pomiarowych).

Z perspektywy utrzymywalności, rozwiązanie bare–metal jest silniej związane z konkretną konfiguracją sprzętową. Każda zmiana mikrokontrolera, peryferiów lub nawet samego przypisania pinów wymaga modyfikacji w wielu miejscach kodu. Brak standaryzowanej warstwy w rodzaju `DeviceTree`/`Kconfig` utrudnia przenoszenie konfiguracji między płytkami oraz automatyzację procesu budowania. W praktyce utrudnia to zarówno dalszy rozwój projektu, jak i potencjalne wykorzystanie istniejącej bazy kodu w innych urządzeniach.

3.1.3. Porównanie z FreeRTOS i innymi RTOS

FreeRTOS jest jednym z najpopularniejszych lekkich kernelów RTOS, jednak jego podstawowa dystrybucja dostarcza głównie sam kernel i podstawowe prymitywy synchronizacji. Ekosystem sterowników, systemów plików, stosów komunikacyjnych czy integracji z bibliotekami graficznymi jest budowany w oparciu o zewnętrzne komponenty, często od różnych producentów. Odpowiedzialność za ich dobór,

integrację i utrzymanie spoczywa w całości na autorze projektu.

W Zephyrze większość tych elementów jest dostępna w ramach jednego, spójnego repozytorium, objętego wspólnym procesem testowania i wydawniczym. Konfiguracja sprzętu i modułów systemu odbywa się w jednolity sposób przez `DeviceTree` i `Kconfig`, co upraszcza przenoszenie projektu między różnymi płytkami oraz ułatwia jego automatyczne budowanie. W przypadku FreeRTOS typowym rozwiązaniem jest korzystanie z mieszaniny plików *board support* przygotowanych przez producenta mikrokontrolera oraz własnych skryptów budowania, co zwiększa rozproszenie konfiguracji i utrudnia zachowanie spójności między platformami.

Zarówno FreeRTOS, jak i Zephyr umożliwiają uruchomienie LVGL, jednak w przypadku FreeRTOS integracja zwykle wymaga ręcznego spięcia zegara *tick*, zadań renderujących oraz sterowników wyświetlacza i wejść dotykowych. W Zephyrze LVGL jest komponentem pierwszoplanowym: posiada dedykowane opcje `Kconfig`, gotowe przykładowe drivery oraz pełną integrację z systemem logowania i zarządzania pamięcią. W konsekwencji konfiguracja interfejsu graficznego ogranicza się do ustawienia kilku parametrów i implementacji funkcji *flush* specyficznej dla danej płytki, zamiast budowania całego stosu od podstaw.

Wreszcie, Zephyr dostarcza zunifikowane narzędzia do testowania (`twister`), zarządzania modułami (`west`) oraz bogaty katalog przykładów, w tym projektów korzystających jednocześnie z LVGL, systemu plików i bardziej zaawansowanych peryferiów. W kontekście pracy inżynierskiej i potencjalnej dalszej rozbudowy platformy takie środowisko znacząco zmniejsza nakład pracy związany z integracją i utrzymaniem całości.

3.1.4. Konkluzja

Z perspektywy wymagań stawianych projektowi — stabilnego odświeżania interfejsu użytkownika, deterministycznej akwizycji i generacji sygnałów 0–10 V, szybkiego uruchomienia peryferiów, możliwości dalszej rozbudowy oraz łatwej rekonfiguracji parametrów pracy — wybór **Zephyr RTOS + LVGL** jest rozwiązaniem najbardziej racjonalnym. Zestaw ten minimalizuje ilość kodu specyficznego dla sprzętu, skraca czas integracji i ogranicza ryzyko błędów czasowych, jednocześnie pozostawiając przestrzeń na rozbudowę funkcjonalności (np. o komunikację sieciową czy zdalne aktualizacje). W porównaniu zarówno z podejściem bare–metal, jak i z lekkimi kernelami w rodzaju FreeRTOS, Zephyr oferuje pełniejszy, spójny ekosystem, który lepiej wspiera zarówno bieżące potrzeby projektu, jak i jego przyszły rozwój.

