



Akademia Górniczo-Hutnicza
im. Stanisława Staszica w Krakowie.
Wydział Informatyki, Elektroniki i Telekomunikacji

Sensory w aplikacjach wbudowanych

SPRAWOZDANIE

Rok I, Systemy wbudowane, Elektronika i Telekomunikacja IIst.

Temat: Moduł z sensorami oparty na procesorze STM32F103, zgodny z wyprowadzeniami Mikrobus™

Zespół: 1. Mateusz Kozyra 2. Mirosław Wiącek 3. Radosław Sajdak	Oceny indywidualne: 1. 2. 3.
Data oddania sprawozdania:	Ocena sprawozdania:
Uwagi prowadzącego zajęcia:	Informacje dodatkowe:

Spis treści

1. Wstęp.....	3
1.1. mikroBUS™.....	3
1.2. STM32F103.....	4
1.3. STS30	5
1.4. BMP280.....	5
1.5. MQ2.....	5
1.6. Makefile	5
1.7. Github.....	5
2. Wykonanie projektu	7
2.1. Projekt płytka	7
2.1.1. Schemat.....	7
2.1.2. Layout	13
2.1.3. Popełnione błędy oraz wykonane przeróbki	16
2.2. Oprogramowanie	22
2.2.1. Biblioteki peryferiów	22
2.2.2. API.....	23
2.2.3. Lista dostępnych funkcji.....	25
2.2.4. System kontroli wersji	26
3. Prezentacja działania.....	27
4. Wnioski	31

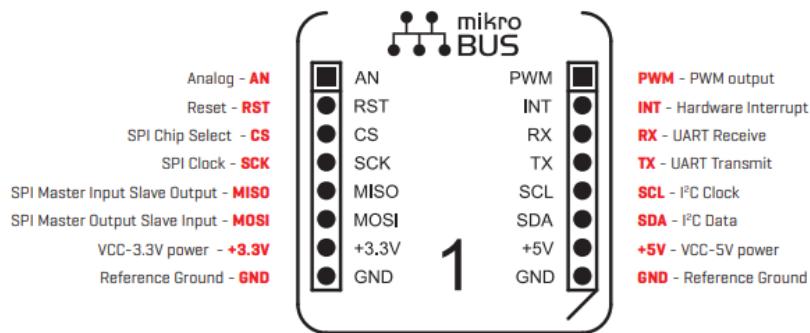
1. Wstęp

Założeniem projektu, było stworzenie modułu sensora, zgodnego ze standardem mikroBUS™. Zespół, korzystając z okazji na zdobycie nowych umiejętności, zdecydował się rozszerzyć założenia projektowe o dwa dodatkowe sensory (co daje łącznie trzy układy pomiarowe na jednej płytce). Układy te, komunikują się z mikroprocesorem STM32F103 znajdującym się na tej samej płycie. W procesorze, dane są przetwarzane, nakładając dodatkową warstwę abstrakcji dla obsługi sensorów. Z płytą, można komunikować się przy użyciu interfejsu UART. Dodatkowym założeniem projektu, było różne podejście do tworzenia oprogramowania, każdego z członków zespołu. Zdecydowano, że jeden będzie tworzyć kod, testując go z mikroprocesorem STM32F4, kolejny bezpośrednio na wykonanej płycie, a ostatni, bez bezpośredniej styczności ze sprzętem. Podejście to, pozwoliło spojrzeć na projekt z innej perspektywy, jak gdyby zespół był większy i rozproszony. Cały proces tworzenia był kontrolowany przy użyciu platformy Github, pozwalającej na wygodną kontrolę wersji oraz dyskusję nad kolejnymi zmianami w kodzie. W niniejszym sprawozdaniu, skupiono się przede wszystkim na procesie tworzenia modułu oraz popełnionych błędach, nie pomijając krótkiego wstępu teoretycznego. Całość, zakończona jest wnioskami na temat procesu oraz zdobytych umiejętności.

1.1. mikroBUS™

MikroBus™ jest standardem definiującym układ gniazd oraz wyprowadzeń na płytce, zawierającej układ scalony (np. sensor). Standard opisuje również dokładne wymiary płytka, a także narzuca warstwę nadruków na płytce. Założeniem mikroBUS™, jest umożliwienie szybkiego i łatwego rozbudowywania układów na rzecz prototypowania. Dostarczane przez wielu producentów płytki z gotowymi gniazdami, pozwalają w prosty sposób uruchomić dowolny sensor, a następnie stworzyć dla niego oprogramowanie. Tym samym, mikroBUS™ okazuje się świetną alternatywą dla niestabilnych płyt stykowych. Na rysunku 1.1 przedstawiono zdefiniowany w standardzie pinout.

Ze względu na zmienione założenia projektu, stworzona płytka zgadza się ze standardem co do wyprowadzeń i wymiarów, jednak nie posiada naniesionych nadruków. Wynika to wprost z braku miejsca, na płytce. Dodatkowo, kilka sensorów na jednej płytce, również nie jest zgodne ze standardem.



Rys. 1.1. Specyfikacja wyprowadzeń mikroBUS™[1]

1.2. STM32F103

Do stworzenia projektu, wybrano procesor STM32F103C8 z rdzeniem ARM®Cortex®-M3 [2]. Wybór, podyktowany był przede wszystkim dostępnością, ponieważ układ ten, w obudowach LQFP48 znajdował się w prywatnych zasobach zespołu. Obecnie, na rynku dostępne jest niewiele procesorów, a części elektroniczne są drogie. Z tego powodu, zdecydowano się korzystać z tego, co było dostępne. Nie oznacza to jednak, że procesor ten był pod jakimkolwiek względem ograniczający. Najistotniejszymi z jego cech są:

- 2 interfejsy I²C
- 3 interfejsy USART
- 2 interfejsy SPI
- Zasilanie 2.0 - 3.6V
- Zewnętrzny zegar 4 - 16MHz
- 12-bitowe, wielokanałowe przetworniki A/D

Układ, programowany jest przy użyciu interfejsu SWD oraz programatora ST-link. Fakt ten, znaczco ułatwił wykonanie działającego układu. Ze względu na przyjęte założenia i wybrane sensory, wybrany procesor bardzo dobrze wpasowywał się w projekt. Dla wykonania oprogramowania, istotne były również dostarczone przez ST biblioteki oraz oprogramowanie STM32CubeMX, wprowadzające wysoki poziom abstrakcji w tworzeniu oprogramowania. Pozwoliło to uniknąć często czasochłonnego, ręcznego pisania do rejestrów procesora, w celu konfiguracji peryferiów. Były również momenty, w których HAL okazał się problemem, co wspomniane zostało w dalszej części sprawozdania.

1.3. STS30

Sensor STS30-DIS marki Sensirion to cyfrowy, precyzyjny sensor temperatury. Wybraliśmy go jako przykładowy sensor działający po I²C, oraz będący bardzo dobrze udokumentowany przez producenta. Oferuje dokładność 0.2 °C w zakresie temperatur 0-60 °C, oraz funkcje takie jak wbudowane ogrzewanie, rejestr statusu, zmienny adres, soft reset po I²C i wiele innych[3]. Wszystkie dane z sensora są weryfikowane CRC-8.

1.4. BMP280

Krótki opis sensora Matiego

1.5. MQ2

Krótki opis sensora Matiego

1.6. Makefile

Jak wspomniano we wstępnie, jednym z założeń, było tworzenie oprogramowania równolegle, na dwóch różnych procesorach. STM32F1 i STM32F4. Procesory te, należąc do zupełnie różnych rodzin, korzystają z innych bibliotek, a do ich konfiguracji niejednokrotnie wymagane są różne kroki. Z tego powodu, w projekcie wykorzystano program *Make*. Pozwala on, na podstawie pliku konfiguracyjnego *makefile*, zdefiniować sposób komplikacji oraz ścieżki bibliotek wykorzystywanych przez kod. Narzędzie to, jest stosunkowo proste w konfiguracji, a jej przykłady są ogólnodostępne w internecie. Dodatkowym argumentem przemawiającym za jego wykorzystaniem, była chęć jego przetestowania w praktyce.

1.7. Github

Github, to serwis internetowy pozwalający przechowywać projekty z wykorzystaniem systemu kontroli wersji. Korzysta on z systemu Git opublikowanego na licencji GNU GPL 2 (wolne i otwarte oprogramowanie). System ten, jest obecnie wykorzystywany przez wielu programistów, a jego znajomość często wymagana jest przez pracodawców. Github, poza kontrolą wersji, dostarcza wygodne i przejrzyste środowisko w przeglądarce, pozwalające programistom na np. przegląd kodu współpracowników. Fakt ten, został wykorzystany podczas tworzenia projektu. Pozwoliło to nie tylko na poprawienie jakości tworzonego oprogramowania, ale również skonfrontowanie różnych podejść do współpracy nad kodem.

2. Wykonanie projektu

2.1. Projekt płytki

2.1.1. Schemat

Projekt płytki, naturalnie rozpoczęto od wykonania schematu ideowego. W tym celu przyjęto następujące założenia:

- Układ, musi być zasilany zewnętrznie (5v oraz 3.3V)
- Wykorzystany zostanie konwerter UART-USB
- Układ, będzie można wyłączyć tranzystorem, sterującym przetwornicami
- Wykorzystany zostanie zewnętrzny generator

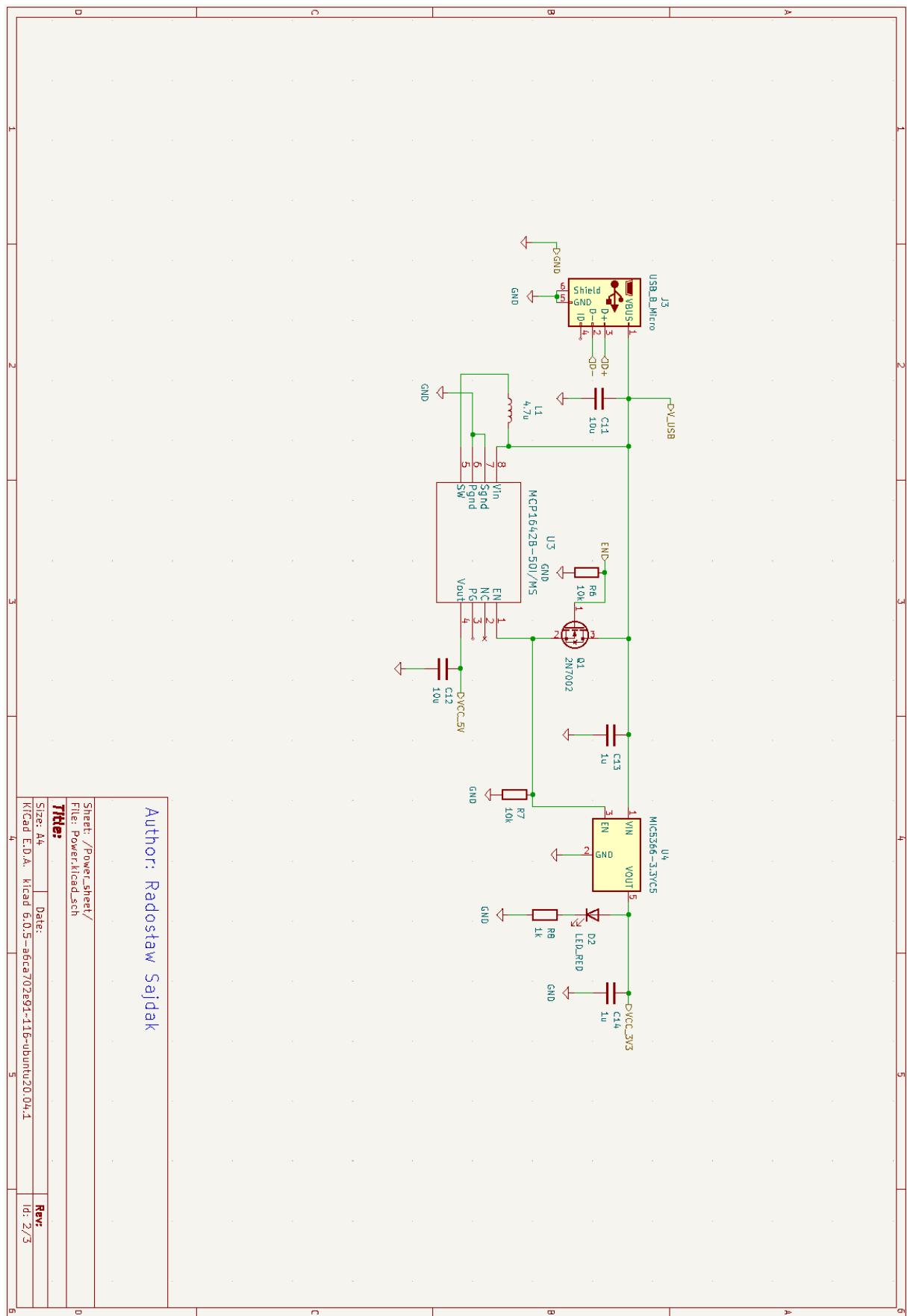
Według standardu mikroBUSTTM, płytka zasilana jest przez układ w który jest wpięta. Często jest to zasilanie procesora, na którym tworzone jest oprogramowanie. Ponieważ w opisywanym projekcie, wykorzystany został procesor, oraz trzy sensory (w tym jeden pracujący z napięciem 5V), zdecydowano się na zasilanie zewnętrzne. Miało ono zminimalizować prawdopodobieństwo nieprawidłowej pracy układu. Jednocześnie, czujnik MQ2 pobierając setki mA prądu, mógłby uszkodzić przetwornicę układu w który została wpięta płytka. Najprostszym rozwiązaniem, było wyprowadzenie portu USB i zasilanie z niego układu. Zwrócono uwagę na fakt, że USB w komputerach, nie ma stabilnego napięcia 5V, a często wręcz 4.7-4.8V. Jest to zachowanie zdefiniowane w standardzie USB 2.0 [4] w rozdziale 7. Z tego powodu, należało wykorzystać regulator napięcia.

Wybór tego i wszystkich kolejnych komponentów, podkutowany był w dużej mierze dostępnością na rynku. Kolejnym kryterium, były parametry układu, opisywane w dostarczanych przez producentów dokumentacjach. Jako układ do zasilania sensora MQ2, wykorzystano przetwornicę typu BOOST - MCP1642 o wyjściowym napięciu właśnie 5V. Pozwoliła ona na stabilne zasilenie czujnika. Jej ważną cechą, jest maksymalny prąd na wyjściu, o wartości 800mA[5]. Wysoki prąd jest tutaj niezbędny, ponieważ zasilanie wspomnianego czujnika wymaga nawet 200mA prądu stałego. Do zasilania reszty elementów, wykorzystano regulator LDO MIC5365 o wyjściowym napięciu 3.3V[6]. Gwarantowany prąd wyjściowy regulatora to 150mA, co jest wartością wystarczającą do zasilenia reszty elementów z zapasem. Całość, z założenia włączana miała być tranzystorem NMOS 2N7002. Bramka tego tranzystora,

połączona jest z pinem RESET standardu mikroBUS™, pozwalając użytkownikowi wyłączyć układ. W przypadku tego elementu, pojawiły się dwa błędy, wymagające przerobienia gotowej płytki. Problem, szerzej opisano w podrozdziale 2.1.3.

Schemat oraz layout, wykonane zostały w programie KiCad. Opensourcowym oprogramowaniu dla różnych systemów operacyjnych. Program ten, umożliwia podział schematu na bloki, co wykorzystano przenosząc sekcję zasilania do osobnego schematu. Na schemacie 2.1 przedstawiono całą, wspomnianą sekcję. Zasilanie, doprowadzone jest z USB do obu przetwornic. Gdy procesor jest zasilony, a więc działa przetwornica 3.3V, włączona zostaje czerwona dioda LED. Widoczny na schemacie rezystor R10, nie jest uwzględniony w layoucie i został dołożony ręcznie do płytki. Więcej na ten temat, opowiada podrozdział 2.1.3.

Na schemacie, widoczna jest również znacząca ilość kondensatorów. Zostały one dodane do schematu, jako wskazane przez producenta jako konieczne dla poprawnego działania układów. Bardzo istotny jest również rezystor R7, gwarantujący ustalony stan niski po wyłączeniu tranzystora.



Rys. 2.1. Schemat części zasilającej moduł

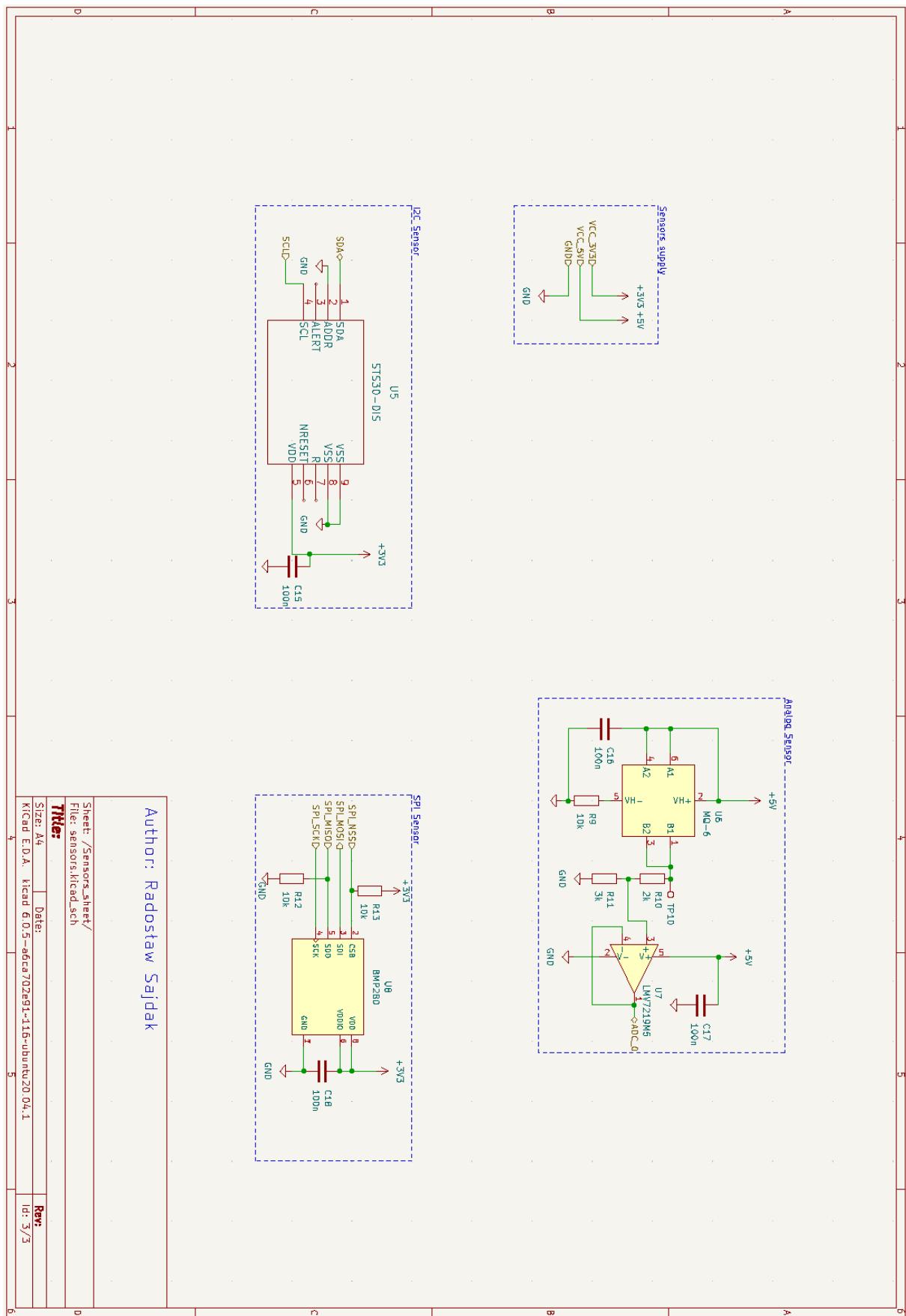
Kolejnym blokiem, jest schemat 2.2, zawierający w sobie wszystkie czujniki. Podobnie jak wcześniej, producenci w swoich dokumentacjach zalecają stosowanie kondensatorów, jak najbliżej zasilania układu, dla zapewnienia jego prawidłowego działania. W przypadku sensora działającego z użyciem SPI - BMP280 - zastosowano rezystor pull-down na linii MISO[7]. Dzięki temu, gdy linia nie jest używana, zagwarantowany jest stan niski. Podobnie w przypadku linii NSS. Zastosowanie rezystora podciągającego do zasilania, pozwala zagwarantować, że układ nie będzie aktywny gdy nie jest to pożądane.

Wartym uwagi jest układ czujnika analogowego MQ2. Ze względu na zastosowany procesor, mierzone przetwornikiem napięcie, musi mieścić się w przedziale 0-3.6V. Ponieważ wybrany czujnik działa w zakresie 0-5V, konieczne jest zastosowanie dzielnika napięcia. W tym przypadku, układ rezystorów oraz czujnika, staje się źródłem prądu, który mógłby uszkodzić procesor. Z tego powodu, zastosowano wzmacniacz operacyjny w konfiguracji nieodwracającego wtórnika napięciowego. Wzmacniacz operacyjny, mający bardzo małą rezystancję wyjściową, stanowi w przybliżeniu źródło napięcia równe co do wartości spadkowi napięcia na R11.