3.2. Architektura aplikacji i główne moduły

Implementacja części programowej została zrealizowana w jednym pliku źródłowym `main.c`, jednak kod jest logicznie podzielony na kilka wyraźnych warstw:

1. **Abstrakcja wejść/wyjść analogowych** – zestaw funkcji:

   - `static float read_ai_voltage(int ch)`,
   - `static float read_ao_voltage(int ch)`,
   - `static void  write_ao_voltage(int ch, float voltage)`,

   które w obecnej wersji projektu pełnią rolę *placeholderów*. Funkcje przyjmują numer kanału i zwracają/ustawiają wartość w woltach (0–10 V). Ich implementacje będą w przyszłości powiązane z właściwymi sterownikami zewnętrznych przetworników A/C i C/A opisanych w części sprzętowej. Już na tym etapie kod utrwala jednak zamierzony interfejs funkcjonalny: w pozostałych częściach

aplikacji nie operuje się bezpośrednio na rejestrach przetworników, lecz na abstrakcyjnych "kanałach" AI/AO.

2. **Model konfiguracji HVAC** – zestaw struktur opisujących parametry regulatora i przypisania kanałów:
   - `struct hvac_pid_cfg` – dyskretne parametry regulatora (`kp`, `ki`, `kd`),
   - `struct hvac_io_cfg` – mapowanie kanałów AI/AO na konkretne wielkości procesowe (temperatura nawiewu, wyciągu, czujnik przeciwzamrożeniowy, wyjście falownika wentylatora, nagrzewnicy, chłodnicy itp.),
   - `struct hvac_seq_band` oraz `struct hvac_seq_cfg` – opis tzw. "pasm sekwencji" (ang. *sequence bands*) dla chłodzenia, grzania, odzysku ciepła i strefy nieczułości (deadband),
   - `struct hvac_config` – struktura nadrzędna agregująca nastawę temperatury (`setpoint`), parametry PID, mapowanie IO i konfigurację pasm sekwencji.
3. **Warstwa logiki regulacji** – funkcje implementujące regulator dyskretny oraz logikę sekwencyjnego przydziału sygnału sterującego do poszczególnych wyjść:
   - `static void  hvac_pid_reset(struct hvac_pid_state *st)`,
   - `static float hvac_pid_step(const struct hvac_pid_cfg *cfg, struct hvac_pid_state *st,`
   - `static void  hvac_apply_sequence(float pid_out_pct, const struct hvac_config *cfg, flo`
   - `static float hvac_get_extract_temp_c(void)`,
   - `static void  hvac_control_step(float dt_sec)`.
4. **Warstwa prezentacji i HMI** – zestaw funkcji operujących na obiektach LVGL:
   - globalne wskaźniki do ekranów: `screen_dashboard`, `screen_io`, `screen_config`,
   - tworzenie nagłówka i przycisków nawigacyjnych: `static void create_nav_buttons(lv_obj_t *nav_con` `static void create_header(lv_obj_t *parent, const char *title_text)`,
   - tworzenie ekranów: `static void create_dashboard_screen(void)`, `static void create_io_visuali` `static void create_config_screen(void)`,
   - funkcje odświeżające etykiety na ekranach: `hvac_update_io_values()`, `hvac_refresh_io_role_labels(` `hvac_refresh_dashboard_setpoint()`, `hvac_refresh_dashboard_seq_labels()`.
5. **Warstwa obsługi konfiguracji JSON** – funkcje:
   - `static int  hvac_load_config_from_json(char *json, size_t len, struct hvac_config *out`
   - `static void hvac_apply_config(const struct hvac_config *cfg)`,

   wraz z tablicami `hvac_config1_json[]` i `hvac_config2_json[]` zawierającymi przykładowe konfiguracje w formacie JSON.
6. **Funkcja `main`** – inicjalizacja systemu, ekranu LCD, utworzenie ekranów LVGL oraz główna pętla, w której wywoływane są zarówno `lv_timer_handler()`, jak i krok funkcji `hvac_control_step()`.

Takie uwarstwienie kodu powoduje, że poszczególne aspekty działania stanowiska – konfiguracja, regulacja, prezentacja danych – są od siebie możliwie niezależne. Przykładowo, zmiana sposobu pozyskiwania temperatury (inny czujnik, inne skalowanie przetwornika) wymaga modyfikacji wyłącznie funkcji `read_ai_voltage()` i ewentualnego przelicznika w `hvac_get_extract_temp_c()`, bez ingerencji w logikę UI czy model JSON.