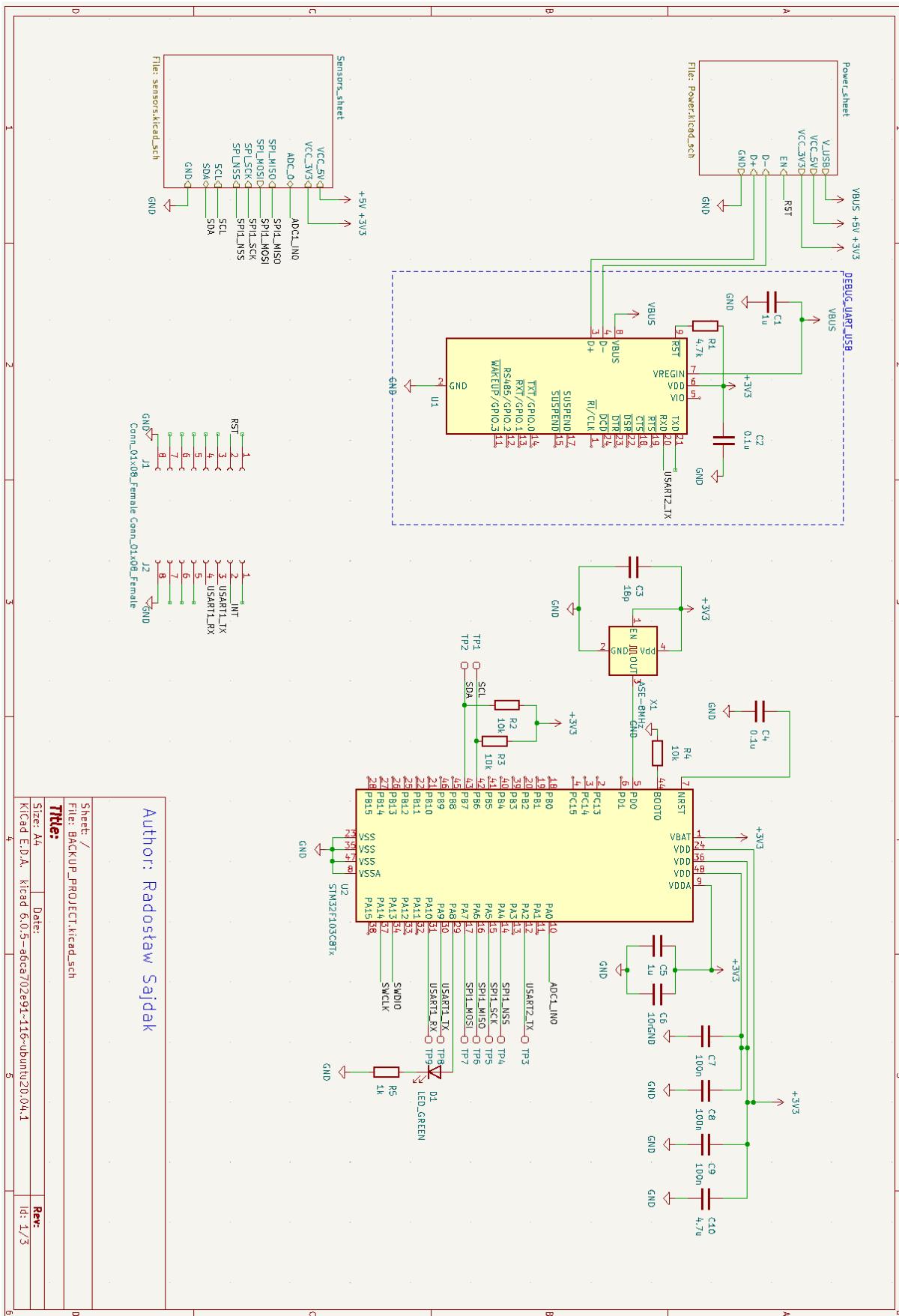
Ostatnim, a zarazem najistotniejszym, jest schemat 2.3. Po jego lewej stronie, zaznaczono fragment odpowiadający za konwersję UART-USB. Wykorzystany układ konwertera to CP2102[8]. Układ ten, pozwala obserwować logi mikroprocesora przy użyciu tego samego przewodu, którym zasilamy płytke. W trakcie tworzenia schematu, rozważano również tylko wyprowadzenie testpointów, pozwalających na podejrzenie logów zewnętrznym konwerterem. Ponieważ jednak planowany jest również inny projekt, wykorzystujący wiele takich konwerterów, zdecydowano się na jego użycie w celach przede wszystkim edukacyjnych i testowych. Głównym zagadnieniem z nim związanym, było użycie pary różnicowej, wymaganej dla prawidłowego godziałania USB.

Po prawej stronie schematu 2.3, widoczny jest blok mikroprocesora STM32F103C8. Zastosowano tutaj zewnętrzny generator o taktowaniu 8MHz, zastępując wewnętrzny zegar mikroprocesora. Istotnym zaganiem, wymaganym przez mikroprocesor według dokumentacji, jest montaż osobnych kondensatorów o wartości 100n przy każdym z pinów zasilania, możliwe jak najbliżej układu. Dla zapewnienia prawidłowego działania magistrali I²C, dodano rezystory podciągające linie do zasilania. Wszystkie piny posiadają też testpointy, mające ułatwić tworzenie oprogramowania dzięki szybkiemu wykryciu błędów w transmisji, a więc np. w złutowaniu układu. Dodatkowo, na pinie 8, wyprowadzono LED. Zgodnie z przyjętymi przez zespół założeniami, wskazuje on gotowość układu do pracy, a więc przejście wszystkich kroków inicjalizacyjnych.

Wartym zaznaczenia jest tutaj fakt nie wyprowadzenia pinów SWDIO oraz SWCLK. Ponieważ są to piny wymagane tylko na etapie tworzenia oprogramowania, podjęto decyzję o ich niewykorzystaniu. Ważną w tym wypadku kwestią, był również brak miejsca na ścieżki, pozwalające na taki zabieg. Niemniej jednak, osoba składająca PCB, bez większych problemów była w stanie włutować przewody programatora bezpośrednio do pinów mikroprocesora.



Rys. 2.2. Schemat połączeń czujników

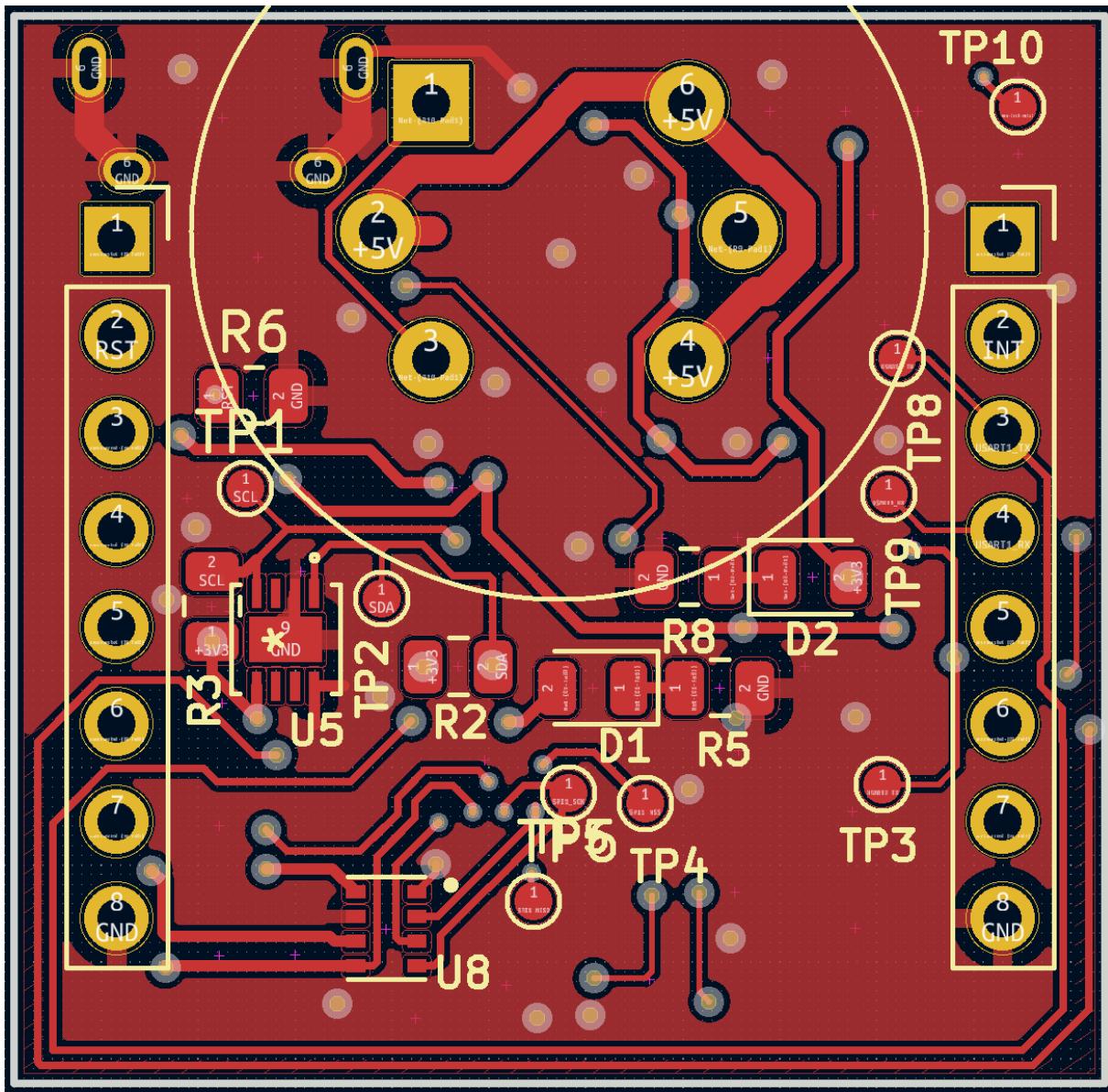


Rys. 2.3. Schemat połączeń konwertera UART-USB oraz mikroprocesora

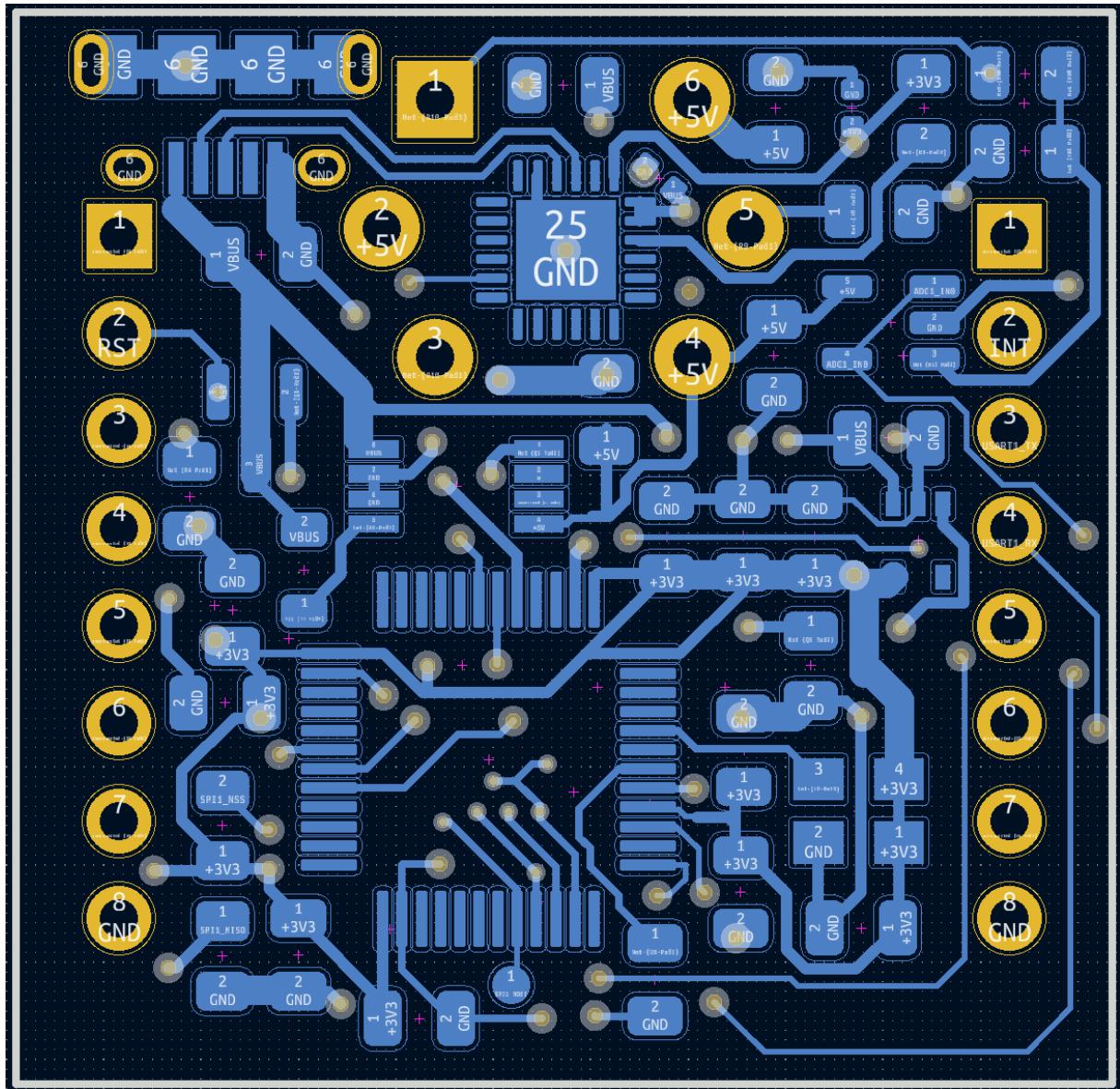
2.1.2. Layout

W kolejnym kroku, wykonany został layout płytki. Obrazek 2.4 przedstawia frontową warstwę elektryczną płytki oraz nadruki na niej. Z założenia, jest to warstwa w większości wypełniona masą (GND). Niestety, ze względu na brak miejsca na warstwie tylnej, masa układu jest znacznie pocięta, choć pozostała jednolita. W przypadku linii 5V, zasilającej czujnik MQ2, zastosowano znacznie szersze ścieżki o szerokości 30mils. Pozostałe, mają szerokość 12 oraz 7,87mils. Druga z tych wartości, jest wartością minimalną, domyślną dla programu KiCad. Mimo ręcznego ustawiania mniejszej szerokości, zgodnej z zaleceniami JLCPCB[9], program automatycznie zmieniał szerokość ścieżki na większą. Podobnie było w przypadku przelotek. Niestety, zorientowano się o tym pod koniec tworzenia płytki, a mogłyby to pomóc oszczędzić miejsce. Szczególnie istotne, jest to dla warstwy sygnałowej, przedstawionej na rysunku 2.5. Ze względu na czytelność, rysunek przedstawia jedynie połączenia elektryczne, bez nadruku. Jako elementy pasywne, wykorzystano te w obudowach 0806 oraz pojedyncze elementy 0402. Również w przypadku tej warstwy, znacznie pogrubiono ścieżki doprowadzające zasilanie. Ma to na celu poprawę przewodności ścieżek, która zależy od ich szerokości. Jak można zauważyć, na płytce jest bardzo gęsto, a część elementów znajduje się na granicy spełnienia warunków DRC (np. odstępów między padami). Ponieważ jednak był to pierwszy projekt płytki autora, popełniono tutaj kilka błędów. Najważniejszym z nich, jest użycie komponentów 0806. Zdecydowano się na nie, ponieważ wszystko lutowane było ręcznie (z użyciem szablonu do pasty, jednak w trakcie wykonywania projektu, większość układów była wielokrotnie ręcznie przelutowywana). Niestety, zajmują one bardzo dużo miejsca na płytce, a obudowy 0603 również dają się łatwo lutować ręcznie. Dobrą zmianą, byłaby również bardziej zwarta sekcja zasilania. Dodatkowe miejsce, mogłyby pozwolić wygodniej umiejscowić komponenty. Również szerokość ścieżek sygnałowych, mniejsza o 1,87mils (0,0254mm) mogłyby pozwolić zaoszczędzić nieco miejsca i lepiej poprowadzić ścieżki.

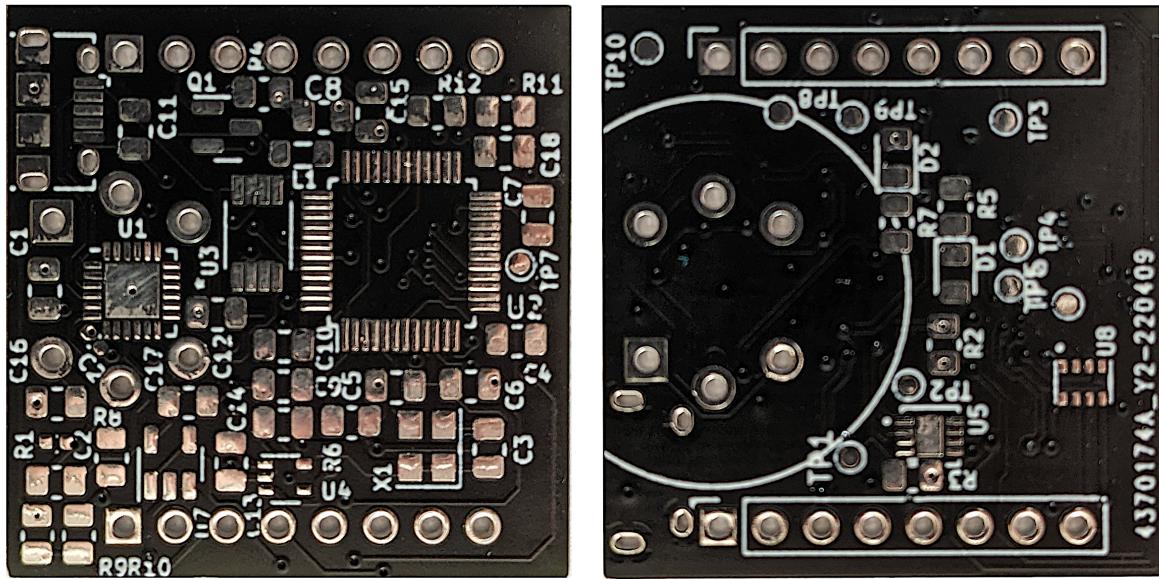
Po wygenerowaniu plików, płytę oraz szablon, zamówiono w serwisie JLCPCB. Jednocześnie, części zamówione w sklepie Mouser. Obrazek 2.6 przedstawia gotową, wykonaną płytę przed przygotowaniem elementów. Natomiast obrazek 2.7 przedstawia zlutowaną, gotową płytę przed wykonaniem niezbędnych przeróbek, omówionych w następnym podrozdziale.



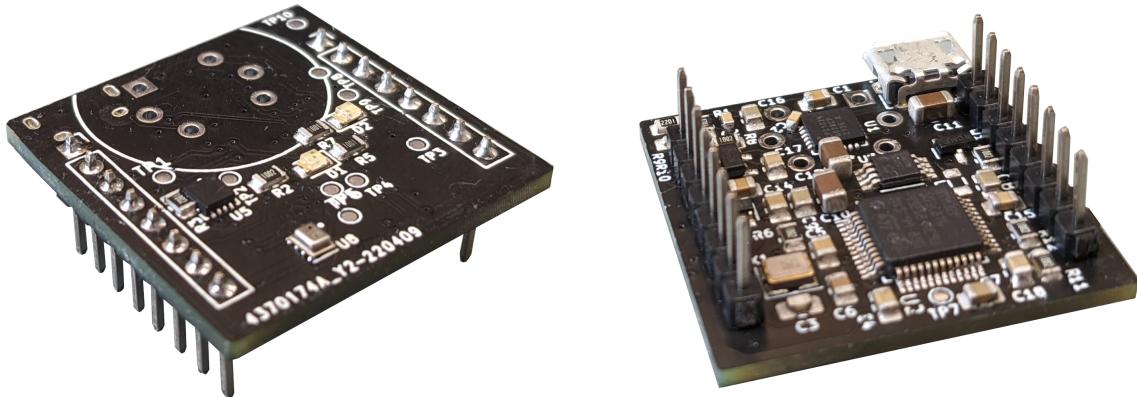
Rys. 2.4. Frontowa warstwa elektryczna płytki



Rys. 2.5. Tylna warstwa sygnałowa płytki



Rys. 2.6. Zdjęcie wykonanej płytki przed lutowaniem elementów



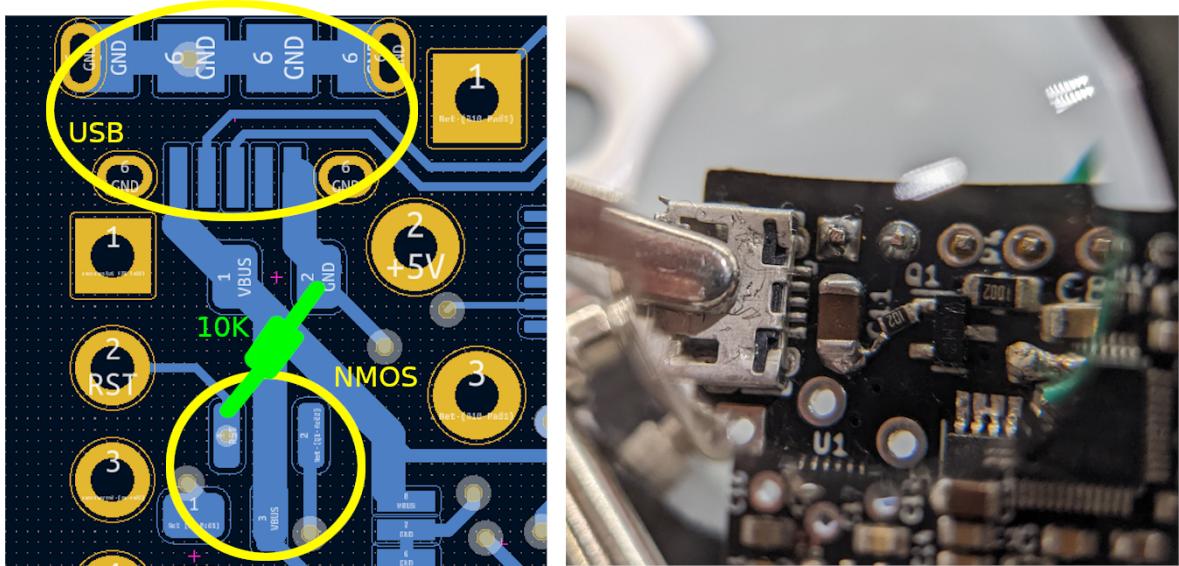
Rys. 2.7. Zdjęcie wykonanej płytki po lutowaniu elementów

2.1.3. Popełnione błędy oraz wykonane przeróbki

Niestety, ze względu na zupełny brak doświadczenia w projektowaniu układów elektronicznych oraz zwykłe niedopatrzenia, popełniony został szereg błędów, wymagający zmian w celu uruchomienia płytki.