Model konfiguracji HVAC i obsługa JSON

Jednym z ważniejszych założeń projektowych było umożliwienie *konfigurowania* stanowiska przez prowadzącego lub studenta, bez rekompilacji firmware'u. W tym celu logika HVAC została opisana w postaci struktury `struct hvac_config`, która jest ściśle odwzorowana na wewnętrzną reprezentację pliku JSON.

Struktura konfiguracji obejmuje:

- pole `setpoint` – zadana temperatura komfortu w stopniach Celsjusza,
- strukturę `hvac_pid_cfg pid` – dyskretne współczynniki regulatora PI(D) w jednostkach całkowitych (wygodnych do wpisywania w pliku tekstowym),
- strukturę `hvac_io_cfg io` – indeksy kanałów analogowych AI i AO przypisane do konkretnych sygnałów procesowych; wartości `-1` oznaczają brak przypisania,
- strukturę `hvac_seq_cfg seq` – cztery pasma sekwencji: `cooling`, `heating`, `heat_recovery` oraz `deadband`, opisane przez pary `from_percent` i `to_percent` w zakresie od -100% do $+100\%$.

W celu przekształcenia JSON na strukturę C wykorzystano bibliotekę JSON systemu Zephyr. Dla każdej struktury zdefiniowano tablicę deskryptorów:

- `hvac_pid_descr[]` dla `struct hvac_pid_cfg`,
- `hvac_io_descr[]` dla `struct hvac_io_cfg`,
- `hvac_seq_band_descr[]` oraz `hvac_seq_descr[]` dla odpowiednio `struct hvac_seq_band` i `struct hvac_seq_cfg`,
- `hvac_cfg_descr[]` – nadrzędny deskryptor dla `struct hvac_config`.

Każdy deskryptor jest zdefiniowany makrami `JSON_OBJ_DESCR_PRIM` (dla pól liczbowych) lub `JSON_OBJ_DESCR_OBJECT` (dla pól zagnieżdżonych). Funkcja:

```
static int hvac_load_config_from_json(char *json, size_t len,
                                      struct hvac_config *out_cfg)
```

przyjmuje wskaźnik na bufor z tekstem JSON i jego długość, czyści strukturę wyjściową (`memset(out_cfg, 0, sizeof(*out_c`) a następnie wywołuje `json_obj_parse()` z odpowiednią tablicą deskryptorów. W przypadku błędu parsowania funkcja loguje szczegóły (`LOG_ERR("JSON parse error: %d", ret)`) i zwraca kod błędu, w przeciwnym wypadku uzupełniona struktura `hvac_config` jest gotowa do użycia.

W aktualnej wersji aplikacji konfiguracje są zapisane w postaci dwóch tablic znakowych:

```
static char hvac_config1_json[] = "{ ... }";
static char hvac_config2_json[] = "{ ... }";
```

i reprezentują dwie przykładowe konfiguracje stanowiska. Ekran "Configuration Loader" pozwala użytkownikowi przełączać się między nimi. Kluczowe jest jednak to, że funkcja `hvac_load_config_from_json()` nie ma żadnej wiedzy o źródle danych: równie dobrze może przyjąć bufor wczytany z pliku na karcie SD (system plików FAT) lub z interfejsu komunikacyjnego. Dzięki temu architektura oprogramowania jest przygotowana na późniejszą rozbudowę o zewnętrzne pliki konfiguracyjne bez ingerencji w logikę sterowania czy interfejs.

Zaimplementowana funkcja:

`static void hvac_apply_config(const struct hvac_config *cfg)`

aktualizuje globalną strukturę `g_hvac_cfg`, resetuje stan integratora regulatora (`hvac_pid_reset()`) oraz odświeża etykiety w interfejsie graficznym:

- nazwy ról kanałów AI/AO poprzez `hvac_refresh_io_role_labels()`, wykorzystując funkcje pomocnicze `hvac_ai_role_name_for_channel()` i `hvac_ao_role_name_for_channel()`,
- ekran Dashboard poprzez `hvac_refresh_dashboard()`.

Regulator PI i logika sekwencyjna

Funkcjonalne serce firmware'u stanowi uproszczony regulator PI, który na podstawie różnicy pomiędzy zadaną temperaturą a temperaturą wyciągu wyznacza wirtualny sygnał sterujący w zakresie od -100% do $+100\%$. Następnie sygnał ten jest przekształcany przez logikę sekwencyjną na trzy niezależne sygnały wyjściowe:

- udział grzania (`heater_pct`),
- udział chłodzenia (`cooler_pct`),
- stopień otwarcia obejścia wymiennika (`bypass_pct`).