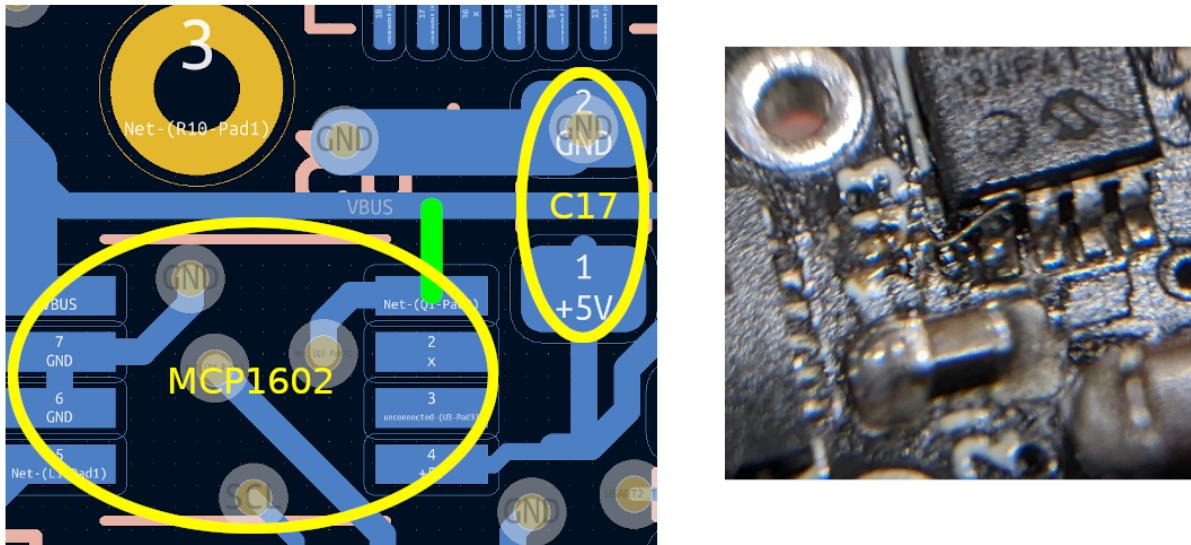
Pierwszym z wykrytych błędów, była bramka tranzystora włączającego przetwornice wisząca w powietrzu. Zaznaczony na schemacie 2.1 rezistor R6, dodany został już po wyprodukowaniu płyt. Ze względu na jego brak, bramka znajdowała się w niaustalonym stanie, a mówiąc kolokwialnie - pływała. Powodowało to włączanie i wyłączenie układu w krótkich odstępach czasu, co uwidoczniło się przez mruganie diody LED po podpięciu zasilania, ale nie doprowadzeniu zasilania na bramkę

tranzystora. Uruchomiony raz układ, nie wyłączał się, ponieważ zgromadzone w bramce ładunki, nie miały gdzie ujść. Rozwiążaniem było właśnie dodanie rezystora R6 ściągającego do masy. Rozwiązanie na fragmencie layoutu oraz złożonej płytce, przedstawia rysunek 2.8. Kolorem żółtym, zaznaczono istotne elementy, pozwalające odnaleźć miejsce na schemacie oraz layoutcie, a kolorem zielonym, dodany element.



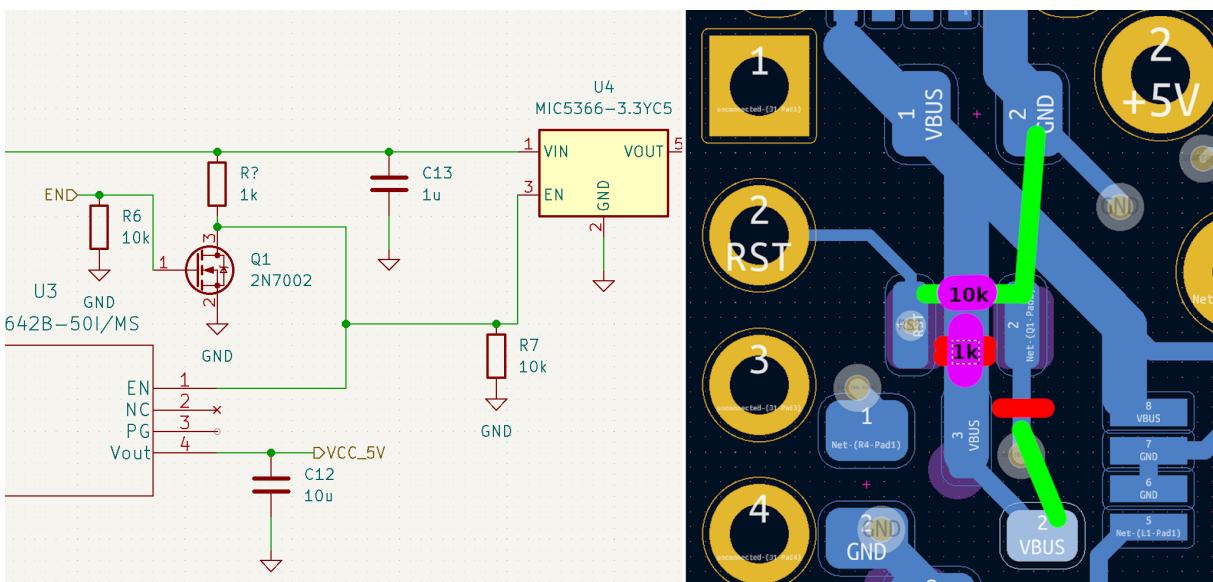
Rys. 2.8. Rezystor ściągający bramkę do masy

Jak się później okazało, wspomniany mosfet generował znacznie więcej problemów. Najważniejszym z nich, było nie włączanie przetwornicy MCP1642. Aby przetwornica mogła być włączona, napięcie na pinie EN, powinno wynosić $0.75 * \text{Vin}[10]$. W przypadku tego układu, wartość ta wynosi 3.5V. Niestety, w stworzonej konfiguracji, napięcie nie przekracza 1.5V. W trakcie realizacji projektu, podjęto kilka różnych prób przerobienia układu w taki sposób, aby przetwornica mogła zadziałać poprawnie. Wszystkie z nich, opisane zostały poniżej. Ostatecznie jednak, zdecydowano podłączyć pin EN na stałe do zasilania USB (Vbus). Nie niesie to za sobą znaczących konsekwencji, ponieważ przetwornica ta, zasila jedynie czujnik MQ2, a ten, jako czujnik analogowy, nie wymaga resetu. Obrazek 2.9 przedstawia zastosowaną poprawkę. Zielonym kolorem zaznaczono miejsce zwarcia ścieżki z pinem przetwornicy.



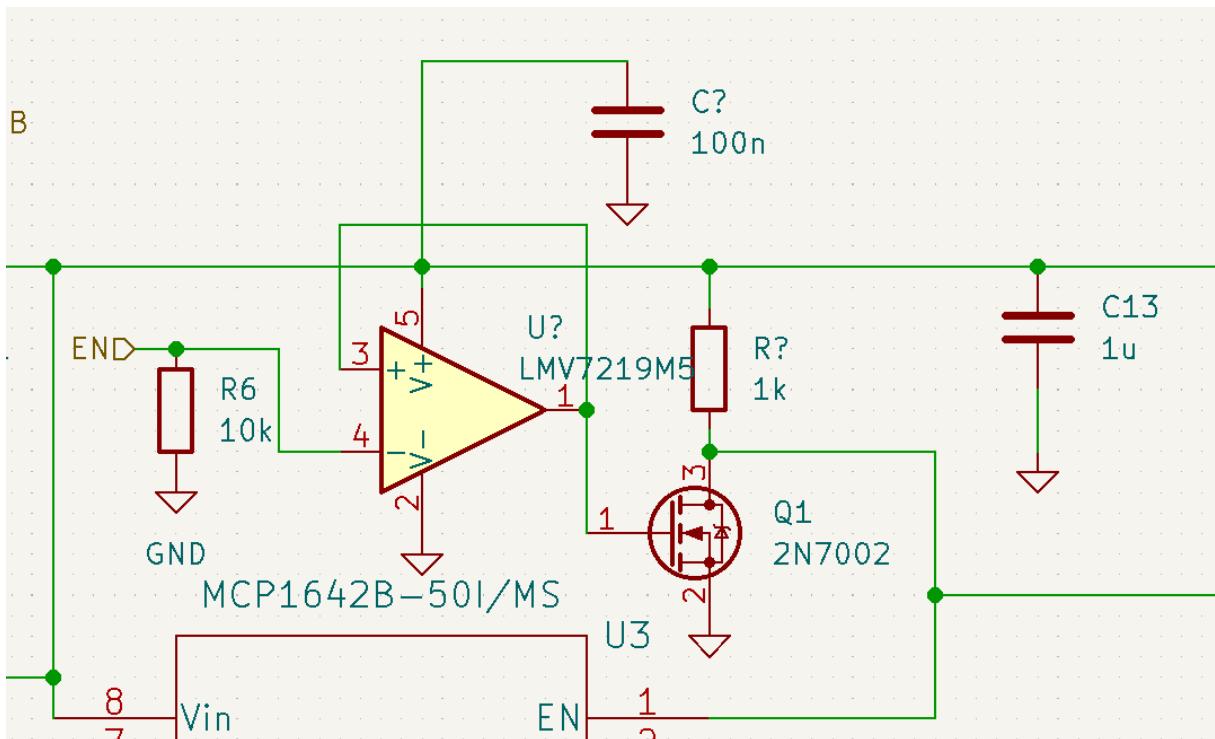
Rys. 2.9. Zwarcie pinu EN ze ścieżką Vbus

Jedną z prób naprawienia układu, było przerobienie fragmentu z tranzystorem NMOS. Zmiany, przedstawiono na rysunku 2.10. Odcięto źródło tranzystora, a następnie zwarto je do masy. Tak samo jak w opisany wcześniej przypadku, dodano rezystor ściągający bramkę do masy. Między dren a zasilanie, dodany został rezystor o wartości $1\text{k}\Omega$. W tej konfiguracji, układ wyłączany byłby stanem wysokim, co rozwiązało w przykładzie poniżej. Niestety, przerobienie układu w ten sposób nie przyniosło zamierzzonego efektu. Mała ilość miejsca wymusiła ręczne wlutowanie rezystora fizycznie pod tranzystor, a jednocześnie prawdopodobnie między wszystkimi cięciami i padami występuowało jakieś niepożądane zwarcie. Niemniej jednak, w przypadku ponownej produkcji płytki, zmiana ta mogłaby być wprowadzona.