Struktura `hvac_pid_state` przechowuje stan integratora (`i_term`). Funkcja:

```
static float hvac_pid_step(const struct hvac_pid_cfg *cfg,
                           struct hvac_pid_state *st,
                           float error,
                           float dt_sec)
```

realizuje dyskretny krok regulatora:

1. oblicza człon proporcjonalny jako iloczyn błędu i wzmocnienia `kp`,
2. aktualizuje i akumuluje człon całkujący na podstawie `ki`, błędu i kroku czasowego `dt_sec`,
3. sumuje oba składniki i ogranicza wynik do przedziału $[-100\%, 100\%]$, zapobiegając nadmiernemu narastaniu sterowania (prosta forma ochrony przed wind–up).

Aktualna temperatura wyciągu jest pobierana przez:

`static float hvac_get_extract_temp_c(void)`

która:

- korzysta z pola `g_hvac_cfg.io.t_extract_ai` do ustalenia numeru kanału analogowego przypisanego do temperatury wyciągu,
- wywołuje `read_ai_voltage(ch)` i dokonuje prostego przeliczenia napięcia na stopnie Celsjusza ($T \approx k \cdot U$); w aktualnej wersji jest to świadomie uproszczone, ponieważ kalibracja oraz dokładne przeliczniki zależą od typu czujnika i zostaną opisane osobno.

Główna logika regulacji jest zaimplementowana w funkcji:

`static void hvac_control_step(float dt_sec)`

która:

1. oblicza błąd regulacji jako różnicę pomiędzy nastawą g_hvac_cfg.setpoint a aktualną temperaturą wyciągu,
2. wywołuje hvac_pid_step() i otrzymuje wirtualne sterowanie pid_out_pct w procentach,
3. przekazuje to sterowanie do funkcji hvac_apply_sequence(), otrzymując trzy sygnały cząstkowe heater_pct, cooler_pct, bypass_pct,
4. przekształca wartości procentowe na odpowiadające im napięcia 0–10 V i wywołuje write_ao_voltage() z odpowiednimi kanałami z g_hvac_cfg.io.

Funkcja:

```
static void hvac_apply_sequence(float pid_out_pct,
                                const struct hvac_config *cfg,
                                float *heater_pct,
                                float *cooler_pct,
                                float *bypass_pct)
```

odpowiada za odwzorowanie pojedynczego sygnału sterującego na poszczególne wykonawcze. Wykorzystuje ona strukturę hvac_seq_cfg, gdzie każde pasmo (cooling, heating, heat_recovery, deadband) opisane jest przedziałem from_percent–to_percent. Algorytm:

- w pierwszej kolejności sprawdza, czy sygnał pid_out_pct znajduje się w paśmie deadband; w takim przypadku wszystkie wyjścia pozostają wyłączone,
- jeśli sygnał jest dodatni i mieści się w paśmie grzania, proporcjonalnie do odległości od from_percent i to_percent wyznacza udział sygnału heater_pct,
- jeśli sygnał jest ujemny i mieści się w paśmie chłodzenia, analogicznie wyznacza cooler_pct,
- w zależności od konfiguracji pasma heat_recovery może zdecydować o stopniu otwarcia bypassu wymiennika (bypass_pct), np. dla wartości sterowania w określonym podprzedziale wokół zera.

Dzięki takiemu podejściu logika sekwencyjna jest w pełni opisana w konfiguracji JSON i może być modyfikowana bez zmiany kodu – np. przesunięcie granicy pomiędzy strefą grzania a deadbandem wymaga jedynie edycji pól "from_percent" i "to_percent" w pliku konfiguracyjnym.

Interfejs użytkownika w LVGL

Interfejs użytkownika składa się z trzech ekranów:

- **Dashboard** – ekran główny wizualizujący nastawę i pasma sekwencji,
- **I/O Visualiser** – ekran prezentujący aktualne wartości napięć na kanałach AI/AO,
- **Configuration Loader** – ekran odpowiedzialny za ładowanie wybranej konfiguracji HVAC.

Wszystkie ekrany korzystają ze wspólnego nagłówka, tworzonego funkcją:

```
static void create_header(lv_obj_t *parent, const char *title_text)
```

Nagłówek zawiera:

- poziomy kontener z przyciskami nawigacyjnymi: `Dashboard`, `I/O Visualiser`, `Config Loader` tworzonymi w funkcji `create_nav_buttons()`,
- etykietę z tytułem ekranu wyrównaną w lewym górnym rogu (`lv_label_set_text(title, title_text)`).