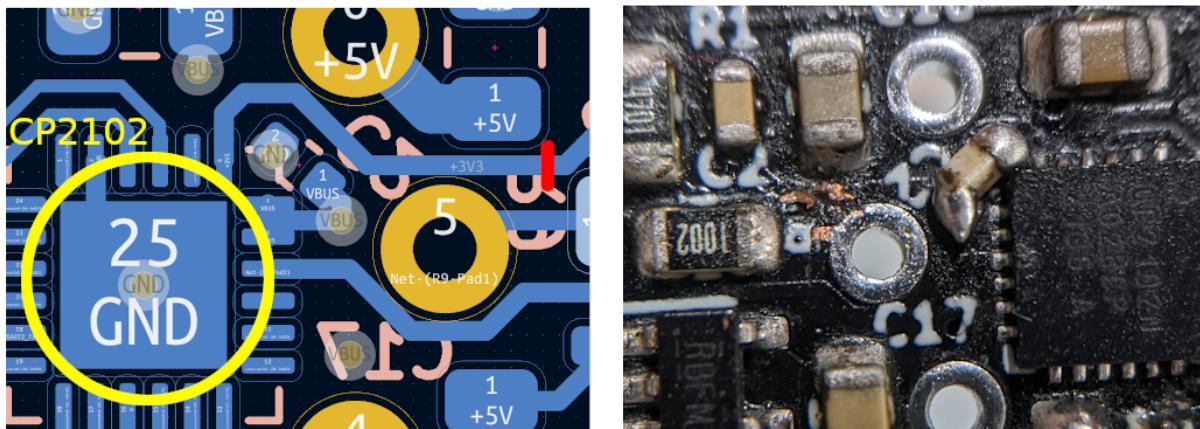
Rys. 2.10. Poprawiony schemat z użyciem tranzystora NMOS

Powyższe rozwiązanie, ma zasadniczą wadę - układ domyślnie jest włączony. Sprawia to, że użytkownik zanim napisze fragment programu mogący sterować układem włączania i wyłączania zasilania, pracuje z układem o pewnym napięciu. Układ pracując może wysyłać lub odbierać dane na pinach RX/TX, a także sterować pinem INT. Nie jest to zachowanie pożąданie. Z tego powodu, wykorzystując dostępne komponenty, zaproponowano również układ ze wzmacniaczem operacyjnym w konfiguracji wtórnika odwracającego. Układ, przedstawiono na schemacie 2.11. Podejście to, w przypadku możliwości skorzystania z innych komponentów, byłoby zupełnie nieopłacalne - w szczególności na dużą skalę. Dodatkowy układ w postaci wzmacniacza operacyjnego, podwyższyłby niepotrzebnie jednostkowy koszt produkcji płytki.



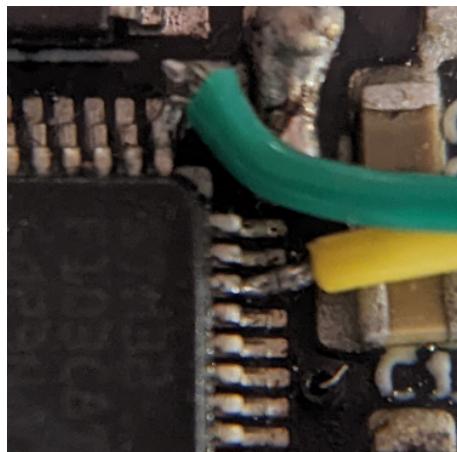
Rys. 2.11. Schemat poprawionego klucza NMOS, włączanego stanem wysokim

Kolejnym z problemów, tym razem wynikającym z niedoczytania dokumentacji, było połączenie wewnętrznego LDO konwertera CP2102, z linią 3.3V. W efekcie, mimo wyłączonej przetwornicy MIC5365, układ utrzymywał zasilanie. W tym przypadku, prostym i szybkim rozwiązaniem było odcięcie ścieżki, łączącej konwerter z wyjściem przetwornicy MIC5365. Zmiana, widoczna jest na obrazku 2.12. Na zdjęciu widać również delikatne naruszenie ścieżki obok, linii rezystora ściągającego MQ2 do masy. Niestety, powstało ono podczas cięcia, ze względu na brak miejsca między komponentami. Nie wpływa ono jednak na działanie układu. Na tym samym zdjęciu, widać również kondensator z bardzo dużą ilością cyny. Jest to kondensator w obudowie 0603, który nie mieściąc się na padach 0402 (niedostępny element), został dolutowany „rozszerzając” pad cyną. Było to spore wyzwanie, aby nie zewrzeć wspomnianych padów, z pinami CP2102.



Rys. 2.12. Przecięcie ścieżki 3.3V na wyjściu LDO CP2102

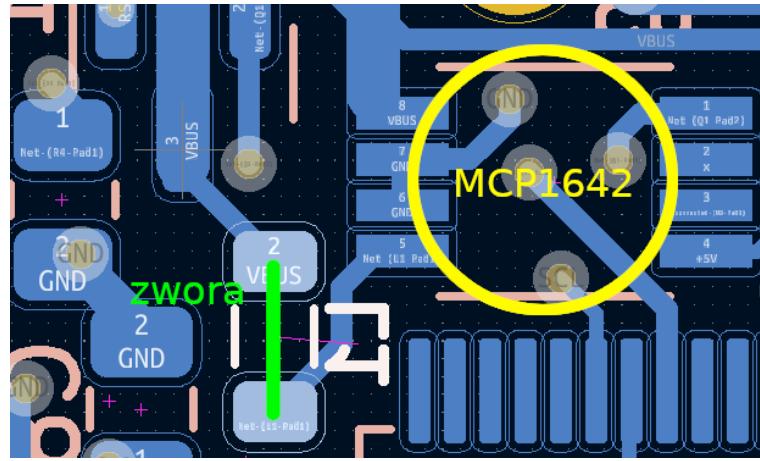
Jak wcześniej wspomniano, brak miejsca oraz inne czynniki, spowodowały, że nie wyprowadzono pinów programatora oraz INT. O ile jednokrotne przyłutowanie przewodów SWD bezpośrednio do procesora, nie było szczególnym problemem, o tyle uciążliwym stało się podczas innych przeróbek. Przewody, wielokrotnie odpadały, gdy płytka została nagrzana. Tym samym w kolejnym projekcie, zdecydowanie należy wyprowadzić pady programatora możliwie daleko od reszty komponentów. Zdjęcie 2.13 przedstawia przyłutowane przewody SWD. Pin przerwania (INT) nie został wyprowadzony, ponieważ w momencie tworzenia płytki, jego użycie nie było pewne. W trakcie tworzenia oprogramowania znaleziono dla niego zastosowanie i za pomocą przewodu połączono go z procesorem.



Rys. 2.13. Przewody SWD wlutowane w procesor. Zielony - SWDCLK, Żółty - SWDIO

Ostatni z napotkanych problemów, wynikał z prostego niedopatrzenia. Większość elementów pasywnych, użytych do zlutowania układu, zostało wzięte od prowadzącego. Niestety, nie posiadał on cewki 4.7uH. Ponieważ zamówienie zostało złożone wcześniej, nie opłacało się zamawiać nawet kilku cewek, ponieważ koszt dostawy znacząco przerósłby ich wkład w działanie układu. Biorąc pod uwagę, że cały

układ zasilany jest prądem stałym, a cewka dla prądu stałego stanowi zwarcie, w miejsce cewki wstawiono zworkę. Pozwoliło to na uruchomienie układu. Należy jednak pamiętać, że cewka w tym miejscu prawdopodobnie zabezpiecza układ przed prądem udarowym, występującym w momencie włączenia zasilania układu. Dlatego, na potrzeby prototypu można było usunąć cewkę, jednak w produkcie komercyjnym byłoby to niedopuszczalne.



Rys. 2.14. Zworka zastępująca cewkę

2.2. Oprogramowanie

2.2.1. Biblioteki peryferiów

Na etapie planowania projektów, podjęto decyzję o samodzielnych stworzeniu bibliotek do każdego z czujników. Takie podejście, narzucało konieczność ujednolicenia sposobu tworzenia kodu. Zdecydowano, że każda biblioteka, będzie korzystać z nakładki na funkcje HALa. Pozwoliło to wprowadzić obsługę błędów na wszystkich poziomach oprogramowania. Kod, tworzony był w sposób możliwie uniwersalny. Dzięki temu, płytka może być łatwo rozbudowana o kolejne czujniki, przy jednocześnie niewielkich zmianach w kodzie programu. Poniżej, krótko omówiono oprogramowanie każdego z czujników, oraz związane z nim biblioteki. Dla zachowania porządku, zdecydowano się rozdzielić w projekcie sterowniki dostarczane przez ST, pliki nagłówkowe oraz stworzone biblioteki. Zastosowanie opisanego w rozdziale 1 makefile, pozwoliło w łatwy sposób zdefiniować ścieżki dla kompilatora.

BMP280

Pierwszy z czujników, jest czujnikiem komunikującym się z mikroprocesorem z wykorzystaniem SPI (Serial Peripheral Interface). Pozwala on na wykonywanie pomiarów temperatury oraz ciśnienia. Stworzona biblioteka „*spi.c*” pozwala w łatwy sposób zainicjalizować peryferium w zdefiniowany sposób, odpowiedni dla stworzonej płytki. Osobna funkcja inicjalizująca piny SS (Slave Select) umożliwia szybkie dodanie kolejnych czujników działających z użyciem SPI. Wymaga to jedynie zainicjalizowania pinu. Dwie kolejne funkcje, odpowiadają za kolejno wysyłanie i odczytywanie danych przy użyciu SPI. Interesującym kierunkiem rozbudowy biblioteki, byłaby możliwość rejestracji pinów SS, co zdążyłoby z użytkownika konieczność samodzielnej kontroli nad używanym interfejsem oraz pinami. W przypadku tej wersji płytki, nie było to jednak konieczne. Biblioteka „*bmp280.c*” korzystająca z „*spi.c*”, odpowiada za podstawową obsługę czujnika BMP280. Poza podstawowymi możliwościami konfiguracji zgodnie z dokumentacją producenta, wykonuje wszelkie obliczenia, zdejmując z użytkownika konieczność przeliczenia danych z rejestrów.