Każdy z przycisków nawigacyjnych ma przypisany callback zdarzenia `LV_EVENT_CLICKED`, który za pomocą funkcji `lv_scr_load()` przełącza aktualny ekran na `screen_dashboard`, `screen_io` lub `screen_config`.

Ekran Dashboard Funkcja:

```
static void create_dashboard_screen(void)
```

tworzy ekran główny, w którym:

- w górnej części, bezpośrednio pod nagłówkiem, umieszczono kontener `row_sp` z informacją o aktualnej nastawie temperatury; etykieta `setpoint_label` jest aktualizowana przez `hvac_refresh_dashboard_setp` i przyjmuje formę "`Setpoint: XX C`",
- po prawej stronie wiersza znajdują się dwa przyciski "-" i "+" z przypisanymi callbackami `on_btn_setpoint_minus()` i `on_btn_setpoint_plus()`, które odpowiednio dekrementują i inkrementują wartość `g_hvac_cfg.setpoint` oraz odświeżają etykietę,
- poniżej utworzony jest zestaw wierszy odpowiadających pasmom sekwencji (`HVAC_SEQ_IDX_COUNT`), każdy zawierający:
  - etykietę z nazwą pasma (tablica `hvac_seq_band_names[]`: "`Cooling`", "`Heating`", "`Heat recovery`", "`Deadband`"),
  - dwa podkontenery `from_cont` i `to_cont`, w których znajdują się:
    - etykieta "From" / "To",
    - przyciski "-" i "+",
    - etykieta z aktualną wartością (np. "`-30 %`", "`60 %`").

Każdy z przycisków modyfikujących granice pasm ma przypisany kontekst `struct seq_btn_ctx` (tablica `g_seq_btn_ctx[HVAC_SEQ_IDX_COUNT][2][2]`), opisujący:

- indeks pasma (`band_index`),
- to, czy modyfikowany jest początek (`is_from = 1`) czy koniec pasma (`is_from = 0`),
- kierunek zmiany (`delta = -1` lub +1).

Callback `on_seq_band_adjust()` pobiera kontekst przez `lv_event_get_user_data(e)` i dokonuje zmiany odpowiedniego pola `from_percent` lub `to_percent`, ograniczając wynik do zakresu $[-100, 100]$ oraz zapewniając podstawową spójność (`from_percent` nie może przekroczyć `to_percent` i odwrotnie). Po każdej zmianie wywoływana jest funkcja `hvac_refresh_dashboard_seq_labels()`, która:

- aktualizuje napisy z wartościami procentowymi granic pasm,
- ukrywa wiersze, dla których `from_percent == to_percent` (nieaktywne pasmo), za pomocą flagi `LV_OBJ_FLAG_HIDDEN`.

Ekran I/O Visualiser Funkcja:

`static void create_io_visualiser(lv_obj_t *parent)`

tworzy przewijany pionowo ekran do podglądu kanałów analogowych. W tym celu:

- wyłączana jest przewijalność obiektu rodzica (`lv_obj_clear_flag(parent, LV_OBJ_FLAG_SCROLLABLE)`),
- tworzony jest podrzędny kontener `scroll` o wysokości 80% ekranu, z włączoną przewijalnością w osi pionowej (`lv_obj_set_scroll_dir(scroll, LV_DIR_VER)`) i z wykorzystaniem układu flex kolumnowego,
- na szczycie kontenera umieszcza się etykietę tytułową "Analog I/O (AI1–AI8 / AO1–AO8)",
- dla każdego indeksu kanału `i` tworzony jest wiersz, w którym znajdują się:
  - część AI: etykieta "`AI1...AI8`", etykieta wartości (np. "`0.00`"), etykieta jednostki "V" oraz etykieta nazwy roli (np. "`T extract`"), przechowywane w tablicach `ai_value_labels[]`, `ai_unit_labels[]` oraz `ai_name_labels[]`,
  - część AO: analogiczny zestaw etykiet dla kanałów `AO1...AO8`, przechowywany w tablicach `ao_value_labels[]`, `ao_unit_labels[]` oraz `ao_name_labels[]`.