STS3x-DIS

Kolejny z czujników to czujnik temperatury. Od poprzednika, różni się większą złożonością i lepszą precyją. Do komunikacji, wykorzystuje magistralę I²C. Stworzona do jego obsługi biblioteka, pozwala zmieniać jego konfigurację przy użyciu czytelnych stałych. Zapewnia nie tylko obsługę rejestrów czujnika, ale również kontrolę CRC otrzymywanych danych. Dzięki temu, użytkownik może być pewien otrzymywanych wartości.

MQ-2

Ostatni z czujników, to analogowy czujnik gazów. Stworzona do obsługi konwertera A/D inicjalizuje wybrane ADC mikoprocesora oraz konkretny kanał, według wartości odpowiednich dla stworzonej płytki. Posiada również funkcję pobrania próbki wybranego konwertera. Sama biblioteka obsługująca MQ-2, poza inicjalizacją i kalibracją sensora, pozwala użytkownikowi na pobranie przetworzonych

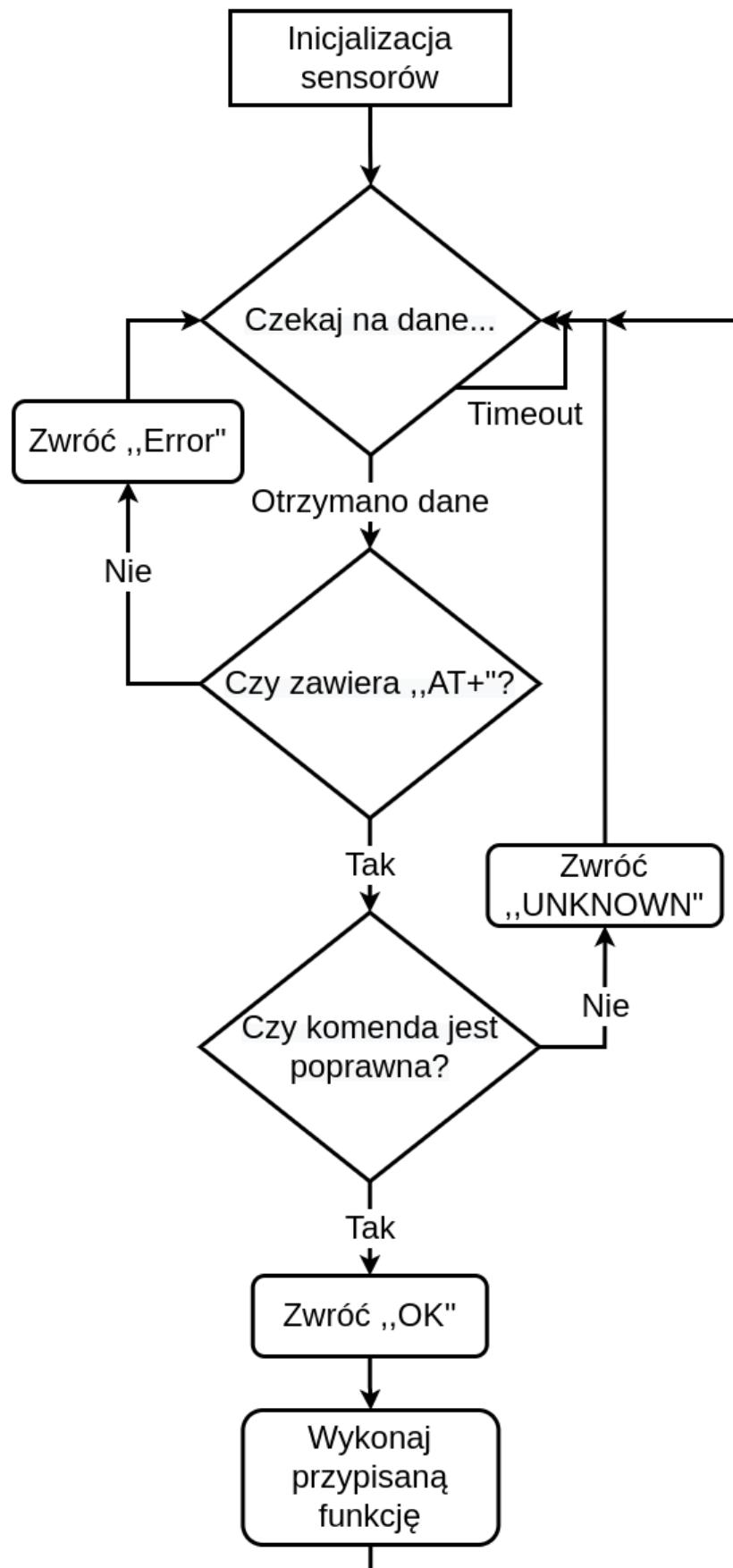
lub surowych danych z czujnika, dla wybranego gazu. Dodatkową funkcją, jest możliwość ustawienia alarmu dla wybranego gazu, na zdefiniowany przez użytkownika próg. Przekroczenie wybranej wartości wyzwoli alarm oraz wysteruje pin INT standardu mikroBUSTM. Aby umożliwić swobodne tworzenie funkcji przerwań czasowych w dowolnej ilości, stworzona została biblioteka „*timers.c*”. Przy użyciu jednokierunkowej, dynamicznej listy struktur oraz mechanizmu SysTick (przerwania wyzwalanego co 1ms) użytkownik może dodawać własne funkcje o predefiniowanym typie, a następnie wywołać je po upływie określonego czasu. Dodatkowa flaga, pozwalałączyć okresowe wyzwalanie danej funkcji. Zaletą tego podejścia jest przede wszystkim „nieograniczona” liczba liczników, jednak niepoprawne wykorzystanie biblioteki (np. funkcje o długim czasie wykonywania), może prowadzić do utraty precyzji liczników.

CP2102

Na potrzeby obsługi konwertera UART-USB, służącego do debugu kodu, stworzona została również prosta biblioteka do obsługi UARTu. Zawiera ona konfigurację peryferium oraz funkcję działającą jak printf, która zamiast na wyjście standardowe, kieruje dane na konkretny UART. W podobny sposób działa biblioteka obsługująca UART do komunikacji użytkownika z mikroprocesorem, wspomniana w podrozdziale 2.2.2. Zostały one rozdzielone ze względu na zachowanie czystości w kodzie.

2.2.2. API

Najważniejszym elementem projektu, spinającym wszystko w jedną całość, jest stworzone API (ang. Application Programming Interface). Pozwala ono użytkownikowi na komunikację z czujkami, przy użyciu interfejsu UART. Zdecydowano, że dla uproszczenia komunikacji, układ będzie zachowywać się podobnie do dostępnych na rynku modemów, wykorzystując komendy z przedrostkiem „AT+”. Diagram 2.15 przedstawia schemat działania API. Zostało ono zrealizowane przy użyciu stałych tablic struktur, zawierających komendę oraz związany z nią callback, czyli wywołanie odpowiedniej funkcji bibliotek sensora. Dla ułatwienia obsługi oraz lepszej czytelności kodu, każdy z czujników posiada własną tablicę wywołań. Tym samym, działanie całego API, to tak naprawdę jedna pętla, wywoływana po wykryciu przez UART komendy „AT+”. Wyszukuje ona komendę na odpowiedniej liście, a następnie wywołuje związaną z nią funkcję. Takie podejście, pozwala w bardzo łatwy sposób rozbudowywać kod w przypadku chęci wykorzystania kolejnych czujników. Jedyne wymagane zmiany, to dodanie kolejnej listy zawierającej komendy oraz powiązane callbacki. Zgodnie z diagramem 2.15, w przypadku niero- poznania komendy AT układ zwraca użytkownikowi informację UNKNOWN (ang. nieznany). **Kolejne komendy, rozpoznawane są przez znak kończący linię \r.** W przypadku niero- poznanych danych, przesłanych do mikroprocesora, zwrócona zostanie informacja ERROR (ang. błąd).



Rys. 2.15. Diagram działania API

2.2.3. Lista dostępnych funkcji

Ogólne

„AT+BMP\r” - Wybiera czujnik BMP280
„AT+STS\r” - Wybiera czujnik STS3x-DIS
„AT+MQ\r” - Wybiera czujnik MQ-2
„AT+BACK\r” - Powrót do menu wyboru czujników

BMP280

„AT+GETDATA\r” - odczytuje próbkę temperatury
„AT+SETCONFIG,[temp],[pressure],[standby],[mode]\r” - Ustawia konfigurację według parametrów z dokumentacji producenta: [temp]- temperature_sampling, [pressure]- pressure_sampling, [standby]- standby_period, [mode]- running_mode. Eg. AT+SETCONFIG,128,16,0,3\r
„AT+WAITRDY\r” - Czeka na flagę RDY rejestracji statusowej. Timeout = 20s.
„AT+SREST\r” - Programowy reset czujnika
„AT+BACK\r” - Powrót do menu wyboru czujników

MQ-2

„AT+GESAMPLE\r” - Odczytuje surową wartość ADC.
„AT+GETLPG\r” - Odczytuje przeliczone stężenie LPG.
„AT+GETCO\r” - Odczytuje przeliczone stężenie CO.
„AT+GETSMOKE\r” - Odczytuje przeliczone stężenie dymu.
„AT+SETTRACKING,[timeout],[gas],[threshold]\r” - Ustawia śledzenie stężenia wybranego gazu. Po wyzwoleniu alarmu, pin INT ustawiany jest w stan niski. Przerwanie czyszczone jest w momencie odczytu dowolnych danych. [timeout]- Okres między próbkami, [gas]- wybrany gaz (0-LPG, 1-CO, 2-SMOKE), [threshold]- Wartość progowa alarmu.
„AT+STOPTRACKING\r” - Przerywa śledzenie gazu. Czyści przerwanie, jeśli wystąpiło.
„AT+BACK\r” - Powrót do menu wyboru czujników

STS3x-DIS

„AT+GETDATA\r” - Odczytuje próbkę temperatury
„AT+SETREPL\r” - Ustawia dokładność na niską
„AT+SETREPM\r” - Ustawia dokładność na średnią
„AT+SETREPH\r” - Ustawia dokładność na wysoką
„AT+SETHEATON\r” - Włącza wewnętrzną grzałkę
„AT+SETHEATOFF\r” - Wyłącza wewnętrzną grzałkę
„AT+SRESET\r” - Programowy reset czujnika
„AT+BACK\r” - Powrót do menu wyboru czujników

2.2.4. System kontroli wersji

Jak wspomniano we wstępie, podczas projektu korzystano z systemu kontroli wersji Git, na platformie Github. Na jego potrzeby, stworzono dwa repozytoria:

Software

https://github.com/RadoslawSajdak/Sensors_in_embedded_systems_software

PCB

https://github.com/RadoslawSajdak/Sensors_in_embedded_systems_PCB

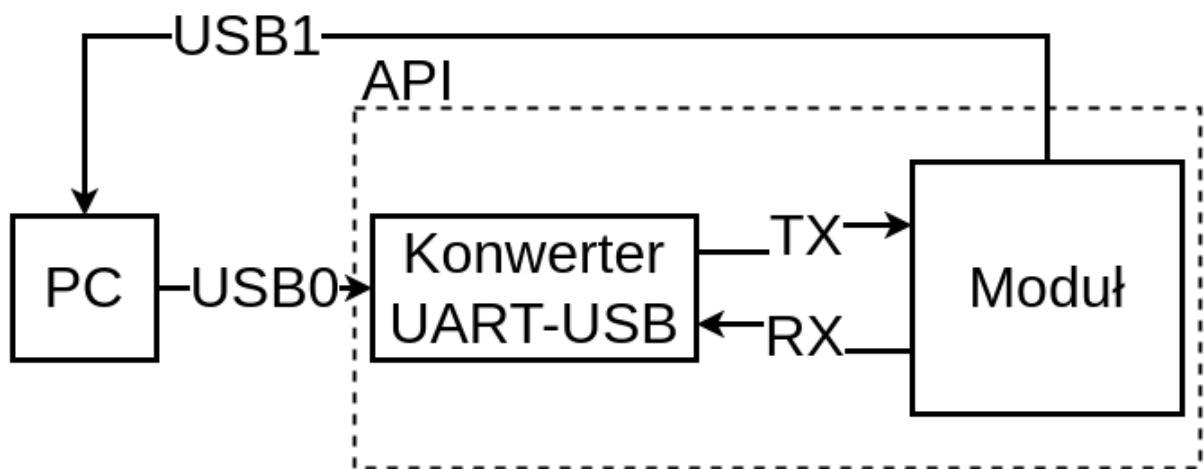
W przypadku pierwszego z nich:

- Wykonano 12 pull requestów
- Stworzono 20 gałęzi
- Wykonano 81 commitów
- Napisano 2080 linii kodu + pliki konfiguracyjne

Skorzystanie z platformy Github pozwoliło przede wszystkim na pełną kontrolę nad kodem opisanym przez powyższe liczby. W trakcie wykonywania projektu, był on nieustannie sprawdzany pod kątem poprawności składni, dobrych praktyk w tworzeniu kodu, a także testowany manualnie. Możliwość przejrzystej dyskusji, pozwoliła wszystkim członkom zespołu nauczyć się wielu nowych technik, a także spojrzeć na kod z cudzej perspektywy. Takie podejście, zdecydowanie jest dobrą praktyką, pozwalającą uniknąć wielu godzin poszukiwania błędów. W swojej książce *Pragmatyczny programista. Od czeladnika do mistrza*. Andrew Hunt oraz David Thomas piszą: „Zawsze należy stosować system kontroli kodu źródłowego. Zawsze. Nawet jeśli stanowimy jednoosobowy zespół i jeśli cały projekt zajmuje zaledwie tydzień. Kontroli kodu źródłowego, powinno podlegać dosłownie wszystko”[11].

3. Prezentacja działania

Na potrzeby prezentacji działania urządzenia, stworzono przedstawiony na diagramie 3.1 układ. Układ PC oraz konwertera UART-USB podłączonego do USB0, symulował mikroprocesor z kodem użytkownika. Pozwoliło to na dwukierunkowe przetestowanie działania API oraz samych sensorów. Dodatkowo, dzięki obecnemu na płytce konwerterowi CP2102, przy użyciu USB1 obserwowano tzw. logi programu. W głównej mierze, dawały one informację na temat tego, czy dane zostały poprawnie odebrane. Jest to jednak pole do łatwej rozbudowy, ponieważ obecna struktura kodu pozwala wysyłać informacje nt. przebiegu dowolnych funkcji.



Rys. 3.1. Diagram aplikacji testowej

Po podłączeniu komponentów zgodnie z powyższym schematem, przy użyciu monitora portu szeregowego Cutecom, wykonano sekwencję komend testujących działanie API. Uporządkowany zapis z monitora, przedstawia obrazek 3.2. Sekwencję rozpoczęto od wyboru czujnika BMP280. Komendy 1-3, opisują prawidłową komunikację z czujnikiem oraz pobranie temperatury. Należy zaznaczyć, że liczba zwracana jest jako 4 bajty danych w formacie fixed point. W celu ich interpretacji, należałoby więc otrzymaną wartość podzielić przez tysiąc. Otrzymana wtedy temperatura, wynosi wówczas 28.631 stopni celsjusza. Czwarta komenda, to przykład wywołanej w niepoprawnym miejscu komendy AT. Ponieważ „AT+STS” nie znajduje się na liście komend sensora BMP280, urządzenie zwróciło informację UNKNOWN. Aby wybrany sensor został przełączony poprawnie, należy postępować zgodnie z krokami 5-7. Komenda numer 8, to przykład wykorzystania nieistniejącej komendy AT. Kolejna, „WRONGMSG” to

przykład zupełnie niepoprawnych danych. Widać, że układ zgodnie z oczekiwaniami zwrócił *ERROR*. Komendą numer 10, pobrano odczyt temperatury z czujnika STS3x-DIS. Wynosi on 37.968 stopni celsjusza. Należy zauważyć, że jest on o około 9 stopni wyższy niż w przypadku odczytu z BMP280. Wynika to najprawdopodobniej z jego umiejscowienia na płytce, ponieważ znajduje się on nad główną linią zasilania i przetwornicą 5V. Wiadomości 11-15 pokazują działającą poprawnie możliwość wpływania na ustawienia czujnika. Uruchomiona wewnętrzna grzałka podniosła temperaturę względem zmierzonej wcześniej, a jej wyłączenie spowodowało spadek mierzonej wartości. W przypadku czujnika MQ-2, komendami 16-19 pobrano aktualną wartość napięcia na pinie związanym z przetwornikiem A/C.

```

GOT a message! BMP 1
[15:24:50:706] GOT a message! SRESET 2
[15:24:56:923] GOT a message! GETDATA 3
[15:25:23:606] GOT a message! STS 4
[15:25:30:835] GOT a message! BACK 5
[15:25:39:400] GOT a message! STS 6
[15:25:50:680] GOT a message! SETREPH 7
[15:25:57:394] GOT a message! SETREPXD 8
[15:26:16:595] GOT a message! GETDATA 10
[15:27:41:207] GOT a message! SETHEATON 11
[15:27:49:840] GOT a message! GETDATA 12
[15:27:57:959] GOT a message! SETHEATOFF 13
[15:28:26:022] GOT a message! GETDATA 14
[15:28:31:866] GOT a message! BACK 15
[15:28:46:060] GOT a message! MQ 16
[15:28:56:906] GOT a message! GETSAMPLE 17
[15:29:09:883] GOT a message! GETSAMPLE 18
[15:30:14:206] GOT a message! BACK 19

AT+BMP
AT+SRESET
AT+GETDATA
AT+STS
AT+BACK
AT+STS
AT+SETREPH
AT+SETREPXD
WRONGMSG 9

AT+GETDATA
AT+SETHEATON
AT+GETDATA
AT+SETHEATOFF
AT+GETDATA
AT+BACK
AT+MQ
AT+GETSAMPLE
AT+GETSAMPLE
AT+BACK

Input: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
OKOKOK28631UNKNOWNOKOKOKUNKNOWNERROROK37968OKOK38622OKOK38027OKOKOK2954OK2070OK

```

Rys. 3.2. Zapis programu cutecom. Po lewej konsola DEBUG, po prawej komendy wysypane do mikroprocesora, na dole odpowiedzi na kolejne komendy

Dodatkową funkcjonalnością którą należało przetestować, było ustawianie konfiguracji czujników przy użyciu parametrów przekazywanych do mikroprocesora. Na obrazku 3.3 widać, że czujnik BMP280 poprawnie przyjmuje konfigurację, gdy zawiera ona cztery parametry (zwraca „OK0”). Po ustawieniu nowej konfiguracji, poprawnie zwraca wartości ciśnienia i temperatury („OK[value]”). Po przesłaniu niepoprawnej komendy *AT+SETCONFIG,96,20,0,1,7,7,7,7,7* zawierającej zbyt dużą ilość argumentów, układ zgodnie z oczekiwaniami zwrócił wiadomość „OK1”. *OK* oznacza tutaj poprawne rozpoznanie komendy, natomiast *1* niepowodzenie w jej wykonaniu. W tym miejscu można wskazać kolejne miejsce na rozbudowę oprogramowania. Opisywane wcześniej ustandaryzowane w obrębie programu kody błędy, mogłyby być zwracane do użytkownika, informując go o problemie który wystąpił.

```
GOT a message! BMP
[16:54:31:403] GOT a message! GETTEMP
[16:54:39:937] GOT a message! SETCONFIG
[16:54:39:937] Parsed params: 96 20 0 1 0
[16:54:43:271] GOT a message! GETTEMP
[16:54:48:639] GOT a message! GETPRESSURE
[16:54:51:964] GOT a message! GETPRESSURE
[17:01:54:445] GOT a message! SETCONFIGRE
[17:01:54:445] Parsed params: 96 20 0 1 7

AT+BMP
AT+SETCONFIG,96,20,0,1
AT+GETTEMP
AT+GETPRESSURE
AT+SETCONFIG,96,20,0,1,7,7,7,7,7,7

Input:
OK|OK28376|OK|OK28313|OK98759|OK98566|OK1
```

Rys. 3.3. Zapis programu cutecom. Po lewej konsola DEBUG, po prawej komendy wysyłane do mikroprocesora, na dole odpowiedzi na kolejne komendy

4. Wnioski

No na pewno napisać, że inaczej bym PCBka zrobił. Że dużo było czytania dokumentacji, trzeba było sobie przypomnieć sporo z elektroniki, a i tak pojawiły się błędy. Na pewno, że 0805 są za duże. Że robienie review pomaga obu stronom czegoś się nauczyć. Że sensory na PCB muszą być przemyślane (MQ + temperatury). Że mimo tego, że N osób widziało schematy i layout, to błędy przeszły produkcję i dużo czasu straciło się na debug zasilaczami, oscyloskopami itp.. Ale trzeba też powiedzieć, że robiliśmy projekt 4fun i dużo się nauczyliśmy.

Bibliografia

- [1] MikroElektronika. *MikroBUS standard specification*.
- [2] ST. *STM32F103x8 Datasheet*.
- [3] Sensiron. *STS3x-DIS Datasheet*.
- [4] USB organization. *USB 2.0 specification*.
- [5] HANWEI ELECTRONICS CO. *MQ-2 Gas Sensor Datasheet*.
- [6] MICREL. *MIC5365-3.3YC5 Datasheet*.
- [7] BOSCH. *BMP280 Digital Pressure Sensor Datasheet*.
- [8] Silicon Labs. *CP2102 Datasheet*.
- [9] JLCPCB. *PCB Capabilities*.
- [10] MICROCHIP. *MCP1642B Datasheet*.
- [11] Hunt A. Thomas D. *Pragmatyczny programista. Od czeladnika do mistrza*. Wydawnictwo Helion, 2021, KINDLE LOC 2757.