  Funkcja:

`void hvac_update_io_values(void)`

odczytuje aktualne napięcia z funkcji `read_ai_voltage(i)` i `read_ao_voltage(i)`, formatuje je do postaci tekstowej z dwoma miejscami po przecinku (`"%.2f"`) i wpisuje do odpowiednich etykiet. Po załadowaniu nowej konfiguracji `hvac_apply_config()` wywołuje funkcję:

`static void hvac_refresh_io_role_labels(void)`

która aktualizuje podpisy ról kanałów AI/AO w oparciu o strukturę `hvac_io_cfg`. Jeżeli dla danego kanału przewidziano funkcję (np. czujnik temperatury zewnętrznej, falownik wentylatora), etykiety przyjmują odpowiednią nazwę (`"T supply"`, `"Fan VFD"`); w przeciwnym wypadku wyświetlany jest opis "`Unused`". Dzięki temu użytkownik widzi od razu, które kanały są wykorzystywane w danej konfiguracji.

Ekran Configuration Loader Funkcja:

`static void create_config_screen(void)`

odpowiada za utworzenie ekranu "Configuration Loader". Pod nagłówkiem umieszczony jest kontener z dwoma przyciskami:

- "`Load Config 1`" z przypisanym callbackiem `on_btn_load_cfg1()`,
- "`Load Config 2`" z callbackiem `on_btn_load_cfg2()`.

Każdy z callbacków tworzy lokalną strukturę `struct hvac_config tmp`, wywołuje `hvac_load_config_from_json()` z odpowiednią tablicą znakową (`hvac_config1_json` lub `hvac_config2_json`), a następnie, w przypadku powodzenia, wywołuje `hvac_apply_config(&tmp)`. Na ekranie znajduje się również etykieta `config_status_label`, która informuje użytkownika o rezultacie: "`Loaded Config 1`"/"`Loaded Config 2`" lub komunikat błędu ("`Error loading Config X`"). Taki ekran stanowi prototyp docelowego mechanizmu wyboru pliku konfiguracyjnego z karty SD.

Główna pętla programu i konfiguracja Zephyra

Ostatnim elementem architektury firmware'u jest główna funkcja:

`int main(void)`

W jej ramach wykonywane są następujące kroki:

1. Inicjalizacja logowania i wypisanie komunikatu startowego:

   ```
   LOG_INF("Starting LVGL UI with JSON config loader + HVAC PI controller (placeholders IO)")
   ```

2. Pobranie uchwytu do urządzenia wyświetlacza za pomocą `DEVICE_DT_GET(DT_CHOSEN(zephyr_display))` oraz weryfikacja jego gotowości funkcją `device_is_ready()`. W przypadku błędu wypisywany jest stosowny log, a funkcja `main` zwraca kontrolę do systemu.

3. Wywołanie funkcji inicjalizujących interfejs:

   - `create_dashboard_screen()`,
   - `create_io_visualiser(screen_dashboard)` lub odpowiednio z innym rodzicem,
   - `create_config_screen()`,

   po czym wywoływana jest `lv_scr_load(screen_dashboard)` w celu ustawienia ekranu Dashboard jako domyślnego.

4. Wyłączenie wygaszania wyświetlacza przez `display_blanking_off(display_dev)` (wymuszenie stale włączonego podświetlenia).

5. Wejście do nieskończonej pętli:

   ```
   while (1) {
       lv_timer_handler();
       k_msleep(10);

       int64_t now_ms = k_uptime_get();
       int64_t diff_ms = now_ms - last_ctrl_ms;

       if (diff_ms >= 100) {
           float dt_sec = diff_ms / 1000.0f;
           hvac_control_step(dt_sec);
           last_ctrl_ms = now_ms;
       }

       // hvac_update_io_values();
   }
   ```

Wywoływanie `lv_timer_handler()` co 10 ms zapewnia płynną obsługę interfejsu graficznego: odświeżanie ekranu, animacje oraz reakcje na dotyk. Jednocześnie za pomocą funkcji `k_uptime_get()`

utrzymywany jest interwał dla pętli regulacji na poziomie ok. 100 ms, co odpowiada częstotliwości 10 Hz – w zupełności wystarczającej dla wolnozmiennych procesów HVAC. Wyodrębnienie `hvac_control_step()` jako niezależnej funkcji pozwala w przyszłości łatwo przenieść ją do osobnego wątku Zephyra, jeśli pojawi się potrzeba równoległego wykonywania bardziej złożonych obliczeń.

Komentowana linia `// hvac_update_io_values();` wskazuje miejsce, w którym w kolejnych iteracjach projektowych można okresowo odświeżać wartości na ekranie I/O Visualiser (po podłączeniu rzeczywistych przetworników A/C i C/A).

W pliku `prj.conf` skonfigurowano rozmiary stosów (`CONFIG_MAIN_STACK_SIZE=8192`, `CONFIG_SYSTEM_WORKQUEUE_S`
oraz rozmiar puli sterty (`CONFIG_HEAP_MEM_POOL_SIZE=16384`), co zapewnia rezerwę pamięci zarówno dla LVGL (bufor ramki `CONFIG_LV_Z_VDB_SIZE`), jak i biblioteki JSON oraz struktur danych aplikacji. Włączono także `CONFIG_ASSERT` oraz `CONFIG_LOG` z domyślnym poziomem logowania `3`, co ułatwia diagnozowanie błędów już na etapie rozwoju firmware'u.

Podsumowanie części programowej

Zaprojektowany firmware łączy w sobie cechy aplikacji edukacyjnej oraz prototypu rzeczywistego sterownika HVAC: z jednej strony udostępnia czytelny, wieloekranowy interfejs graficzny dla użytkownika, z drugiej – implementuje parametryczną logikę regulacji opartą o regulator PI i konfigurowalne pasma sekwencji. Wykorzystanie Zephyr RTOS pozwoliło skorzystać z gotowych bloków funkcjonalnych (wyświetlacz, logowanie, JSON), a biblioteka LVGL zapewniła elastyczny sposób budowy interfejsu HMI. Model konfiguracji w formacie JSON oraz wyraźne rozdzielenie warstw (sprzęt–regulacja–prezentacja) sprawiają, że system jest łatwy do dalszej rozbudowy i utrzymania, a także przystosowany do przyszłego przeniesienia konfiguracji do zewnętrznych plików przechowywanych na karcie SD.

3.3. Walidacja

3.3.1. Metodyka

Źródła wzorcowe (kalibrator napięcia), obciążenie dla wyjścia 0–10 V, środowisko testowe.

3.3.2. Wyniki

Tabele dokładności, histogram odchyleń, niepewność typu A/B, budżet niepewności.

3.3.3. Dyskusja

Ograniczenia, dryft temperaturowy, histereza, propozycje ulepszeń.

BIBLIOGRAFIA

[1] *1N4728A Thru 1N4764A 1 W Zener Diodes*. Diody Zenera serii 1N47xxA stosowane w torze ochrony wejścia zasilania (D4). GOOD-ARK Electronics. 2024. URL: https://www.goodark.com/specification/1N4728%20thru%201N4764.pdf.

[2] *ADS8688 12-Bit, 500-kSPS, 8-Channel Data Acquisition System*. Zewnętrzny ADC U7. Texas Instruments. 2021. URL: https://www.ti.com/lit/ds/symlink/ads8688.pdf.

[3] Abdoalnasir Almabrok, Marios Psarakis i Anastasios Dounis. „Fast Tuning of the PID Controller in an HVAC System Using the Big Bang–Big Crunch Algorithm and FPGA Technology". W: *Algorithms* 11.10 (2018), s. 146. DOI: 10.3390/a11100146. URL: https://doi.org/10.3390/a11100146.

[4] *Ambient Temperature Sensor-PT1000 with 0-10V Output*. Czujnik temperatury otoczenia z przetwornikiem PT1000 na sygnał 0–10 V. Seven Sensor Solutions. 2023. URL: https://www.sevensensor.com/ambient-temperature-sensor-pt1000-u.

[5] *AO4407A P-Channel Enhancement Mode Field Effect Transistor*. Tranzystor P-MOSFET mocy do ochrony przed odwrotną polaryzacją (Q1). Alpha i Omega Semiconductor. 2024. URL: https://www.aosmd.com/res/data_sheets/AO4407A.pdf.

[6] *Application Guidelines for 0–10 V Control Interfaces in HVAC*. ASHRAE. 2019.

[7] ASHRAE. *ASHRAE Guideline 36-2021: High-Performance Sequences of Operation for HVAC Systems*. Guideline 36-2021. American Society of Heating, Refrigerating i Air-Conditioning Engineers. 2021. URL: https://www.ashrae.org/technical-resources/bookstore/guideline-36-2018.

[8] Bonnie C. Baker. *Analog and Interface Guide – Volume 1: A Compilation of Technical Articles and Design Notes*. Zbiór not aplikacyjnych o projektowaniu układów analogowych i PCB. Microchip Technology Inc. 2005. URL: https://ww1.microchip.com/downloads/en/devicedoc/00924b.pdf.

[9] *Considerations for Mixed Signal Circuit Board Design (AN-404)*. Application Note AN-404. Analog Devices, Inc. 2000. URL: https://www.analog.com/media/en/technical-documentation/application-notes/495266810an-404.pdf.

[10] *DAC7568 12-Bit, Octal, Buffered Voltage Output DAC*. Zewnętrzny DAC U3. Texas Instruments. 2017. URL: https://www.ti.com/lit/ds/symlink/dac7568.pdf.

[11] A. C. Fernandes i in. „Control of airflow in ventilation systems using embedded systems on microcontrollers". W: *Microsystem Technologies* 25.9 (2019), s. 3597–3608. DOI: 10.1007/s00542-019-04407-1. URL: https://doi.org/10.1007/s00542-019-04407-1.

[12] *HD & HO Series Deluxe Humidity Transmitters*. Aktywne czujniki wilgotności/temperatury z wyjściem 0–5 V / 0–10 V. Veris Industries. 2014. URL: https://www.veris.com/ASSETS/DOCUMENTS/ITEMS/EN/HD_HO_d0321.pdf.

[13] *High Accuracy Pressure Transmitter, 0–10 V Output*. Nadajnik ciśnienia z wyjściem napięciowym 0–10 V do systemów automatyki. Jakar Electronics. 2020. URL: https://www.jakar.cz/en/p/high-accuracy-pressure-transmitter/4103.

[14] V. Kirubakaran i in. „Energy efficient model based algorithm for control of building HVAC systems”. W: *Ecotoxicology and Environmental Safety* 121 (2015), s. 236–243. DOI: 10.1016/j.ecoenv.2015.03.027. URL: https://doi.org/10.1016/j.ecoenv.2015.03.027.

[15] *LM2596 SIMPLE SWITCHER 3-A Step-Down Voltage Regulator*. Regulatory zasilania U1 (5 V) i U2 (3.3 V). Texas Instruments. 2020. URL: https://www.ti.com/lit/ds/symlink/lm2596.pdf.

[16] *LVGL – Light and Versatile Embedded Graphics Library*. https://lvgl.io/. Oficjalna strona projektu LVGL, dostęp: 2025-12-03. 2025.

[17] Henry W. Ott. „Partitioning and Layout of a Mixed-Signal PCB”. W: *Printed Circuit Design* 18.6 (2001). Klasyczny artykuł o podziale mas i prowadzeniu ścieżek w układach mieszanych, s. 16–26. URL: https://www.hottconsultants.com/pdf_files/mixed-signal.pdf.

[18] Guangji Qu i Mohammed Zaheer-uddin. „Real-time tuning of PI controllers in HVAC systems”. W: *International Journal of Energy Research* 28.15 (2004), s. 1313–1327. DOI: 10.1002/er.1030. URL: https://doi.org/10.1002/er.1030.

[19] *STM32F746xG Datasheet*. MCU zastosowany na płytce STM32F746G-Discovery. STMicroelectronics. 2023. URL: https://www.st.com/en/microcontrollers-microprocessors/stm32f746ng.html.

[20] STMicroelectronics. *32F746GDISCOVERY — Discovery kit with STM32F746NG MCU*. 2025. URL: https://www.st.com/en/evaluation-tools/32f746gdiscovery.html (term. wiz. 10. 10. 2025).

[21] *TLVx9304 Low-Noise, Rail-to-Rail Output Operational Amplifiers*. Bufor analogowy U5, U6. Texas Instruments. 2023. URL: https://www.ti.com/lit/ds/symlink/tlv9304.pdf.

[22] *Zephyr Integration Guide – LVGL Documentation*. https://docs.lvgl.io/9.1/integration/os/zephyr.html. Opis integracji LVGL z Zephyr RTOS, dostęp: 2025-12-03. 2024.

[23] *Zephyr Project Documentation*. https://docs.zephyrproject.org/latest/. Dokumentacja systemu Zephyr RTOS, dostęp: 2025-12-03. 2025.

[24] *Zephyr Project RTOS*. https://www.zephyrproject.org/. Dostęp: 2025-12-03. 2025.